UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

" DISEÑO Y CONSTRUCCION DE UNA TARJETA PROGRAMADORA, UN ENSAMBLADOR Y UN SIMULADOR PARA LOS MICROCONTROLADORES MC68705P3, R3 Y U3 "

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A N:

LUZ MARIA CASTILLO JIMENEZ

GRACIELA SANTILLAN ALCALA

ASESOR: M.I. JUAN CARLOS ROA BEIZA

CIDUAD UNIVERSITARIA ENERO DE 1997

Agradecimientos

Agradecemos muy sinceramente el apoyo para la realixación de esta tesis a todos los profesores que nos dieron clase durante nuestra carrera muy especialmente a:

M.C. Juan Carlos Roa Beixa director de nuestra tesis, por su paciencia confianza y apoyo incondicional.

Ing. Enrique Arenas Sánchex por todas sus enseñanxas, por su apoyo y confianxa.

A la familia Martínex Larco por su enorme paciencia, apoyo y colaboración.

A nuestras mamás por su fe, cariño, paciencia y consuelo.

Y a todos los integrantes del Departamento de Cálculo y a los miembros de la Jefatura de la División de Ciencias Básicas y muy especialmente a nuestra

Facultad de Ingeniería
y a nuestra
Universidad.

Esperamos reintegrarles a todos algo de lo mucho que nos dieron.

Índice

Capítulo 2. Planteamiento del Problema

Capítulo 3. Diseño y Construcción

Apéndice D

Glosario

Bibliografía

Introducción

Introducción

Objetivos

Diseñar y construir un prototipo que permita programar adecuadamente los MC68705P3, R3 y U3 utilizando una memoria EPROM y vaciando ésta a través de una rutina de software hacia el microcontrolador.

Objetivo del desarrollo de una Tarjeta Programadora

Diseñar y construir una tarjeta programadora para los microcontroladores 68705P3, R3 y U3. La tarjeta funcionará en modo Stand-Alone, es decir que no será esclava de un sistema PC, esto permitirá desarrollar prototipos de una forma rápida, ya que el microcontrolador se programará una vez que el programa haya sido evaluado y funcione perfectamente.

Objetivo del desarrollo de un Ensamblador

Se elaborará un ensamblador especialmente diseñado para el código que utilizan los microcontroladores MC68705P3, R3 y U3, utilizando las ventajas de los códigos

de las instrucciones que utilizan los MCU's fabricados por Motorola, algunas de las cuales son:

Las instrucciones son de un mismo tamaño.
Los operandos pueden ser de tres bytes.
Manejo de modos de direccionamiento.

Objetivo del desarrollo de un Simulador

Se diseñará un simulador que tomará el código objeto resultante del proceso de ensamblaje y por medio de un ambiente agradable se simulará la acción que realiza el microcontrolador al ejecutar un programa.

Propósito de la tesis

El propósito principal de esta tesis es la de proporcionar una herramienta más en la teoría de microcomputadoras, dado que proporcionamos un ensamblador y un simulador para una familia de microcontroladores, que ayudará a entender cómo funcionan y podrán realizar aplicaciones de acuerdo a las necesidades que se tengan.

Otro de los propósitos es el ambiente en que fueron desarrollados y en que son utilizados el ensamblador y el simulador, dado que en la actualidad casi la mayoría de las aplicaciones en la industria están cambiando de plataforma a un ambiente en

Windows, nosotros quisimos desarrollar los dos programas en dicho ambiente, el lenguaje que escogimos fue Turbo C++ orientado a objetos, dadas las características y ventajas de éste y el paquete en que lo desarrollamos es Visual C++ v.1 por el apoyo muy fuerte que da con la Microsoft Fundation Class para desarrollar paquetes con ambiente Windows.

Además la elección de los microcontroladores se debió a las características de estos, dado que manejan los componentes principales para el aprendizaje de estos (timer, oscilador, convertidor A/D, puertos E/S), agregando que son pequeños, de bajo costo y fáciles de conseguir. Se proporciona además el diagrama de la tarjeta programadora con la cual se programan los MCU y se comentan los cambios que se le hicieron a la tableta original que proporciona Motorola en la compra de su tarjeta programadora para estos MCU.

Propósito del diseño de un ensamblador para los MCU's 68705P3, R3 y U3

Este ensamblador simplificará el desarrollo de programas para los microcontroladores mencionados ya que utilizará secuencias que faciliten las operaciones comúnmente usadas en la programación de los MCU.

Además será una herramienta valiosa en el aprendizaje de los alumnos de las materias en que se utilicen microcontroladores.

El programa fuente se genera en mnemónicos en cualquier procesador de texto ASCII, se carga en el ensamblador automáticamente efectuándose el ensamblaje de dos pasadas, y al final de éste se genera un reporte de errores de sintaxis.

La pantalla de trabajo del ensamblador es en ambiente Windows donde las opciones son elegidas a través de íconos o botones de acción.

El ensamblador genera archivos de trabajo como son: el programa objeto y el archivo de impresión (que contiene el archivo fuente y el objeto, uno al lado del otro), que son utilizados en los MCU indicados anteriormente.

Propósito del diseño de un ensamblador para los MCU's 68705P3, R3 y U3

El simulador tomará y ejecutará el programa objeto resultante del ensamblaje y simulará la acción que efectúa el microcontrolador, ya que mostrará los registros internos, el mapa de memoria y los puertos utilizados en él. Dando una visión muy clara de cómo el microcontrolador ejecuta las instrucciones que han sido programadas y que el usuario de éste sistema observe si se están realizando adecuadamente las acciones que el programó, antes de ser cargadas en el microcontrolador.

El simulador también tendrá un ambiente de trabajo Windows donde se manejarán las opciones a través de botones de selección.

Los valores se actualizarán dinamicamente conforme se vaya ejecutando cada una de las instrucciones. Se muestra una pantalla en donde se presentan los valores que toman los registros internos conforme se ejecutan cada una de las instrucciones del programa simulado.

Todo esto será desarrollado con el lenguaje Turbo C++ orientado a objeto que proporciona Visual C++. Esto permitirá que se obtenga un sistema visualmente atractivo al usuario, amigable y de fácil manejo.

Capítulo 1

Antecedentes Básicos

Capítulo 1
Antecedentes Básicos

1.1 Clasificación de los lenguajes de programación

Una revolución tiene lugar en los lenguajes de programación. La revolución es necesaria. Se necesitan instrucciones de computadora mucho más fáciles que las del pasado.

La nueva generación de lenguajes de computación necesitan mucho más potencia que las generaciones previas, así los resultados se pueden obtener mucho más rápido, y se necesitan lenguajes mas amigables al usuario.

1.1.1 Generaciones de los lenguajes de programación

1.1.1.1 Lenguajes de Primera generación (1GL)

La primera generación de lenguajes de computación fue la de **lenguaje de máquina**. En estos no había intérprete o compilador para trasladar el lenguajes a una forma diferente. Las primeras computadoras se programaron con una notación binaria[1], por ejemplo:

[1]Martin James/Fourth Generation Languages

011011 000000 000000 000001 110101

esta instrucción puede ser "limpia el acumulador y suma el contenido a la localidad 117".

Esto resulta muy complejo de programar sin incurrir en errores. La situación mejoró mediante el uso de mnemónicos (códigos que representan operaciones). Así la instrucción anterior sería:

CLA 000000 000000 000001 110101

Más tarde los números se escribieron en forma más conocida con el fin de almacenarse en localidades o registros.

CLA 0 0 117

1.1.1.2 Lenguajes de segunda generación (2GL)

Esta generación que comienza a mediados de 1950, estuvo representada, principalmente, por el lenguaje ensamblador simbólico, en el cual se usan direcciones simbólicas para representar direcciones físicas, consideremos la instrucción del ejemplo anterior, la cual quedaría CLA SALARIO, donde SALARIO representa la localidad de memoria en la cual se almacena la variable salario. El direccionamiento simbólico fue un gran progreso porque ahora cuando las localidades físicas de las

variables o instrucciones deben cambiarse (y estas se cambian constantemente), el programador no tiene que re-introducir las nuevas direcciones físicas. Los primeros lenguajes ensambladores optimizaron la posición de las instrucciones programadas mediante un tambor que almacenaba las instrucciones[2].

1.1.1.3 Lenguajes de tercera generación (3GL)

Estos lenguajes comenzaron a utilizarse en los años 1960 y fueron llamados **"lenguajes de alto nivel"**. Algunos de estos estaban enfocados a trabajos científicos tal como ALGOL y FORTRAN, otros a trabajos comerciales como COBOL. Algunos lenguajes como PL/I y después ADA, abarcan ambos tipos de trabajo, científico y comercial.

Con la tercera generación el lenguaje comenzó a ser independiente del hardware. Un programador puede realizar programas sin necesidad de conocer el conjunto de instrucciones de máquina y registros. En caso de que el programador desee optimizar la ejecución del programa sólo necesita conocer algunas características de la máquina. Debido a esta independencia con el hardware los programas pueden ejecutarse en diferentes máquinas.

Los lenguajes de tercera generación dan un paso hacia el lenguaje del usuario, ya que utilizan palabras en inglés y fórmulas expresadas en notación matemática, lo cual es más fácil de escribir.

---

[2]Martin James/Fourth Generation Languages

Es más fácil escribir:

$$X=(A+B)/(C+D)$$

que:

```
CLA C
ADD D
STD Y
CLA A
ADD B
DIV Y
STD X
```

Los lenguajes de tercera generación necesitan un vasto número de líneas de código y fueron diseñados para profesionales más que para usuarios finales. Consume tiempo depurarlos y su modificación en sistemas complejos es muy difícil.

1.1.1.4 Lenguajes de cuarta generación (4GL)

Fueron diseñados en respuesta a los problemas de los lenguajes de tercera generación para alcanzar los siguientes objetivos:

- Para acelerar el proceso de desarrollo de la aplicación.
- Hacer aplicaciones fáciles y rápidas de cambiar , reduciendo así costos de mantenimiento.
- Minimizar los problemas de depuración.

- Generar un código libre de errores con las expresiones de alto nivel.
- Hacer lenguajes amigables al usuario y que los usuarios finales puedan resolver sus problemas mediante el uso de la computadora.

Los 4GL permiten generar algunas aplicaciones con menos líneas de código que las que se necesitarían con COBOL, PL/I, ADA u otro. Se les ha llamado lenguajes de alta productividad[3]; emplean una diversidad de mecanismos en forma de panel, interacción con la pantalla y gráficos asistidos por computadora.

Muchos lenguajes de cuarta generación dependen de una base de datos y esta de un diccionario de datos o directorio. El directorio contiene formatos de pantalla, de reportes, estructuras de diálogo, asociaciones con varios datos, validación de inspección, controles de seguridad, autorización para leer o modificar datos, cálculos que se usan para crear campos derivados, rangos permisibles y relación entre datos. La extensión del diccionario que contiene reglas de transacción lógica se conoce como enciclopedia[3].

1.1.1.5 Lenguajes de quinta generación (5GL)

El término quinta generación generalmente se refiere a sistemas que usan disciplinas originadas en un campo de inteligencia artificial[4] tales como:

- Conocimiento básico de sistemas

[3]Martin James/Fourth Generation Languages

[4]Martin James/Fourth Generation Languages

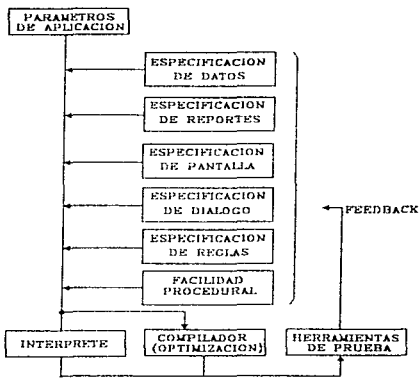


Figura 1.1.2 Componentes de un 4GL para realizar rutinas de aplicación

- Sistemas expertos
- Inferencia de máquinas
- Procesamiento de lenguajes humanos

Los sistemas de quinta generación involucran un complejo conocimiento de lo que una máquina puede hacer. En algunos casos, este procesamiento de inferencia se usa para realizar tareas que, siendo complejas, son triviales para los humanos tal como

aprender a hablar, la visión y el lenguaje humano. En algunas áreas especializadas del conocimiento, la máquina puede aparentar ser experta y mejor que los mismos humanos.

Los lenguajes diseñados para estas áreas son denominados lenguajes de quinta generación. Además estos lenguajes emplean un alto procesamiento en paralelo para trabajar en un problema simultáneamente.

Las generaciones de lenguajes de programación poseen ciertas características que permiten reclasificarlos de la siguiente forma:

1.1.2. Reclasificación de las generaciones de lenguajes de programación

1.1.2.1 Lenguajes procedurales

Son lenguajes que dan un conjunto de instrucciones para que se ejecuten por una computadora en una secuencia específica[5]. Esta secuencia puede variar, dependiendo de las condiciones de la prueba. Se pueden ejecutar grupos de instrucciones repetidamente.

Los lenguajes 1GL, 2GL y 3GL son de este tipo. Un lenguaje procedural especifica como se realiza la tarea deseada. El programador debe dar las instrucciones detalladas para que se realice cada acción.

Un lenguaje de este tipo no permite instrucciones GOTO e IF(condición) GOTO.

[5]Martin James/Fourth Generation Languages

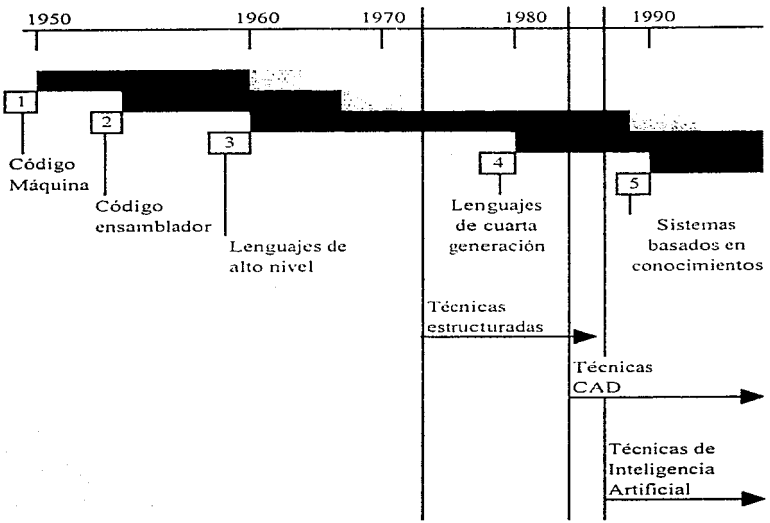


Figura Las generaciones de lenguajes de programación

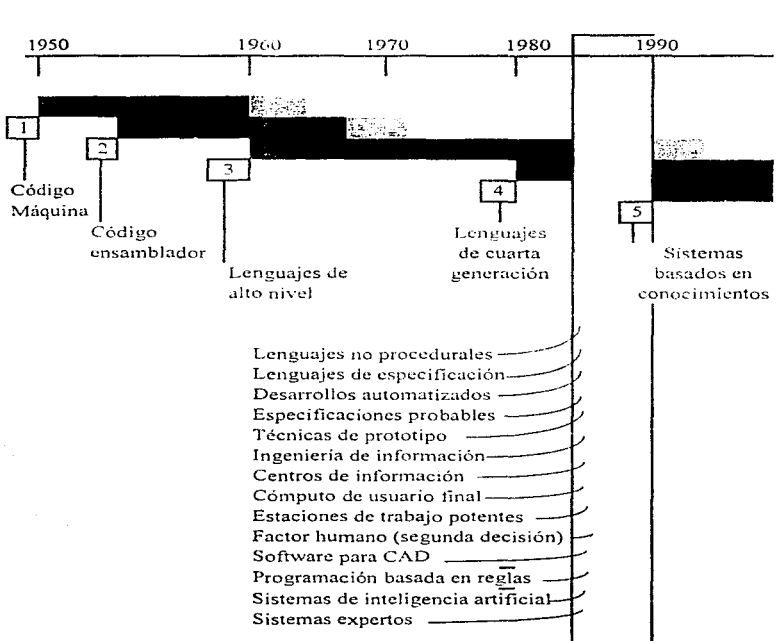


Figura Nuevas e importantes técnicas para la aplicación en el desarrollo de programas. Estas técnicas afectaron fuertemente los aspectos deseables de los lenguajes de construcción.

CONSTRUCTORES DE NO REPETICION

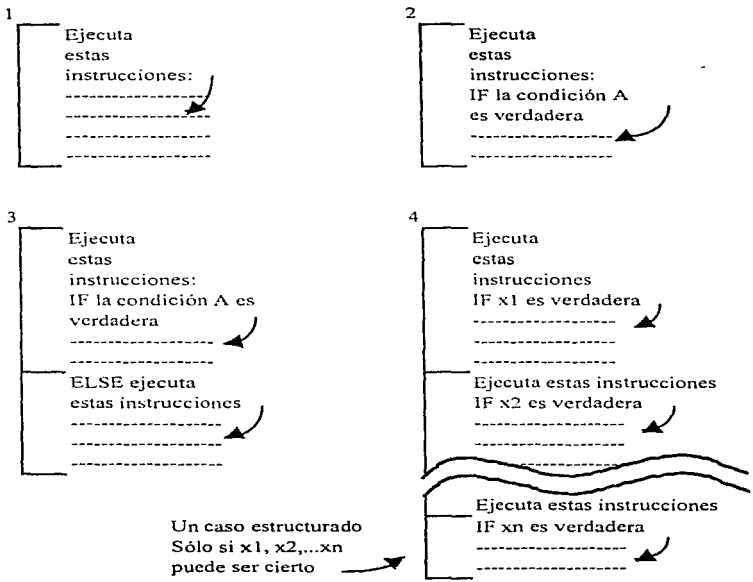


Figura Constructores de un lenguajes procedural

1.1.2.2 Lenguajes no procedurales

Son lenguajes que describen que resultados se desean pero no especifican la secuencia de pasos (procedimientos) a seguir[6] . Algunos lenguajes de alto nivel son de este tipo.

Por ejemplo

LIST BY CUSTOMER AVERAGE (INVOICE TOTAL)

es un programa completo. El software decide cómo formatear la lista, cómo saltar páginas, cómo numerar las páginas, cómo clasificar y cómo calcular el promedio.

El usuario ve el despliegue en la pantalla y puede manipularlo.

Muchos lenguajes no procedurales pueden manejar sólo limitadas clases de aplicaciones.

Mientras muchos lenguajes son puramente procedurales o puramente no procedurales, otros combinan ambos tipos. Esto es generalmente lo más deseable, porque operaciones no procedurales son más rápidas y simplifican el uso del lenguaje, mientras que si se aplican lenguajes procedurales se extiende el rango de aplicación y hay mayor flexibilidad de manipulación lógica.

1.1.2.3 Lenguajes declarativos

[6]Martin James/Fourth Generation Languages

Son lenguajes que declaran un conjunto de datos y que permiten la declaración de dudas o problemas que usan estos datos[7]. No especifican completamente la secuencia de pasos (procedimientos) para manejar la duda o problema. Un lenguaje declarativo es una forma de lenguaje no procedural.
Por ejemplo:

P=1*100-i_N-(20)

el usuario no tiene que indicar la secuencia de pasos para que el cálculo se ejecute.

1.1.2.4 Lenguajes basados en reglas

Son lenguajes con los cuales un conjunto de datos y reglas pueden describirse[7]. Estos datos y reglas pueden usarse para responder preguntas o resolver problemas.
Por ejemplo:

female(mary)
male(john)
likes(john, mary)
book(future shock, toffler, random house)

Las reglas se expresan así:

[7]Martin James/Fourth Generation Languages

sister (x,y):-female(x), parent(x,z), parent(y,z)

x es la hermana de y si x es female y el pariente de x es el mismo que el de y.

1.1.2.5 Lenguajes de programación funcional

Son lenguajes en los cuales las expresiones están compuestas de llamadas a funciones[8], estas son las unidades básicas de estos programas.

Por ejemplo:

La función SUB 1 N retorna (N-1)

La función FACTORIAL (SUB 1 N) retorna (N-1)!

1.1.2.6 Lenguajes de Diálogo o Lenguajes interactivos

Son lenguajes en los cuales un programa se crea como resultado de un diálogo interactivo entre el usuario y el software[8].

Por ejemplo:

>Reportex

Nombre de la lista>

Criterio para incluir datos en la lista>

[8]Martin James/Fourth Generation Languages

Una gran variedad de formas de diálogo se usan en diferentes lenguajes de cuarta generación.

1.1.2.7 Lenguajes Interactivos Full-Screen

Son lenguajes en los cuales el usuario interactua con una pantalla, rellenando términos, apuntando términos, moviendo términos, modificando los valores por omisión, etc.

La interacción full-screen es rápida y más poderosa que la interacción orientada en línea.

1.1.2.8 Lenguajes de Programación Gráfica

Son lenguajes en los cuales los programas se hacen interactivamente con técnicas gráficas; la lógica, estructuras de control, árboles de decisión, navegación de bases de datos, procesamiento de reglas, y otras más pueden expresarse claramente con diagramas. Los diagramas pueden hacerse en una pantalla de computadora. Un tipo de diagrama puede convertirse en otro. Los diagramas pueden ligarse a un diccionario de datos y usarse para generar código.

Para generar código a partir de diagramas, la técnica de diagramación ha de ser convenientemente rigurosa. La programación gráfica puede ser rápida y poderosa.

Los lenguajes de tercera generación no fueron **amigables al usuario.** Estos asumieron que el programador debía dedicarse devotamente a aprender la sintaxis del

lenguaje. Esta sintaxis no tenía nada que ver con el lenguaje humano, aunque permitía el uso de palabras en inglés.

Cuando se utiliza un código no procedural, la situación cambia. Una vasta diversidad de técnicas son posibles para decirle a una computadora que hacer más que como hacerlo. Muchas de esas técnicas pueden ser más amigables al usuario.

En cuanto a sus **límites de funcionalidad** los 4GL varían enormemente en su poder y capacidad. Algunos son meramente lenguajes de preguntas, otros generadores de reportes o de gráficos, otros pueden generar aplicaciones completas, algunos son lenguajes de programación de alto nivel. Algunos lenguajes pueden usarse por usuarios finales, o por analistas de sistemas asistiendo directamente a los usuarios finales.

Existe un gran debate acerca de si los lenguajes de pregunta, de generación de reportes y otros lenguajes de funciones limitadas pueden llamarse 4GL.

En lo referente a **restricción de opciones** los lenguajes de alto nivel eliminan opciones que eran válidas en lenguajes de bajo nivel. Esto puede ser más fácil de aprender y mantener, pero causa protestas de parte de los programadores de lenguajes de bajo nivel los cuales dicen que necesitan la "flexibilidad" de los lenguajes de bajo nivel.

En los 4GL se eliminan opciones que fomentan diseños muy particulares (misdesign), usando constructores que son fiables y constructores que son tan potentes como es posible para generar un código relativamente rápido y de simple acción para el desarrollador. El código y las acciones generadas usan los mejores principios de diseño.

Las **opciones por default** (por omisión) en los 4GL permiten que el usuario no necesariamente especifique todo. Un compilador o intérprete asume inteligentemente lo que el usuario necesita. Por ejemplo, si se selecciona automáticamente el formato de un reporte, las opciones por default ponen la imagen, tipos de caracteres para despliegue de gráficos, colocan etiquetas en los ejes o en una columna.

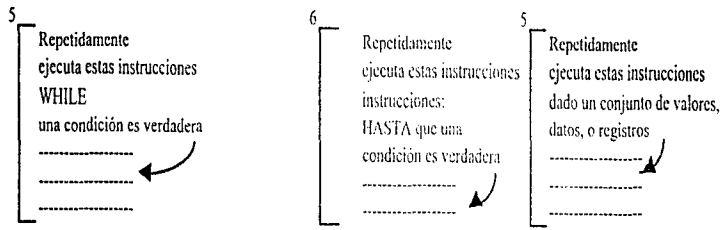
En caso de que el usuario desee modificarlos se le da una lista de los parámetros en la pantalla y el selecciona aquellos que desee modificar y le da los valores apropiados.

Uno de los avances más importantes en la evolución de los 4GL es el **evitar el uso de mnemónicos y sintaxis que sea confusa** para los programadores no profesionales. Los primeros lenguajes 4GL no evitaron la sintaxis no humana. Los humanos recuerdan el lenguaje humano y el de su profesión, pueden recordar gráficas y pueden leer gráficos de diagramas de flujo, pero no pueden recordar mnemónicas y sintaxis, la cual debe ser precisa y correcta y que además requiere mucho tiempo de aprendizaje y de práctica.

1.1.2.9 Lenguajes orientados a objetos

Los lenguajes orientados a objetos usan un par de primitivas diferentes: **objetos y mensajes**. Un objeto es un paquete de información y una descripción de como se manipula. Este contiene no solamente datos sino que también contiene un conjunto de funciones que permiten accesar a dichos datos. Cada objeto se comunica con otros objetos mediante el envío de mensajes. Cuando un objeto recibe un mensaje, este

CONSTRUCCIONES CON REPETICION



CONSTRUCCIONES BRANCH

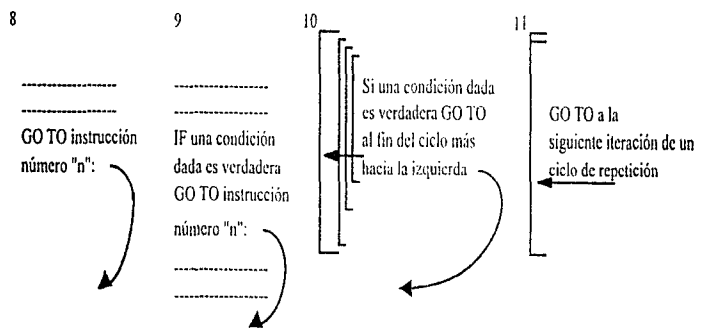


Figura Constructores de un lenguaje procedural

decide que hacer con él y usualmente envía mensajes a otros objetos. Así en lugar de aplicar funciones activas a datos pasivos, como un programa convencional, los mensajes dicen a los objetos como pensar y el objeto llama al código y decide como implementar la operación.

Los objetos están agrupados en clases que poseen propiedades de herencia. La herencia hace posible el uso de nuevas clases de objetos mediante una simple especialización de las clases existentes en lugar de comenzar desde cero.

1.2 Características y ventajas de la Programación Orientada a Objetos.

La programación orientada a objetos, u O.O.P. como es conocida, lleva el enfoque tradicional de la programación un paso más allá de la programación llamada procedural o de procedimientos, que se aplicó en los años ochenta.

La programación orientada a objetos proporciona un mejor vehículo de programación que ayuda a los programadores a alcanzar sus metas (aplicaciones terminadas y depuradas) con mayor rapidez que el enfoque de la programación de procedimiento al que han estado acostumbrados. En pocas palabras, la programación orientada a objetos hace activos los datos. En vez de esperar por un código que maneje las variables, las variables del programa saben cómo manejarse ellas mismas. Cuando es momento de imprimir el contenido de una variable, no se imprime la variable, sino

que se le dice a la variable que ella misma se imprima. Este método de programación es, de hecho, una forma muy natural de programar.

Cuando se programa sin usar la O.O.P. se está programando por procedimientos (un programa de procedimiento es un programa paso a paso que guía a la aplicación por una serie de instrucciones). Un programa de procedimiento ejecuta cada enunciado en el orden literal de sus comandos, incluso si esos comandos hacen que el programa ramifique en todas direcciones. El C, Pascal, QBasic y COBOL son ejemplos de lenguajes de programación de procedimiento.

El mundo actual de la computación no se lleva bien con los programas de procedimientos. Por ejemplo, las interfases de usuario gráficas (GUI), como el Windows de Microsoft, pueden ser programadas en lenguajes de procedimientos, pero es más adecuado escribir programas GUI con lenguajes orientados a objetos.

Los iconos, cuadros de diálogo y casillas de selección en la pantalla de una GUI se activan cuando sucede un evento, como la opresión del ratón o de alguna tecla. Cuando se escribe una aplicación GUI con un lenguaje de procedimientos, se tienen que hacer continuas pruebas para estos eventos, usando instrucciones de programación como enunciados switch gigantes del C. Como se mencionó anteriormente un lenguaje de programación orientado a objetos permite datos activos. Todos los elementos de la GUI en la pantalla se convierten en conceptos de datos activos en el programa.

En los lenguajes de procedimientos las variables tienen propiedades, una variable puede ser entera y guardar un número, otra variable puede ser una variable de cadena o un arreglo de caracteres. En los lenguajes O.O.P. las variables tienen tanto propiedades como **comportamientos**. Los comportamientos son los disparadores

activos para los datos. Cuando se le da un comportamiento a una variable se le enseña a esa variable (escribiendo un código) cómo inicializarse, cómo desplegarse, cómo calcular sus propios valores, etc.

Cuando las variables tienen comportamientos y propiedades el resto del trabajo de programación se facilita. Ya no se tiene que escribir el código que maneje las variables cada vez que se les usa.

Pero claro un programa de procedimientos estructurados bien escrito es mucho mejor que un programa O.O.P. mal escrito. En la actualidad las compañías usan adecuadamente millones de líneas de código no O.O.P. todos los días, y hay pocas razones para volver a escribir el código que ahora trabaja bien, sin importar el método o lenguaje en que haya sido escrito.

Sin embargo, el mundo actual de la computación plantea unos cuantos retos más que el de ayer. No solamente se está moviendo la gente hacia los ambientes gráficos de usuario, sino que el mundo está llegando a ser una economía global, los negocios deben interactuar con toda clase de datos no antes vistos, la cantidad de datos que debe procesarse es inimaginable y la lista de pendientes de los departamentos de procesos de datos continua creciendo día a día.

La programación orientada a objetos no es una respuesta para todos los problemas de programación actuales, sin embargo, la O.O.P. ayuda a resolver mucho atolladeros de programación ya que es una forma natural de programar.

1.3 Ventajas y desventajas del lenguaje ensamblador sobre lenguajes de alto nivel.

En resumen podemos decir que los lenguajes de programación de computadoras pueden clasificarse en dos grupos: lenguajes de bajo nivel y lenguajes de alto nivel[9]. Los lenguajes de bajo nivel están limitados a lenguaje de máquina y tienen una fuerte correspondencia entre las operaciones implementadas por el lenguaje y las operaciones implementadas por el hardware. Los lenguajes de alto nivel, están limitados a lenguajes usados por los humanos para expresar problemas y algoritmos. Cada sentencia en un lenguaje de alto nivel puede ser equivalente a muchas sentencias en un lenguaje de bajo nivel.

Los lenguajes de programación de alto nivel proveen al programador de un ambiente amigable, así que la tarea de codificación de algoritmos se ve simplificada considerablemente. En un algoritmo, cada código en el lenguaje de programación seleccionado, se traslada normalmente sin la intervención humana en un conjunto de instrucciones de máquina equivalentes. Estas instrucciones se ejecutan mediante el hardware y si todo va bien, se produce el efecto deseado del algoritmo.

La clave de la ventaja que ofrecen los lenguajes de alto nivel es la **abstracción**.

Conforme el nivel de abstracción se incrementa, el programador necesita conocer menos y menos acerca del hardware en el cual se ejecuta el programa. En otras palabras, las ideas de programación pueden separarse del diseño de hardware, por ejemplo, en un nivel de abstracción de un lenguaje de alto nivel, el programador puede referirse a nombres de variables simbólicas en vez de referirse a direcciones numéricas de memoria.

---

[9]Watson Des/High Level Languages

El control de abstracción permite al programador de lenguaje de alto nivel expresar algoritmos en términos de estructuras tales como ciclos while, sentencias if y procedimientos, la composición de un programa en un conjunto de subprogramas es otro ejemplo de la idea de abstracción.

Un lenguaje de programación de alto nivel se traslada a su código de máquina equivalente mediante un programa llamado compilador.

1.4 La necesidad de lenguajes de alto nivel

Una programación en código de máquina tiene las siguientes desventajas:

- Existe una gran probabilidad de error en todas las etapas del proceso de programación.
- Cualquier programa que pueda expresarse como un simple algoritmo resulta en largos programas, puesto que cada operación de máquina sólo puede ejecutar una operación más simple que dicho código de máquina. Esto hace que los programas se vuelvan difíciles de mantener y validar, esto resulta en una gran carga para el programador.
- Los cálculos de dirección de memoria también tienen que realizarse a mano, lo cual implica un trabajo pesado y la probabilidad de errores aumenta. Además hay que tener cuidado en mantener las localidades de almacenamiento precisas de todas las instrucciones y datos, así que si el programa requiere alguna modificación, los argumentos de direcciones

deben alterarse.

Algunas desventajas del código de máquina pueden superarse mediante una computadora responsable de la traslación. El programa podría escribirse en términos de operaciones máquina básicas, pero la traslación a notación binaria se ejecuta mediante la computadora. El programa que realiza la traslación se denomina ensamblador.

1.5 Lenguajes de alto nivel contra lenguajes de bajo nivel

Los lenguajes de alto nivel ofrecen mejores ventajas sobre los lenguajes de bajo nivel.

La primera motivación para usarlos es que muchos problemas pueden resolverse mucho más rápida y fácilmente.

Por ejemplo, un lenguaje de alto nivel diseñado para cálculos numéricos tienen la habilidad de expresar expresiones aritméticas en la forma convencionalmente usada tal como la notación infija. Muchas otras características de los lenguajes de alto nivel son importantes al respecto, tal como la disponibilidad de datos, facilidades de estructuración de programas, de funciones potentes, relevantes, operadores y la ventaja de programas estructurados en módulos diferentes.

Una ventaja relacionada con los lenguajes de alto nivel es que generalmente son mucho más fáciles de aprender y entender. El programador de un lenguaje de alto nivel no necesita conocer en detalle la estructura del hardware. El ambiente de

programación provisto por el lenguaje de alto nivel debe ser mucho más confortable que el que ofrece la máquina o el ensamblador. En otras palabras el lenguaje de alto nivel ofrece abstracción.

Los programas en lenguaje de alto nivel son mucho más fáciles de seguir que los escritos en un lenguaje ensamblador por las siguientes razones:

- Son más probables de **documentarse** por sí mismos.
- La estructura del programa puede hacerse para reflejar la estructura del problema original.
- El significado exacto de los nombres puede determinarse por variables y subprogramas.
- La solución del problema no necesita oscurecerse como sucede en los lenguajes de programación de bajo nivel por el nivel de detalle que necesitan.

No obstante es posible que algún programador escriba programas oscuros en un lenguaje de alto nivel.

La **portabilidad** es otra motivación para escribir en lenguajes de alto nivel. Si un programa se escribe en lenguaje de alto nivel estándar, existe un alto grado de independencia con la máquina. Esto es un contraste con respecto a los lenguajes de bajo nivel, los cuales son muy enfocados a las características de la máquina y por lo tanto no generan programas portables. La portabilidad se obtiene como resultado de la abstracción del lenguaje de alto nivel. Este es un aspecto importante de la

programación.

Los programas en lenguaje de alto nivel son mucho más fáciles de mantener por las siguientes razones:

- Dado que un programa en lenguaje de alto nivel generalmente es más fácil de seguir y entender que su equivalente en lenguaje de bajo nivel, el programador puede **localizar sus errores** más fácilmente mediante una simple inspección visual.
- El compilador puede proveer una facilidad de **depuración**. Si el compilador ofrece un conjunto flexible de operaciones de verificación en tiempo de compilación, la probabilidad de errores en el código compilado puede reducirse significativamente. Operaciones tales como verificación y prueba de tipo separan la compatibilidad de parámetros entre la definición de un subprograma y su llamada.
- Dependiendo de la naturaleza del lenguaje, el compilador opcionalmente puede incluir instrucciones en el código generado para la detección de errores de run-time (durante la ejecución), tales como sobreflujo numérico (overflow) y violación de los límites de los arreglos. Tales verificaciones pueden reducir dramáticamente el tiempo de depuración.
- Algunos lenguajes inherentemente soportan mantenimiento interactivo. Por ejemplo, en BASIC es muy fácil detener la corrida de un programa en cualquier punto e imprimir los valores de las variables claves. Alternativamente, algunas implementaciones de lenguaje incorporan un

paquete de mantenimiento que permite que el programador examine el funcionamiento de la corrida del programa, usando nombres y estructuras definidas en el alto nivel original del programa fuente. Idealmente el compilador podría permitir la generación de un código run-time de depuración que se activaría o desactivaría bajo el control del programador. Por ejemplo, en circunstancias donde el programador puede palpar que una sección de código o un programa completo es más improbable que falle y su eficiencia en cuanto al tiempo de ejecución es importante el programador podría instruir al compilador para no producir código de depuración, esto reduciría la sobrecarga en el tiempo de ejecución.

Por lo tanto, los lenguajes de alto nivel pueden ser aprendidos, usados y mantenidos por lo que los programas en lenguajes de alto nivel son más **fáciles de modificar** que los programas en lenguaje de bajo nivel. Esto es una ventaja importante. Una ventaja mayor es el potencial para la modificación en cuanto a la habilidad para localizar los efectos de una modificación, sin embargo, algunos lenguajes especialmente los de bajo nivel, son pobres al respecto. Por ejemplo, si todas las variables son globales, una modificación que altere los valores de dichas variables puede tener repercusiones en el resto del programa. Otros lenguajes proveen facilidades de información ya que la extensión de las variables puede limitarse estrictamente y contener los efectos de una modificación.

Algunas ventajas para continuar con el uso de los lenguajes de bajo nivel son:

- Puede ser necesario ejecutar algunas funciones de hardware de bajo nivel que no soporte directamente el lenguaje de alto nivel.
  Una técnica convencional permite que tales funciones se transporten fuera del lenguaje de alto nivel, esto provee una librería de control de subrutina escrita para un lenguaje de bajo nivel, llamada por un programa en lenguaje de alto nivel.
  En implementaciones de los lenguajes de programación los aspectos específicos de la máquina, de las funciones en lenguajes de alto nivel tales como entrada y salida, y localidades de almacenamiento, son provistas por una librería de subrutina en lenguaje de bajo nivel. Para algunas aplicaciones el pobre control ofrecido por muchos lenguajes de alto nivel puede ser una mayor dificultad, algunos lenguajes de alto nivel ofrecen facilidades excepcionales de manejo permitiendo el control del programa antes de las fallas, mediante interrupciones o eventos externos asíncronos.
- Ventajas o eficiencias son ofrecidas en los lenguajes de bajo nivel. Para algunas aplicaciones, puede ser vital que el programa pueda ser muy pequeño y/o ejecutado lo más rápido posible. Algunas personas dicen que el único camino para realizar esto es vía lenguaje de bajo nivel. Sin embargo, comparativamente los recientes desarrollos en el diseño de los compiladores -en particular en las técnicas de generación de código -

hacen posible que un compilador genere un código de muy alta calidad, el cual es tan bueno como, sino es que mejor que, uno escrito a mano.

Lenguaje de alto nivel	**Lenguaje de bajo nivel**
- Fácil de aprender y entender. - Los programas pueden documentarse a sí mismos. - Rápida solución de problemas. - Portabilidad. - Simplificación de la depuración. - Simplificación de la modificación y mantenimiento	- Acceso a operaciones máquina de bajo nivel. - Eficiencia de tiempo/espacio (no siempre sucede).

1.6 Traductores, características y ventajas

Un traslador recibe una entrada y entonces convierte un programa fuente en su programa objeto o programa destino. El programa fuente está escrito en un lenguaje fuente y el programa objeto se traslada a un lenguaje objeto.

Si el lenguaje fuente ha trasladar es lenguaje ensamblador y el lenguajes objeto

es lenguaje de máquina al traslador se le llama **ensamblador**[10].

Debido a que un programa en ensamblador consta de una serie de sentencias en línea, parece natural tener un ensamblador que lea una sentencia, la traduzca a lenguaje de máquina y escriba el código de máquina generado en un archivo y la porción del listado correspondiente, si la hay, en otro. El proceso se repetiría hasta que todo el programa se haya traducido. Por desgracia este método no funciona.

Considérese la situación en la que la primera sentencia sea un salto a E. El ensamblador no puede ensamblar esta instrucción hasta que conozca la dirección de la sentencia E. La sentencia E puede estar cerca del fin del programa, lo que impide que el ensamblador encuentre la dirección sin leerse primero casi todo el programa. Esta dificultad se llama **referencia adelantada,** porque el símbolo E se ha usado antes de que se haya definido; es decir, se ha hecho referencia a un símbolo cuya definición aparecerá más tarde.

Las referencias adelantadas pueden tratarse de dos maneras. La primera es que el ensamblador podría leer el programa fuente dos veces. Cada lectura del programa fuente se llama una pasada, y todo traductor que lea el programa de entrada dos veces se llama un **traductor de dos pasadas**. En la pasada 1, el ensamblador de dos pasadas colecciona todas las definiciones de símbolos, incluyendo las etiquetas de las sentencias, y las almacena en una tabla. En el momento en que empieza la segunda pasada, ya se conocen los valores de todos los símbolos; ya no existe el problema de las referencias adelantadas y puede leerse cada sentencia, ensamblarla y obtener una

[10]Tremblay/The Theory and Practice of compiler Writing

salida. Este método es conceptualmente sencillo, aunque requiere una pasada adicional sobre la entrada.

El segundo método consiste en intentar hacer el ensamblaje en una pasada, a pesar de todo. Cuando se encuentra una sentencia que no puede ser ensamblada por contener una referencia adelantada, no se genera ninguna salida; en vez de hacerlo , se introduce la sentencia con la referencia adelantada en una tabla, con la indicación de que todavía no ha sido ensamblada. Al final del ensamblaje, todos los símbolos habrán sido definidos, de modo que todas las sentencias de dicha tabla pueden ensamblarse.

Este último método genera una salida que no está en el mismo orden que en el de las dos pasadas. Si al ensamblaje le sigue una carga, el cargador puede reordenar las piezas de salida para dejarlas en el orden correcto. Por tanto, esta objeción no tiene importancia. El problema del ensamblador de una pasada estriba en que si hay muchas sentencias que contengan referencias adelantadas, la tabla que debe contener todas las sentencias no ensambladas puede hacerse demasiado grande y no caber en memoria. Sin embargo el ensamblador podría escribir la tabla en memoria secundaria y leerla de nuevo más tarde.

Los ensambladores de una pasada tienden a ser más complicados que los de dos. Por ejemplo, los programadores en ensamblador a menudo esperan un listado que contenga las sentencias fuente listadas en el orden en que aparecen en el programa, junto con su traducción a lenguaje de máquina. El ensamblador de una pasada no puede imprimir la traducción a lenguaje de máquina hasta que la haya hecho. Por tanto, debe efectuar una segunda pasada para imprimir el listado o bien tener otra tabla interna donde se guarden las piezas de listado hasta que puedan imprimirse. La

mayoría de los ensambladores realizan dos pasadas.

Los ensambladores modernos pueden reconocer direcciones simbólicas y mnemónicos que representan las operaciones de máquina.

Entonces el ensamblador realiza la traslación a la cadena equivalente de 1's y 0's. El ensamblador también tiene suficiente capacidad para los problemas del cálculo de direcciones, usando nombres tipo texto para las direcciones así como para otros datos. Por lo tanto, el programador no necesita estar preocupado por las alteraciones de los argumentos de dirección hechas durante la adición o remoción de instrucciones o datos durante el desarrollo del programa, ya que esto lo realiza automáticamente el ensamblador. La consecuencia de esta traslación automática es que los programas en lenguaje ensamblador son mucho más fáciles de escribir y mantener que los programas en código de máquina.

Los términos **compilador e intérprete** se refieren a la forma en que se ejecuta un programa. En teoría cualquier lenguaje de programación puede ser compilado o interpretado, pero algunos suelen ejecutar en una forma u otra.

Los lenguajes que usamos para crear aplicaciones son muy diferentes al lenguaje de máquina. Los 4GL están mucho más lejos de los lenguajes máquina que los lenguajes de tercera generación. Es necesario trasladar desde el lenguaje de aplicación a la creación de un código de máquina ejecutable.

El código escrito en 3GL se denomina código fuente. Este se traslada con un compilador a un código objeto el cual se ejecuta en una computadora destino asistida por un sistema operativo.

Un **intérprete** lee el código fuente de un programa línea a línea, realizando las

instrucciones específicas contenidas en esa línea. Un intérprete opera en línea (**on-line**). Este traslada unidad por unidad, trasladando y ejecutando el significado del código fuente. Un intérprete **produce resultados**. Este ejecuta el código fuente a la vez que lo traslada, es decir, trabaja en línea.

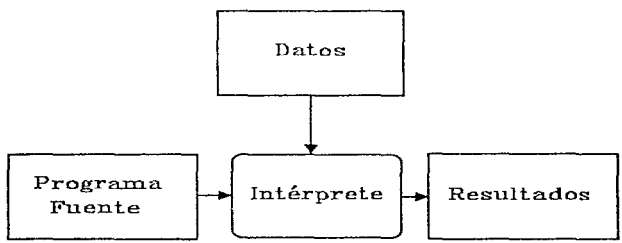


Figura 1.6.1 Proceso de un Intérprete

Un **compilador** lee el programa entero y lo convierte a **código objeto**, que es una traducción del código fuente del programa a una forma que puede ser ejecutada directamente por la computadora. El código objeto también se suele denominar código binario o código de máquina. Una vez que el programa está compilado las líneas de código fuente dejan de tener sentido en la ejecución del programa.

Un compilador opera fuera de línea (**off-line**). Este rastrea el programa completo, usando tiempos, creando tablas de variables para habilitarlas al hacer la

traslación a código objeto.

Cuando se usa un intérprete, el código fuente debe estar presente cada vez que se quiere ejecutar el programa.

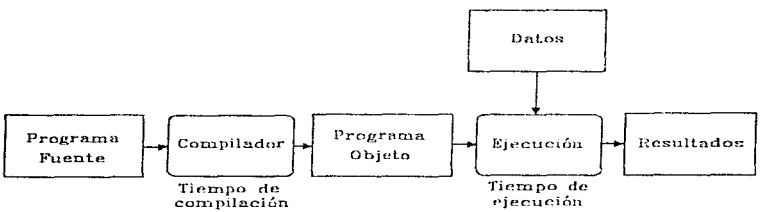


Figura 1.6.2 Proceso de compilación

Generalmente un intérprete es mucho menos eficiente que un compilador. Partes de la traslación se hacen repetitivamente. Cada tiempo el intérprete encuentra una sentencia y debe trasladarla, igual si la encuentra varias veces.

Esto contrasta con un compilador el cual, rastrea el programa completo varias veces elaborando sus tablas. Un compilador puede diseñarse para optimizar el código que produce. Un intérprete no puede optimizar el código de la misma forma y tiene que retrasladar cada vez que el programa se re-ejecute.

Más 4GL están diseñados para operación en línea, y tiene lugar un diálogo entre el lenguaje de software y la persona que realiza la aplicación. Antes de cualquier

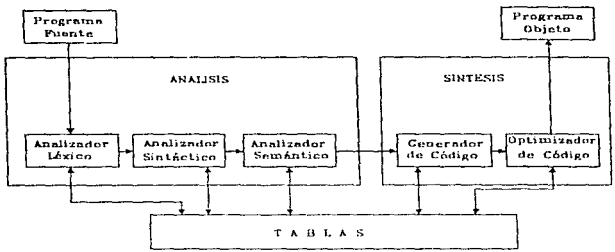


Figura 1.6.3 Componentes de un compilador

prueba en la interpretación o compilación, el diálogo ayuda al usuario a hacer su programa y verificar errores. La verificación inicial usualmente la hace el intérprete.

Muchos 4GL usan intérpretes y por ello adquieren la reputación de ser inapropiados para cálculos pesados cuando se trabaja con grandes volúmenes de información. Algunos 4GL usan un compilador de optimización y están diseñados para cálculos pesados. En el futuro esto puede sensarse para ciertas clases de 4GL para tener ambos, un intérprete y un compilador. El compilador ayuda a realizar el cómputo eficientemente; el intérprete ayuda al diseñador a hacer una prueba y verificación en línea. El diseñador puede hacer su aplicación usando un intérprete, probando prototipos de salida con el usuario final y entonces, cuando el usuario final esté satisfecho, alimentar el código fuente dentro de un compilador de optimización.

El intérprete puede habilitar el uso para ver resultados tan rápido como sea posible, el compilador puede producir código para la máquina tan eficiente como sea posible.

El compilador puede compilar el código fuente de tal forma que puede ejecutarse en diferentes máquinas. El problema de portabilidad puede superarse, en parte, recompilando los programas ejecutándolos en diferentes máquinas o en máquinas incompatibles.

1.7 Historia de las microcomputadoras[11]

1.7.1. De la microelectrónica a la microcomputadora

La llamada "electrónica" comenzó con la aparición de la válvula electrónica alrededor de 1910, los llamados entonces "circuitos de baja corriente" se originaron con la conexión de componentes pasivos (resistencias, condensadores y bobinas) con la de componentes activos (válvula electrónica), soldados con conductores de diversos colores. En 1948 la válvula electrónica comenzó su despedida con la aparición del transistor bipolar (tabla 1.7.1).

Tabla 1.7.1 De la electrónica a la microelectrónica

Componente	Circuito	
1910 Válvula electrónica	Cables	Electrónica
1948 Transistor (bipolar)		

[11]Keil/Microcomputadores

Componente	Circuito	
1950	Circuito impreso	
1962 Circuito lógico	Circuito integrado (SSI, MSI)	Microelectrónica
1963 Transistor MOS		
1970 Memoria MOS	Integración en gran escala (LSI) (1000 componentes/chip)	
1973 Microprocesador Microcomputador		
1977	Integración en escala muy grande (VLSI) (50000 componentes/chip)	

En los 50's aparecieron los "circuitos impresos", actualmente se les denomina "placas". En general se trata de placas de materiales sintéticos (fibra de vidrio, baquelita, entre otros) de tamaño de una postal, en la que las pistas conductoras reemplazan el anterior cableado. Las terminales de los componentes electrónicos se pasan a través de perforaciones en la placa y se soldan a las pistas. Con ello se logró:

- ✓ mejorar la fiabilidad del circuito,
- ✓ reducir los costos de fabricación,
- ✓ aumentar la capacidad y facilitar su mantenimiento.

Se habla de microelectrónica por la década de los años 60, cuando se logró

producir a un costo ventajoso varios componentes y sus conexiones sobre material de cristal de silicio de unos pocos milímetros cuadrados de superficie a los que se le denominó "chip".

1.7.2 Tipos de densidad de integración y sus costos

En 1962 surgieron los primeros circuitos integrados que contenían transistores, diodos, resistencias, entre otros componentes y funcionaban como compuertas lógicas. La tecnología digital había comenzado. El transistor, al igual que en su tiempo la válvula electrónica, se utiliza para amplificar débiles señales eléctricas, una de las principales funciones del transistor es como interruptor, conduce o no corriente eléctrica. A estos estados se le denominaron respectivamente "1 lógico" y "0 lógico".

En 1963, el transistor **MOS** (Metal-Oxide-Semiconductor) siendo el metal el conductor, el óxido el aislante y el silicio el semiconductor. En comparación con el transistor bipolar, el MOS sólo tiene un tipo de portadores de carga, el **PMOS** los que según la tecnología serán sólo positivos o **NMOS** sólo negativos y **CMOS** (C=complementario) si se integran ambas tecnologías MOS en un chip.

La gran ventaja de los transistores MOS con respecto a los bipolares es su reducción de pérdidas de potencia, permitiendo aumentar más aún la densidad de integración. En 1968 se produjeron circuitos con más de 100 de estos transistores.

Actualmente la densidad de integración se clasifica en:

- ***Small Scalle Integration*** **(SSI,** integración de pequeña escala) y
- ***Medium Scalle Integration*** **(MSI,** integración de media escala), que remarcaron los años sesenta.

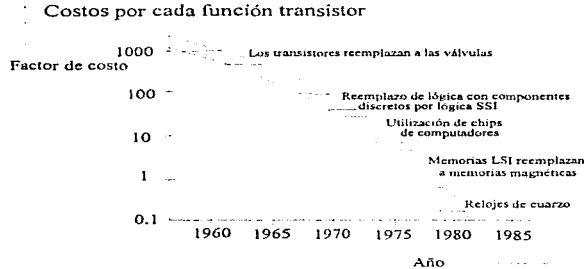


Figura 1.7.1 Costos por cada función transistor

- *Large Scale Int*egration (LSI, integración de gran escala) los trajeron los años setentas, y se considera como tales a los circuitos integrados con más de 1000 componentes y
- *Very Large Scale Integration* (VLSI, integración de escala muy grande) a los de más de 50000 componentes.

Este incremento de la densidad de integración, gracias a procesos de producción altamente automatizados, permitió llegar a una reducción de costos en un factor de 1000 por cada transistor (figura 1.7.1).

1.7.3 Ventajas de la microelectrónica

Los circuitos integrados son:

- ✓ más pequeños (es posible realizar circuitos de alta complejidad, antes casi inimaginables)
- ✓ más fiables (un sólo componente en lugar de miles)
- ✓ de menor consumo (que gran número de componentes) y
- ✓ de menor costo (procesos de fabricación altamente automatizados).

1.7.4 Repercusión de la microelectrónica

Durante la primera década de los circuitos integrados (**CI**) se desarrollaba un circuito específico para cada aplicación. La baja escala de integración que en sus comienzos tenían los circuitos integrados hizo que en el mercado se tuvieran únicamente circuitos lógicos simples **AND, NAND, OR, XOR**, que servían a todos los diseñadores de circuitos para técnicas digitales. Como consecuencia de la microelectrónica, los circuitos lógicos se convirtieron en un producto masivo.

Circuitos para usos específicos de los clientes

En los circuitos para usos específicos de un cliente éste debe asumir la totalidad de los costos de fabricación debido a que son circuitos integrados que se fabrican especialmente para él. Los costos comprenden, entre otros, los de diseño y las máscaras. Y solamente es rentable únicamente si puede comprar una cantidad muy grande de circuitos integrados (en el orden de las 10000 unidades) para poder disminuir los costos totales.

Microcomputadoras

El microprocesador brindó una segunda posibilidad. Su fabricación requiere alta tecnología, es una estructura de alta complejidad, pero la ventaja del producto es su uso masivo. Es posible debido a que el microprocesador no es capaz de desempeñar ninguna función por si solo, esta debe ser "comunicada" previamente, lo cual ocurre gracias a las posibilidades que tiene el diseño de los circuitos para lograrlo.

Esto significa:

✓ **Cada diseñador tiene la posibilidad de asignar al microprocesador una función específica.**

Esta función no está incluida en el propio circuito integrado, lo está en un

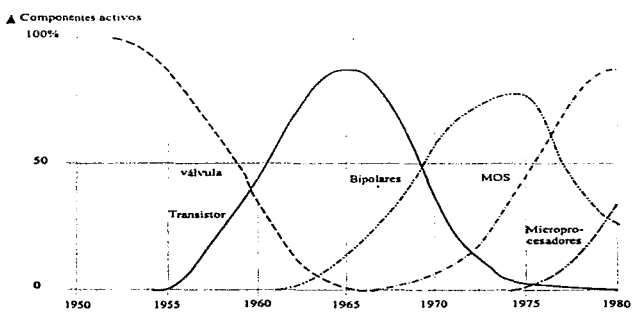


Figura 1.7.2 Evolución tecnológica de los componentes semiconductores

"programa", es decir en una secuencia de instrucciones para el microprocesador, que están almacenadas en uno o más módulos de memoria.

Además que al microprocesador se le pueden anexar otros módulos (periféricos) en su entorno, los módulos de "entrada-salida" (E/S). Con ello el microcomputador permite la penetración de la microelectrónica en equipos que no se fabrican en cantidades tan grandes como para que tenga sentido desarrollar un circuito integrado específico para un cliente (figura. 1.7.2).

1.7.5 Aparición de las computadoras programables

Un microprocesador representa un único chip con una unidad central completa como la de las computadoras grandes. La diferencia radica en la extensión de sus trabajos (figura 1.7.3).

El alemán Conrad Zuse construyó a base de relés, la primer computadora con programación libre, le siguió EE.UU en 1946 con la primera computadora con válvulas electrónicas denominada ENIAC.

1.7.6 Microcomputadoras

La empresa norteamericana Intel al inicio de la década de los setenta, desarrolló un circuito integrado según los deseos de un cliente y además de contener una lógica de control para conductores de entrada/salida, tuviera memoria y realizara operaciones aritméticas. Surgió el chip de calculadora 4004 que reemplazaba a 6 circuitos integrados específicos. El 4 en la denominación indicaba que se procesaban simultáneamente 4 bits. La denominación "bit" designa a la unidad más pequeña de

información.

Otro pedido de un cliente para controlar una terminal de datos, dio lugar en 1972 al 8008. Este circuito integrado estaba diseñado para procesar señales alfanuméricas mientras que el 4004 estaba especializado en aritmética **BCD** o sea en cifras decimales codificadas en binario.

En el transcurso de los dos siguientes años, nació el muy difundido

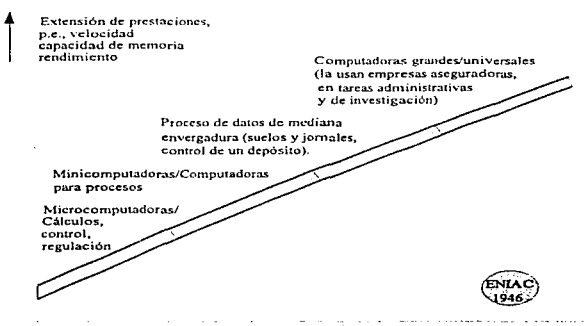


Figura 1.7.3 Diagrama precios-prestaciones de computadoras

microprocesador 8080. En 1976 llegaba su sucesor, el 8085 y en 1978, la primera micromputadora de 16 bits, el 8086. La capacidad creció 100 veces en tanto que el

precio cayó de $300 a $3 dólares americanos (figura 1.7.4, figura 1.7.5).

Ventajas y desventajas de cada tecnología

✓ Circuitos integrados con transistores bipolares (técnica **TTL**, *Transistor-Transistor-Logic*, lógica transistor-transistor) posibilitan por ejemplo, mayores velocidades de conmutación que

✗ en aquellos transistores de efecto de campo (técnica MOS).

✓ Los transistores de efecto de campo permiten una mayor densidad de componentes debido a que disipan una potencia menor y no requieren un aislamiento adicional entre ellos.

✓ A su vez NMOS es más rápido que PMOS, necesita una única fuente de

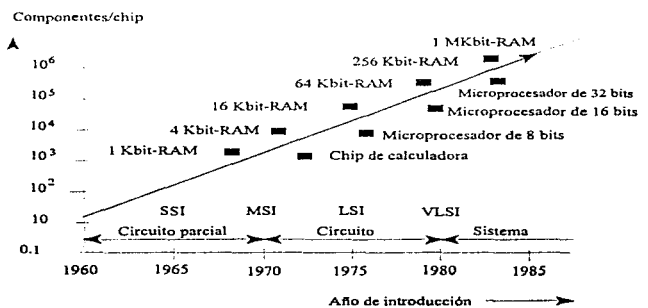


Figura 1.7.4 Evolución de la densidad de integración

alimentación (+5V) y permite una conexión directa con circuitos TTL (compatibilidad con TTL).

✓ Los circuitos CMOS tienen una corriente de reposo extremadamente reducida, del orden del nanoamperio. por ello se adaptan especialmente a equipos alimentados con baterías. Además se les puede alimentar con un amplio margen de tensiones (3 a 15V) y se eliminan costosos circuitos estabilizadores de tensión, gracias a sus favorables características de transferencia se obtiene un nivel de inmunidad a tensiones

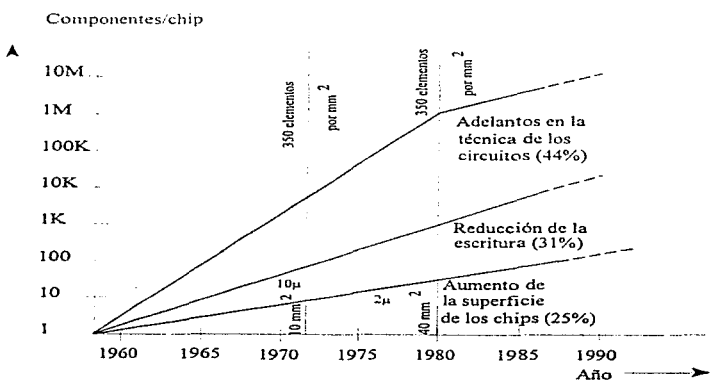


Figura 1.7.5 Influencias sobre la densidad de integración

perturbadoras relativamente grande (1.5 V contra 0.4V del TTL). Por estas cualidades los módulos CMOS se utilizan preferentemente en la electrónica para automóviles.

✗ Sus desventajas son la gran superficie de sus componentes y su costoso proceso de fabricación.

✓ Las tecnologías se van mejorando constantemente, MYMOS, por ejemplo, ya reduce las estructuras a 3 μm.

✗ Para obtener estas dimensiones se requieren equipos de producción muy costosos.

✓ Las instalaciones de implantación de iones, por ejemplo, permiten que el proceso de dopado se efectúe con mayor precisión que en los hornos de difusión. Con el traspaso de las estructuras de la máscara al silicio, la litografía de haces electrónicos reemplazará a la litografía óptica para obtener, de esta manera, estructuras más pequeñas que 1μm.

Superficie del chip

La superficie del chip no se puede hacer de cualquier tamaño. Si se exceden las dimensiones actualmente usuales, de 25 a 30 milímetros cuadrados, resulta una reducción exponencial del "rendimiento". La mayoría de las pérdidas de rendimiento se producen por defectos individuales que, entre otros, son defectos de material, agujeros producidos por óxido, defectos fototécnicos, así como impurezas moleculares e iónicas. Un único defecto crítico ya es suficiente para inutilizar funcionalmente un circuito. La misión principal del ingeniero que efectúa el diseño será, pues, la de optimizar las pistas que conectan los componentes, debido a que ocupan la mayor parte

de la superficie.

1.7.7 Usos de las microcomputadoras

En la actualidad la microcomputadora penetra en todos los campos de la tecnología. Por una parte, se trata en muchos casos de reemplazar componentes electromecánicos o puramente mecánicos en sistemas técnicos. Por otra parte, han llegado a ser factibles muchas cosas que antes resultaba imposible realizar o hacerlo únicamente a un costo prácticamente injustificable.

Proceso de datos

En el proceso de datos, usando microcomputadoras, se puede descentralizar las tareas de cálculo. Esto significa que se los preelabora y prepara directamente en el puesto de trabajo, o sea allí donde se originan los datos. Por medio de estas "terminales inteligentes" se efectúa el proceso de datos en el puesto de trabajo, ya sea en la oficina, en la producción o en el mostrador.

Además de los ejemplos mencionados, existen desde las calculadoras de bolsillo hasta las computadoras personales (*Personal-Computers*-PC), sistemas de computadoras independientes con los usos más variados, desde la afición hasta las aplicaciones profesionales en el área comercial.

Telecomunicaciones

Para poder satisfacer los crecientes requerimientos de las comunicaciones se crearon nuevos sistemas para las telecomunicaciones, con ayuda de las

microcomputadoras. Esto llevó a nuevas características de las prestaciones de telefonía y del télex. El televisor, con el videotexto y otros servicios, evoluciona hacia un sistema informático multifacético.

Técnología de mediciones, control y regulación

La técnología de mediciones, control y regulación es el campo de aplicación más amplio para las microcomputadoras. De ella proceden también los impulsos más importantes para la "arquitectura" de los actuales microprocesadores y microcomputadoras.

En la *industria*, sistemas de uno o más procesadores controlan líneas de producción, no sólo en grandes fábricas sino también en forma rentable en las pequeñas, como por ejemplo, en fábricas de pastas alimenticias y embutidos. Además aumentan la productividad y mejoran la calidad (figura 1.10).

Los robots industriales controlados por sensores se hacen cargo de pesadísimos trabajos corporales, por ejemplo, en ambientes con temperaturas elevadas, expuestas a peligro de explosiones o insalubres.

En la *tecnología médica*, la microcomputadora sirve a la salud, alivia el control del paciente sometido a terapia intensiva (presión sanguínea, pulso, temperatura, EEG, ECG), amplía el conjunto de los equipos de diagnóstico y análisis, efectúa la evaluación estadística de los datos y los almacena. En la tecnología de las prótesis posibilita, por ejemplo, nuevas ayudas para caminar y la visión.

En el *control del tráfico*, la microcomputadora cuida nuestra seguridad, esto se extiende desde el control en función del tránsito de los semáforos hasta la seguridad

en vuelo. Computadoras a bordo de los automóviles controlan sistemas antibloqueo de frenos, señalan el peligro de hielo formado sobre el suelo resbaladizo y controlan en forma óptima la combustión (economía de energía, protección del medio ambiente).

En el *hogar* se necesitan aparatos que economicen energía, sean más seguros, brinden mayor rendimiento y confort. En consecuencia, se encuentran en microcomputadoras no sólo en el control de la cocina, en la lavadora y lavavajillas, sino también en el control de la calefacción, así como en los aparatos de la electrónica de entretenimiento y tiempo libre.

1.7.8 Significado para el usuario de las microcomputadoras

Para los usuarios de microcomputadoras existen tres estrategias de mercado:

- ★ *Invención* — Especialmente al pequeño fabricante de aparatos la microcomputadora le brinda la oportunidad de abrir nuevos mercados con productos novedosos y con buenas ideas materializadas con mayor rapidez en forma de los correspondientes productos.
- ★ Mejoramiento — El microcomputador posibilita un incremento muy elevado de las prestaciones de los equipos, con reducción de los costos por cada función.
- ★ *Conservación de equipos* — Es la de conservar sistemas e instalaciones con larga vida útil y elevado valor total

acumulado.

Ventajas y problemas

Las ventajas para el usuario de las microcompuadoras resultan por sí solas (figura 1.7.6).

- ✓ sistemas de módulos y grupos modulares estándar (también se puede obtener rápidamente precios convenientes en pequeñas cantidades),
- ✓ mejoras en las funciones de los equipos ("inteligencia"),
- ✓ programables por el propio usuario. Los conocimientos prácticos del sistema no se divulgan a terceros,
- ✓ no existe dependencia de un único fabricante de componentes (second source-segunda fuente) y
- ✓ se mejora la posición en el mercado en el cual se vuelcan nuevos equipos e fecha más temprana.

Los problemas que muchas veces deben resolverse se pueden esquematizar como sigue:

- ✗ reestructuración en la fábrica (desarrollo, fabricación).
- ✗ cambios en la estructura del personal (cantidad, calificación); capacitación,
- ✗ se requiere de un grado de flexibilidad debido al rápido cambio en la tecnología de las microcomputadoras.

La aplicación de la tecnología de las microcomputadoras requiere del fabricante de equipos:

- ☞ adecuada formación o perfeccionamiento del personal (hardware o software),

- estrecha cooperación con fabricantes de componentes electrónicos,
- dirección cooperativa, flexible y que fomente las innovaciones,
- cuidadosa planificación del producto, de la estrategia de mercado y
- la innovaciones que deben ejecutar a tiempo.

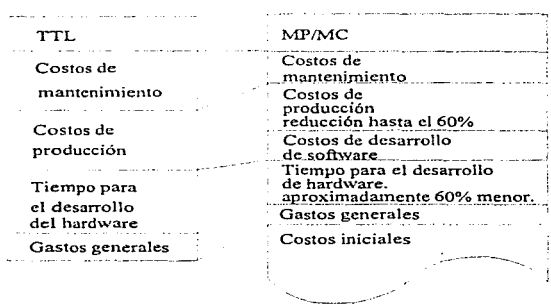


Figura 1.7.6 Variación de los costos en el desarrollo de equipos con microprocesadores y microcomputadoras

1.8 Características principales y ventajas de los MCU's 68705P3, R3 y U3

1.8.1 Características de Hardware de la Familia 6805[12]

1.8.1.1 Introducción[13]

Cada miembro de la Familia M6805 HMOS/M146805 CMOS (excepto para el MC146805E2) contiene en el chip, casi todo el soporte de hardware necesario para un sistema procesador completo. El diagrama de bloques de la figura 1.8.1 muestra una Unidad Central de Proceso (CPU) la cual es idéntica para todos los miembros de la familia, incluyendo el MC146805E2. Hay solo una diferencia principal en varios miembros de la familia que es solamente el tamaño de los registros stack pointer y contador del programa. El tamaño de esos dos registros es determinado por la cantidad de memoria en el dispositivo, ello varia de 11 a 13 bits. Cada miembro de la familia contiene dentro del chip un oscilador, el cual provee el tiempo de procesamiento, además de un reset (restablecedor o reinicializador) y un interruptor lógico. Los periféricos E/S tales como un timer, algunas líneas bidireccionales E/S, RAM, ROM (excepto para el MC146805E2) están incluidas en todos los chips de los miembros de la familia. Los periféricos y la memoria están ubicados en localidades similares para

[12]Traducción y resumen realizado por Graciela Santillán Alcalá del manual "M6805 HMOS/M146805 CMOS Family Users Manual"

[13]Para más información sobre los componentes de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A3, A64, A127.

toda la familia; por lo tanto, una vez que el usuario esta familiarizado con solo un dispositivo de la familia, esta familiarizado con todos. Además nuevos dispositivos pueden ser incorporados en la familia como son; para los bloques periféricos asociados con el CPU. Esos bloques periféricos podrían incluir líneas adicionales de E/S, mas RAM, EPROM, convertidor A/D, phase-lock-loop o un bus externo.

Se puede elegir entre un económico HMOS de bajo-poder, o bien un CMOS.

La familia de MCU/MPU MC6805 HMOS/M146805 CMOS son implementados usando un simple mapa de direcciones, E/S mapeadas en memoria y la arquitectura de Von Newmann. Los dispositivos periféricos de E/S (A/D, timer, PLL, etc), son accesados por el CPU vía el control periférico y/o registros de datos, los cuales están localizados en el mapa de direcciones. Los datos son transferidos a los

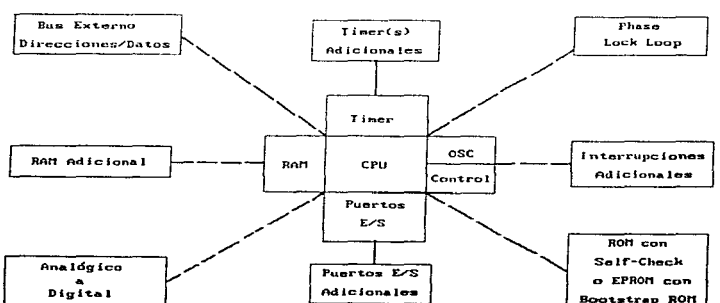


Figura 1.8.1 Diagrama de Bloques de la Familia M6805 HMOS /M146805 CMOS

dispositivos periféricos de E/S con las mismas instrucciones que son usadas para accesar a memoria. La clave para usar la familia MC6805 HMOS/M146805 CMOS está en aprender las características de como los registros periféricos afectan la operación del dispositivo. Ya que como no es usado un segundo mapa de direccionamiento no hay necesidad para el diseñador del sistema de aprender un segundo conjunto de instrucciones especializadas de E/S.

1.8.1.2 Tecnología de Procesamiento

Los diseñadores del sistema tienen la opción de usar entre la tecnología HMOS (M6805) o el CMOS (M146805). Puesto que cada tecnología tiene sus ventajas, hay aplicaciones en las cuales favorecen a una de la otra. La tabla 1.8.1 provee una comparación de las características representativas entre HMOS y CMOS.

Tabla 1.8.1 Comparación de características entre HMOS y CMOS

HMOS	CMOS
Convenientemente económico	Bajo consumo de energía
Consume 10 veces más energía Aumenta la inmunidad al ruido	Amplio rango de voltaje (3-6V) que el CMOS
Operación dinámica Requiere reloj continuo	Más caro ya que la celda del CMOS es más grande
Limitado rango de voltaje	Sensible Latch-up para SCR

HMOS	CMOS
Rápido	Dispositivo Silicon-Gate es tan rápido como el dispositivo HMOS. Completamente operación estática

1.8.1.3 Tipos de almacenamiento[14]

Almacenamiento Temporal (RAM)

La memoria de acceso aleatorio (RAM) es usada como un almacenamiento temporal por el CPU. La RAM es temporal ya que es volátil y su contenido se pierde si la fuente de energía es apagada. Comúnmente la RAM puede ser utilizada para lectura o escritura es mejor usada para almacenar variables. Toda la RAM en el chip está contenida en las primeras 128 localidades de memoria y la parte alta de la RAM es usada actualmente por el procesador como un stack de control del programa. El stack es usado para almacenar direcciones de retorno en llamadas a subrutinas para interrupciones. El registro del stack pointer es usado para mantener la vía de la siguiente dirección libre de una localidad del stack. El stack opera en modo LIFO (último en entrar primero en salir (last in first out)) tal que las operaciones pueden ser anidadas. El tamaño actual del stack varía entre los diferentes miembros de la familia,

[14]Para más información sobre los mapas de memoria de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A12, A73 y A136.

en todos los casos, el sobrepasar los límites del stack puede ser evitado. Si el límite del stack es sobrepasado el apuntador al stack, alrededor del tope del stack ($7F) probablemente el stack de datos se perderá. Cada interrupción requiere cinco bytes del espacio del stack y cada subrutina requiere dos bytes. Si en el peor de los casos un programa requiere cinco niveles de subrutinas anidadas y un nivel de interrupción, entonces 15 bytes del espacio del stack serían reservados. Algún stack RAM no reservado puede ser usado para otro propósito.

La opción low-power standby RAM para HMOS está disponible en el MC6805P4. Aunque el procesador es dinámico la RAM es estática y puede ser alimentada por una fuente separada de energía. La cantidad de RAM implementada es una opción enmascarada y está determinada por la aplicación en particular.

Almacenamiento Permanente (ROM o EPROM)

Todos los dispositivos de la familia M6805 HMOS/M146805 CMOS, excepto el MC146805E2 contienen alguna forma de memoria permanente no volátil. Esto puede ser para cualquiera de los dos programas enmascarados ROM o EPROM borrable con luz U.V.; el M6805 HMOS EPROM contiene versiones EPROM como almacenamiento principal y una pequeña ROM enmascarada la cual es usada para almacenar tablas y constantes. Las versiones de ROM enmascarada son las más económicas para cantidades limitadas usadas para producción de prototipos. Actualmente existen 3 versiones de EPROM. Cada una tiene un poco más almacenamiento y más versatilidad que las actuales versiones de ROM enmascarada, comúnmente, las versiones de EPROM pueden emular las funciones de más de una de

las versiones actuales de ROM enmascarada y podrían ser usadas para futuras versiones de ROM enmascarada.

1.8.1.4 Oscilador[15]

El chip contiene un oscilador en todos los dispositivos de la familia MC6805 HMOS/M146805 CMOS éste esencialmente genera el cronometraje usado por el dispositivo. El oscilador puede ser usado en un número diferente de modelos.

Excepto para la EPROM miembro de la familia M6805 HMOS, una opción de fabricación enmascarada es requerida para seleccionar cualquiera de los dos circuitos: el del oscilador a cristal o el oscilador por resistencia. La frecuencia del oscilador es internamente dividida en cuatro para producir el reloj interno del sistema. El dispositivo EPROM de la familia M6805 HMOS utiliza la opción de registro enmascarado MOR para seleccionar el circuito oscilador a cristal o a resistencia.

La familia de dispositivos M146805 CMOS también utiliza la opción enmascarada de fabricación para seleccionar entre los circuitos de cristal o a resistencia. Comúnmente, una segunda opción enmascarada de fabricación provee cualquiera de las divisiones entre 2 o dividir entre 4, el circuito para producir el sistema interno de reloj. El dispositivo EPROM de la familia M146805 CMOS también utiliza la opción de registro enmascarado (MOR) para seleccionar el circuito a cristal o a resistencia.

---

[15]**Para más información sobre conexiones del oscilador de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A7, A68 y A131.**

1.8.1.5 Resets (Inicializadores)[16]

La familia de procesadores M6805 HMOS/M146805 CMOS pueden ser inicializados en dos formas. Ya sea por el encendido inicial o por el pin reset externo de entrada ($\overline{\text{RESET}}$). Además un voltaje bajo **(LVI)** inhibe al circuito esto incluye a algunas versiones de ROM enmascaradas HMOS a forzar un reset si V_{CC} cae a V_{LVI}. Algunos de los métodos de reset permiten un comienzo ordenado, además la entrada de ($\overline{\text{RESET}}$) puede ser usada para salir en los modos STOP y WAIT CMOS del programa de ejecución. Ambas entradas LVI y el del externo $\overline{\text{RESET}}$ permiten a el procesador recuperarse por otro lado de errores catastróficos. El reset externo ($\overline{\text{RESET}}$) es implementado como una entrada trigger Schmitt para mejorar la inmunidad al ruido. La figura 1.8.2 ilustra el tiempo requerido y los niveles lógicos para dispositivos implementados con LVI. Todos los miembros de la familia M6805 HMOS tienen el equivalente de una resistencia de carga tal que el pin de $\overline{\text{RESET}}$ reflejara un voltaje bajo **Vcc.**

El circuito de reset de encendido HMOS incluye el equivalente de una resistencia de carga interna, tal que solamente un capacitor es requerido externamente. El reset de encendido ocurre cuando una transición positiva es detectada en **Vcc.** El reset de encendido es usado estrictamente para encender las condiciones de energía y no podrá ser usada para detectar alguna baja en la fuente de voltaje. No hay provisión para un reset power-down. Para los dispositivos CMOS, el circuito de encendido para

[16]**Para más información sobre Resets de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A17, A78 y A142.**

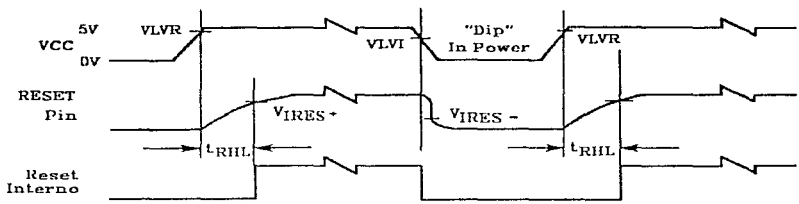


Figura 1.8.2 Tiempos de Encendido y Reset

un retardo 1920 tcyc desde el tiempo de la primer operación del oscilador. Si el pin externo de $\overline{\text{RESET}}$ está en bajo al fin de los 1920 t_{CYC} del tiempo de salida, el procesador permanece en la condición de reset hasta que el pin de $\overline{\text{RESET}}$ va a alto. El reset realiza lo siguiente:

1. Todas las interrupciones son limpiadas con "0"
2. Todas las interrupciones enmascaradas se ponen en "1"
3. Todos los registros de direcciones de datos son limpiados en "0" (entrada)
4. El apuntador al stack es reset en **S7F** (parte alta del stack)
5. Las instrucciones de **STOP** y **WAIT (M146805 CMOS solamente)** son inicializados.

6. El vector de reset es ejecutado y colocado en el contador del programa (el vector de reset contiene la dirección de la rutina de reset)

1.8.1.6 Interrupciones[17]

Generalidades

La ejecución de un programa de la familia M6805 HMOS/M146805 CMOS puede ser interrumpida de las siguientes maneras:

1. La vía externa, el pin $\overline{\text{IRQ}}$ (CMOS) o $\overline{\text{INT}}$ (HMOS). Además, algunos miembros HMOS M6805 incluyen un segundo pin de interrupción externo $\overline{\text{INT2}}$. Las interrupciones externas están enmascaradas.
2. Internamente en el chip con el timer. La interrupción del timer es enmascarada.
3. Internamente para ejecutar la interrupción del software (**SWI**). El **SWI** no es enmascarable.

Cuando una interrupción externa o por timer ocurre, la interrupción no es servida inmediatamente, después de que la actual instrucción es ejecutada y terminada, la interrupción es considerada pendiente. Después que la ejecución de la actual instrucción es completada, la interrupción enmascarada puede ser atendida. Si ambas interrupciones la externa y la del timer están pendientes, la interrupción externa es atendida primero, comúnmente, la petición de interrupción del timer permanece pendiente a menos que sea liberada durante el servicio a la rutina por la interrupción

[17]Para más información sobre interrupciones de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A19, A80 y A143.

externa. La interrupción de software es muy frecuentemente ejecutada de la misma manera que alguna otra instrucción. El pin de interrupción ($\overline{IRQ}$ o $\overline{INT}$) pueden ser verificados con las instrucciones **BIL** o **BIH** (excepto $\overline{INT2}$) para ser usadas como un pin de entrada adicional a pesar del estado de la interrupción enmascarada en el registro de código de condición.

Interrupción del Timer

Si el bit del timer enmascarado (**TCR6**) es limpiado, entonces cada vez que el timer se decrementa a cero (transición de $01 a $00) una petición de interrupción es generada. La interrupción actual de proceso es generada solamente si la interrupción enmascarada el bit del registro del código de condición (**CCR**) también es limpiado. Cuando la interrupción es reconocida, el estado actual de la máquina es introducido dentro del stack y el bit en el **CCR** es colocado y la interrupción actual es atendida. El contenido del vector de interrupción del timer contiene la localidad de la rutina de servicio de la interrupción del timer, esta es entonces cargada en el contador del programa.

Si el modo **WAIT CMOS** es habilitado (para la familia M146805 CMOS) el timer puede ser usado para salir del modo de baja energía y el vector del timer **WAIT** es usado en lugar del vector de interrupción normal del timer. El software puede ser usado para limpiar el bit de la interrupción del timer (**TCR7**). Al final de la rutina de servicio de interrupción del timer, el software normalmente ejecuta una instrucción **RTI** que restaura el estado del CPU anterior a ésta y comienza la ejecución del programa donde se quedó. Notar que si una interrupción externa del hardware es usada

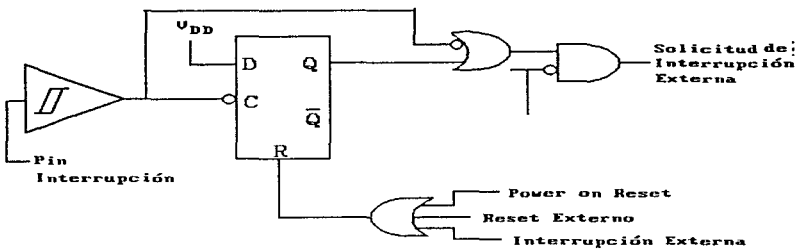
para salir del modo WAIT, el vector del timer interrumpiría a el vector normal del timer en lugar del vector WAIT del timer.

Interrupción Externa

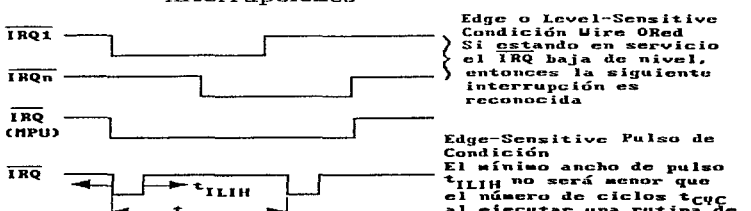
Todas las interrupciones externas son enmascaradas. Si el bit de la interrupción enmascarada (**bit I**) del CCR es colocado, todas las interrupciones son deshabilitadas. Limpiando el bit I se habilita la interrupción externa. Además, $\overline{INT2}$ requiere que el **bit 6** de los diversos registros también sean limpiados. La interrupción externa reconoce ambos niveles es decir es sensible al flanco; las interrupciones para la familia M146805 CMOS son como muestra la figura 1.8.3. La familia HMOS requiere de un flanco negativo para realizar una interrupción; este es generalmente usado en una OR-alambrada como una fuente de interrupción múltiple como muestra la figura 1.8.3b. Las interrupciones sensibles al flanco pueden ser usadas para generar interrupciones periódicas, comúnmente, después que la interrupción es ejecutada por el procesador, la fuente de la petición de interrupción puede retornar a otras tareas. Las peticiones periódicas de interrupción requieren que las líneas de llamadas a interrupción se mantengan bajas por un t_{CYC} y que no sea repetido hasta el fin de la rutina de servicio y las operaciones del stack sean completadas. Esto asegura que todas las peticiones sean reconocidas. La línea de interrupción también debe ser liberada alta para permitir que el procesador siga su trabajo normalmente.

Sobre servicios de peticiones de interrupción pendientes el procesador ejecuta las siguientes secuencias:

1. Enmascara todas las interrupciones (fija el bit I)



a) Diagrama del funcionamiento de Interrupciones



b) Diagrama de modos de Interrupción

Figura 1.8.3 Interrupciones Externas

2. Introduce al stack todos los registros del CPU
3. Carga el contador del programa con un vector apropiado que contiene la localización ($\overline{INT2}$ usa el mismo vector de localización como lo hace el timer).
4. Ejecuta la rutina de servicio.

Interrupción por Software (SWI)

La interrupción por software es ejecutada igual que cualquier otra instrucción y como tal, tomará precedencia sobre las interrupciones por hardware solamente si el **bit I** es colocado (interrupción enmascarada). La instrucción **SWI** es ejecutada en forma similar a la interrupción por hardware si el **bit I** es fijado, los registros del CPU son introducidos al stack, etc. El **SWI** es ejecutado sin reparar en el estado de la interrupción enmascarada en el **CCR**; comúnmente, cuando el **bit I** es limpiado y una interrupción externa o interna de hardware está pendiente, la instrucción **SWI** (o alguna otra instrucción) no serán ejecutados hasta después que la interrupción por hardware haya sido atendida. El **SWI** usa esto bajo un único vector de localización.

1.8.1.7 Puertos E/S[18]

Por lo menos 16 líneas de E/S individuales programables y bidireccionales son incluidas en cada miembro de la familia M6805 HMOS/M146805 CMOS; comúnmente más de estas existen en miembros de la familia. Cada línea es programada

[18]Para más información sobre puertos de los MCU's 68705P3, R3 y U3, consultar el apéndice A en la páginas A9, A69, A133.

individualmente como una entrada o salida, esto es posible a través de los registros de direcciones de datos (DDR) como muestra la figura 1.8.4. La tabla 1.8.2 provee la operación de los registros de datos. Los datos pueden ser escritos o leídos en los puertos si y sólo si, previamente se han programado sus DDR's respectivos. Una vez hecho esto último los datos son capturados en los "latches" de cada uno de los registros. Los puertos pueden ser programados como entradas, salidas o una combinación de estos en forma individual por cada uno de los bits del DDR respectivo.

Algunos dispositivos incluyen un número de líneas de entrada solamente. Esas líneas no tienen DDR y tienen solamente registros de lectura.

Tabla 1.8.2 Registro de Acceso al Puerto de Datos

R/$\overline{W}$	Bit DDR	Resultado
0	0	El pin E/S es un modo de entrada. El dato es escrito a el latch del dato de salida.
0	1	El dato es escrito en el latch del dato de salida y a la salida del pin E/S.
1	0	El estado del pin E/S es de lectura.
1	1	El pin E/S está en modo de salida. El latch del dato de salida es de lectura.

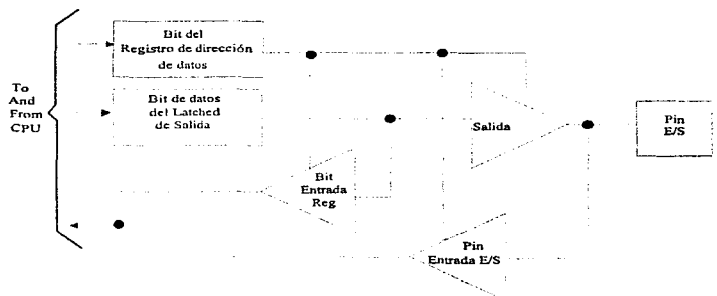


Figura 1.8.4 Circuito típico de Puertos de E/S

1.8.1.8 Descripción del Timer[19]

Generalidades

Todos los dispositivos de la familia M6805 HMOS/M146805 CMOS contienen por lo menos un timer en el chip. El timer está compuesto básicamente por un prescalador de 7 bits y un contador de 8 bits y una interrupción lógica. Los dispositivos

[19]Para más información sobre el temporizador de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A27, A87 y A151.

M6805 HMOS y M146805 CMOS difieren significativamente en dos áreas. Primero, la entrada del timer, como muestra la figura 1.8.5 y 1.8.6 se programa diferente. En la familia M146805 CMOS, la entrada se selecciona programando los **bits 4 y 5** del registro de control del timer (TCR). En la familia M6805 HMOS estos bits son programables enmascarables (excepto para los MC6805R3 y M6805U3). La segunda diferencia es el prescalador es programable por software en al familia MC146805 CMOS y programado enmascarable en la familia M6805 HMOS.

La interrupción por timer opera de manera similar a una interrupción externa, comúnmente, el usuario deberá limpiar el bit de petición de interrupción (TCR7) para prevenir que ocurra un segundo servicio de interrupción por timer.

Descripciones de los timer HMOS y CMOS siguen con más detalle. La versión EPROM permite operaciones con cualquiera de los timer CMOS o HMOS vía el registro de opción enmascarado (MOR).

Timer de la Familia M146805 CMOS

Generalidades

El timer del MCU contiene un contador de 8 bits programable por software con 7 bits seleccionables por software para el prescalador como muestra la figura 1.8.5. El contador puede ser precargado bajo un programa de control y decrementado hacia cero. Cuando el contador se decrementa a cero, el bit de petición de interrupción por timer. esto es el **bit 7** del registro de control del timer (TCR) es colocado. Entonces la interrupción por timer no es enmascarada, esto es el **bit 6** del **TCR** y el **bit I** en el

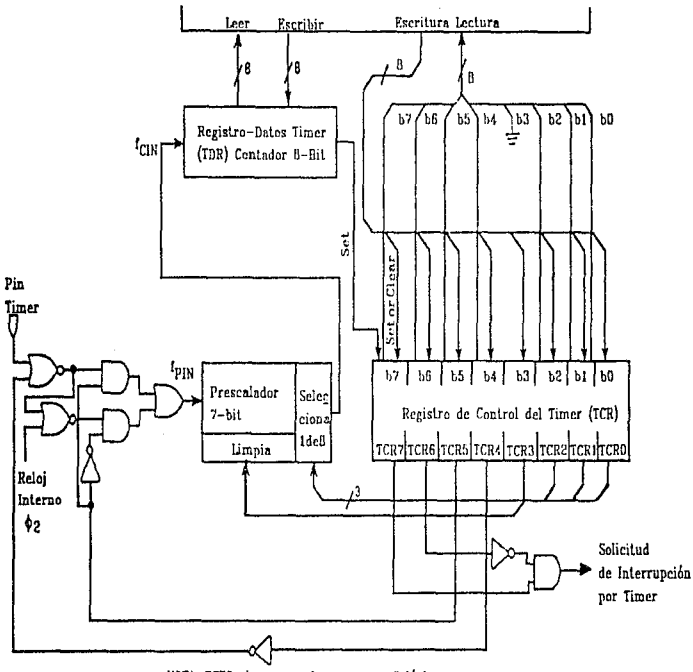
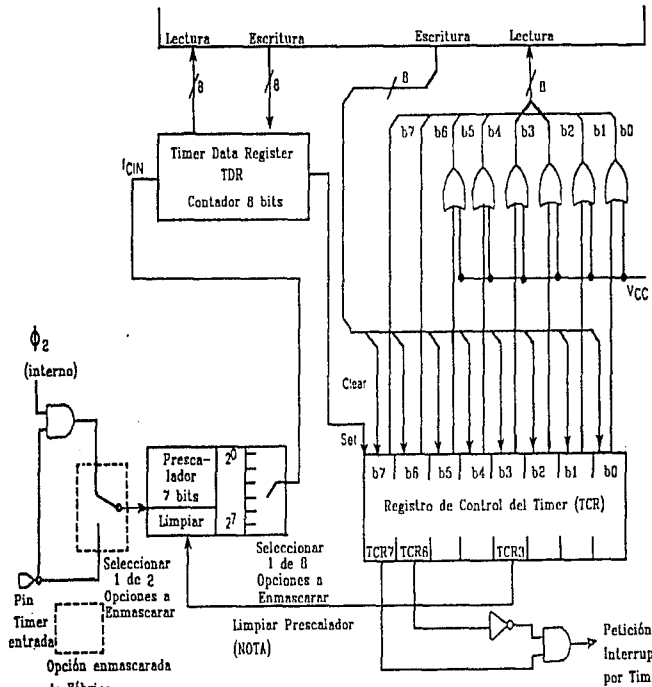


Figura 1.8.5 Diagrama de Bloques del Timer de la familia M146805 CMOS



1.8.6 Diagrama de bloques del Timer de la familia 6805 HMOS

CCR sean limpiados entonces el procesador recibe una interrupción. Después de la terminación de la actual instrucción el procesador procede a almacenar los registros apropiados al stack y entonces va a buscar el vector de direcciones del timer en orden para empezar la rutina de servicio.

El contador continua la cuenta después de alcanzar el cero, permitiendo al software determinar el numero de entradas internas o externas de reloj, desde que el bit de petición de interrupción por timer fue enviado, el contador puede ser leído en cualquier momento por el procesador sin interrumpir la cuenta. El contenido del contador llega a ser estable antes de leer una porción de un ciclo por lo tanto no hay cambio durante la lectura. El bit de petición de interrupción por timer permanece en ese estado hasta que es limpiado por software. Si una limpieza (**escribir TCR7=0**) ocurre antes de que la interrupción por timer sea atendida, la interrupción es perdida. El bit del **TCR** y puede ser usado como un buscador del estado del bit en un modo de no interrupción de operación (**TCR6=1**).

El prescalador es un divisor de 7 bits el cual es usado para extender a la longitud máxima del timer, **bit 0, bit 1 y el bit 2** del **TCR** son programados para cambiar apropiadamente la salida del procesador el cual es usado como la salida del contador. El procesador no puede escribir en el o leer del prescalador; comúnmente su contenido es limpiado con puros "0s" para la operación de escritura en el **TCR** cuando el **bit 3** del dato escrito es igual a 1. Esto permite el truncamiento del contador.

La entrada del timer puede ser configurada en una de tres diferentes modos de operación, además de deshabilitar el modo dependiendo del valor escrito en el TCR4 y TCR5. Referido al párrafo del bit del registro de control del timer.

Modo 1 de entrada al Timer.

Si el TCR4 y el TCR5 son ambos programados en "0" la entrada a el timer es la del reloj interno y el pin de entrada a el timer es deshabilitado. El reloj interno puede ser usado para la generación de interrupciones periódicas, también como una referencia en frecuencia para tomar medidas. El reloj interno es la instrucción del ciclo de reloj. Durante una instrucción WAIT, el reloj interno del timer continua corriendo en su ciclo normal.

Modo 2 de entrada al Timer

Con TCR4=1 y TCR5=0 el reloj interno y el pin de entrada al timer están conectadas a una compuerta AND para formar la señal de entrada al timer. Este modo puede ser usado para medir el ancho de los pulsos externos. El pulso externo simplifica el número de pulsos del el reloj interno dependiendo de la duración del pulso externo. La resolución de la medida de este modo es ±1 reloj.

Modo 3 de entrada al Timer

Si TCR4=0 y TCR5=1 entonces todas las entradas al timer están deshabilitadas.

Modo 4 de entrada al Timer

Si TCR4=1 y TCR5=0 el reloj de entrada interno al timer es deshabilitado y el pin de entrada al TIMER llega a ser la entrada a el timer. En este modo el timer puede ser usado para contar eventos externos también como una frecuencia externa para la generación de interrupciones periódicas. El contador es sincronizado por el borde de

bajada de una señal externa.

La figura 1.8.5 muestra un diagrama de bloque del subsistema del timer. Habilitar el reset o la instrucción **STOP** causa que en el contador se ponga **SF0.**

Registro de Control del Timer (TCR)

Los 8 bits en el **TCR** son usados para controlar varias funciones tales como configuración del modo de operación, la razón a dividir por el prescalador y generar la señal de petición de interrupción por timer. Una descripción de cada función de los bits del **TCR** es proporcionada abajo. Todos los bits en este registro excepto el **bit 3** son bits de lectura/escritura.

BIT	b7	b6	b5	b4	b3	b2	b1	b0
OPCIÓN	TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0

TCR7 Bit de petición de interrupción del timer: el bit es usado para indicar la interrupción por timer cuando este es un "1" lógico.

1 Se habilita cada vez que el contador decrementa a cero, o bajo el control de un programa.

0 Limpiado por un reset externo, reset de encendido, instrucción de **STOP** o bajo control de un programa.

TCR6 **Bit de interrupción del timer enmascarado: cuando este bit es "1" lógico**

inhibe la interrupción al **TIMER** al procesador.

1- Es habilitado por el reset externo, reset de encendido, instrucción de **STOP** o control del programa.

0- Limpiado bajo el control del programa.

TCR5 Bit selecciona la fuente del reloj externa o interna de entrada que puede ser el pin externo del TIMER o el reloj interno (sin afectar el reset).

1 Selecciona la fuente externa de reloj.

0 Selecciona la fuente interna de reloj.

TCR4 Bit de habilitación externo: el bit de control usado para habilitar el pin externo del timer (sin ser afectado por el reset).

1 Habilita el pin externo del timer.

0 Deshabilita el pin externo del timer.

Resumen de las opciones de la fuente del reloj del Timer

TCR5	TCR4	Opción
0	0	Reloj interno al Timer
0	1	AND entre el reloj interno y el pin TIMER al timer
1	0	Entrada para deshabilitar el timer
1	1	Pin TIMER al timer

Referencia en la figura 1.8.5 para la representación lógica

TCR3 Bit reset del prescalador del Timer: escribir un "1" en este bit inicializa el prescalador a cero. Una lectura de su situación comúnmente indica "0" (sin ser afectado por el reset).

TCR2, TCR1, TCR0 Bits de selección del prescalador: decodificacion para seleccionar una de las ocho etapas en el prescalador (sin ser afectado por el reset).

Timer de la familia M6805 HMOS

El diagrama de bloques del timer para los miembros de esa familia es mostrada en la figura 1.8.6. Este timer consiste en un contador de 8 bits programable por software (registro de datos del timer, **TDR**) el cual se decrementa hacia cero por una entrada de reloj de un prescalador. La entrada del reloj del prescalador es recibido de cualquiera de los pines **TIMER** vía una fuente externa o de la señal **ϕ2** del MCU. La entrada del reloj al prescalador es determinada por una opción enmascarada cuando el MCU es fabricado.

La opción enmascarada permite al prescalador ser disparado directamente por el pin **TIMER** externo o uniendo en una compuerta la señal ϕ2 del MCU. Cuando la señal ϕ2 es usada como una fuente de reloj, solamente puede ser aplicado cuando el pin **TIMER** es un uno lógico. Esto permite al usuario formar un pulso para medir el ancho del pulso introducido por el pin **TIMER** de entrada. Para proveer una entrada ϕ2 continua al prescalador en esta configuración, solamente es necesario conectar el

pin **TIMER** a V_{CC}.

El prescalador selecciona la razón a dividir a través de una opción enmascarada la cual es determinada cuando el MCU es fabricado. Esta opción permite al **TDR** ser disparado con cada pulso de reloj que entra al prescalador (2^0), o por el 128avo. pulso de reloj que entra al prescalador (2^7) o por alguna otra potencia de 2.

El **TDR** (contador de 8 bits) puede ser cargado bajo el control del programa y es decrementado hasta ser cero por cada salida del prescalador. Una vez que el **TDR** se ha decrementado a cero se colocan los 7 bits del registro de control del timer (**TDR**) para generar una petición de interrupción por timer. El **bit 6** del **TCR** puede ser fijado por software para inhibir la petición de interrupción por timer o limpiado por software al pasar la interrupción al procesador, lo que permite limpiar el **bit I**. Ya que el **bit 8** del contador (**TDR**) continua la cuenta (decreciente) después cae a través de $FF a cero; puede ser leído en algún tiempo por el procesador sin distorcionar la longitud del tiempo desde que ha ocurrido una interrupción por timer sin distorcionar el proceso de conteo. Una vez que el procesador recibe una interrupción por timer, el MCU responde para salvar el presente estado del CPU en el stack busca el vector del timer y ejecutar la rutina de interrupción. El procesador es sensible al nivel de la línea de petición de interrupción por timer; por lo tanto, si la interrupción está enmascarada (el **bit I** fijo) el **bit 7** del **TCR** puede ser limpiado por la rutina de servicio de interrupción del timer sin generar una interrupción. Cuando se esta atendiendo una interrupción del timer, el **bit 7 del TCR** deberá ser limpiado por la rutina de servicio de interrupción del timer.

Al encender o inicializar, el prescalador y el **TDR** (contador de 8 bits) son todos inicializados con unos lógicos, el **TCR** el **bit 7** es limpiado y el **bit 6** del **TCR** es

fijado.

NOTA

La descripción de arriba no se aplica a todos los miembros EPROM de la familia M6805 HMOS/MC146805 CMOS (o a los MC6805R3 y MC6805U3) esto es porque los MCU's EPROM usan **TCR de 0-5 bits** para seleccionar la salida del prescalador y dividirla en una razón determinando la fuente del reloj y limpiando el prescalador, las versiones de EPROM pueden también ser programadas vía el **MOR** permitiendo al prescalador ser programado por software.

1.8.1.9 Convertidor Analógico Digital A/D[20]

El MCU MC6805R2 y el MCU MC68705R3 ambos tienen un convertidor A/D de 8 bits implementado en el chip. Este convertidor A/D usa una técnica de aproximación sucesiva. Este posee 4 entradas analógicas externas, vía del puerto D. Cuatro referencias analógicas internas pueden ser seleccionadas para propósitos de calibración (V_{RH}, V_{RL}, $V_{RH}/2$, $V_{RH}/4$). La precisión de esos canales internos no necesariamente se encuentra en la precisión de las especificaciones de los canales externos.

La selección del multiplexor es controlada por el registro de control del convertidor A/D **(ACR) los bits 0, 1 y 2**; se ven en la tabla 1.8.3. Este registro es

[20]Para mayor información sobre el convertidor A/D del MCU 6805R3, consultar el apéndice A en la página A100.

limpiado durante la condición de reset.

Tabla 1.8.3 Selección del multiplexor de entrada del A/D

Registro de control A/D	Entrada A/D Seleccionada	Salida (Hex) Min	Tip	Max
0 0 0	AN0			
0 0 1	AN1			
0 1 0	AN2			
0 1 1	AN3			
1 0 1	VRH*	FE	FF	FF
1 0 1	VRL*	00	00	01
1 1 0	VRH/4*	3F	40	41
1 1 1	VRH/2*	7F	80	81

Cada vez que se escribe en el **ACR**, el desarrollo de conversión es abortado, la bandera de conversión terminada (el **bit 7** del **ACR**) es limpiado y las entradas seleccionadas son muestreadas y guardadas internamente.

El convertidor opera continuamente usando 30 ciclos de máquina (incluyendo los 5 ciclos del tiempo de muestreo) para completar una conversión de la entrada analógica muestreada. Cuando la conversión es completada, la muestra digitalizada o el valor digital es colocado en el registro de resultados del convertidor A/D **(ARR)**, la

bandera de conversión completada es colocada, la entrada seleccionada es otra vez muestreada y una nueva conversión es iniciada. La conversión de datos es actualizada en la parte interna del ciclo que no es usado para lectura. Esto asegura que los datos son válidos y estables continuamente y disponibles después de la conversión inicial.

NOTA

Transitorios negativos o alguna otra linea analógica durante la conversión produciría un error de lectura.

La razón métrica del A/D la da la referencia de dos voltajes (V_{RH} y V_{RL}) V_{RH} son suministradas al convertidor vía los pines del puerto D. Una entrada más grande que lo convierte a $FF y no proporciona una indicación de overflow (sobrecarga). Para conversiones de razón métrica, la fuente de cada entrada analógica usaría V_{RH} como el suministro de voltaje mas alto y ser referenciada por V_{RL} o tierra.

1.8.1.10 Comparador de fase

El comparador de fase compara la frecuencia de fase de f_{VAR} y f_{REF} y de acuerdo a la fase genera relaciones de tres niveles de salida **(1, 0 o Hi-Z), ϕCOMP**, se muestra en al figura 1.8.7. La forma de onda de salida es entonces, integrada, amplificada y el voltaje resultante de DC es aplicado al voltaje del oscilador del controlador.

En la práctica las características lineales alrededor de la región de un estado estable no puede ser alcanzada debido a propagaciones internas de retardos. De este modo, el comparador muestra características no lineales y para sistemas con amarre de

fase, resulta en una "reacción" creando un efecto de bandas laterales y distorsión de FM. Para evitar este efecto, un pulso muy corto es introducido periódicamente en el sistema. El ciclo, en turno intenta cancelar esta interferencia y de este modo conduce al comparador de fase a esa zona lineal.

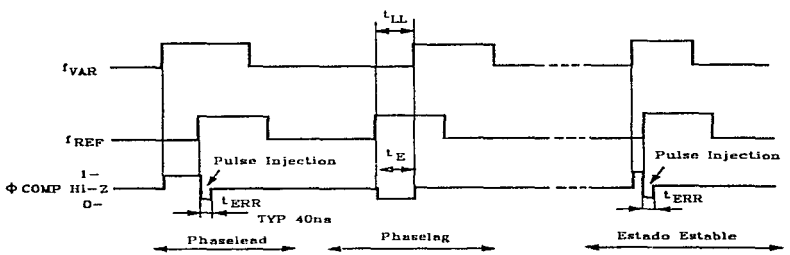


Figura 1.8.7 Forma de curvas del Comparador de Fase

1.8.2 Descripción del software[21]

1.8.2.1 Introducción[22]

Durante los comienzos de los años 1970, los microprocesadores (MPU) y microcomputadores (MCU) ayudaron a facilitar los diseños de hardware proporcionando un hardware más inteligente. Sin embargo, dado que la potencia de cualquier MPU o MCU es el resultado de programas de software, surgió una escasez de ingenieros de software. Así, como los MPU y MCU reducen costos de hardware, el desarrollo del software eleva los costos. Como resultado, los diseñadores actuales de sistemas son más cuidadosos en considerar el software y el costo de soporte del sistema.

Procesadores tales como las familias M6805 HMOS/M6805 CMOS, las cuales están diseñadas para incluir las características de programación inherentes de las minicomputadoras, requieren menos esfuerzo para el programador y permiten que los diseños de sistemas sean mucho más eficientes. La importancia de un software amigable al usuario, en mini y mainframe es un factor ampliamente aceptado. Un software fácil al usuario es la llave para escribir y mantener programas eficientes.

La arquitectura de la familia M6805 HMOS/M146805 CMOS se basa en el

---

[21]Traducción y resumen realizado por Luz María Castillo Jiménez del manual "M6805 HMOS/M146805 CMOS Family Users Manual, Chapter 2, Software Description"

[22]Para más información sobre conjunto de instrucciones de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A39, A104, A163.

modelo de Von Neumann el cual coloca todos los datos, programa y espacios de E/S en un simple mapa de memoria. Así, un simple mapa de memoria puede soportar las instrucciones y propósitos especiales del conjunto de instrucciones de la familia M6805 HMOS/M146805 CMOS.

Los resultados de esto son: un conjunto de instrucciones pequeño, regular y fácil de recordar.

Un conjunto de instrucciones regular es simétrico en cuanto a que, para generar más instrucciones se utilizan instrucciones complementarias. Algunas de estas instrucciones (más complementos) se listan a continuación:

LDA	-	STA	Carga y almacena
INC	-	DEC	Incremento y decremento
BEQ	-	BNE	Salta si es igual y salta si no es igual
ADD	-	SUB	Adición y substracción
AND	-	ORA	AND lógica y OR lógica
BCLR	-	BSET	Limpia bit y conjunto de bits
ROR	-	ROL	Rota a la derecha y rota a la izquierda
JSR	-	RTS	Salta a subrutina y retorna de una subrutina

La simetría que ofrece el conjunto de instrucciones de la familia M6805 HMOS/M146805 CMOS significa que el programador sólo necesita recordar alrededor de 30 a 40 instrucciones separadas para conocer el conjunto entero de instrucciones. La familia M146805 HMOS tiene 59 instrucciones y la familia M146805 CMOS tiene

61. Las 2 instrucciones adicionales de la familia M146805 CMOS son las instrucciones STOP y WAIT las cuales habilitan los modos standby de baja potencia del CMOS.

El conjunto de instrucciones se expande mediante el uso de una gran variedad de modos de direccionamiento. Los modos de direccionamiento, los cuales son parte de la herencia de las minicomputadoras de la familia M6805 HMOS/M146805 CMOS, expanden el conjunto de instrucciones permitiendo que el programador especifique como debe manipularse el dato para una instrucción en particular. Las 59/61 instrucciones separadas engrandecen los siete modos de direccionamiento, expendiéndolos en 207/209 códigos de operación; sin embargo, el programador sólo necesita recordar 66/68 (59/61 instrucciones más 7 modos de direccionamiento), en lugar de 207/209.

1.8.2.2 Conjunto de registros[23]

Cada miembro de la familia M6805 HMOS/ M146805 CMOS contiene 5 registros como se muestra en la figura 1.8.8. El acumulador (A) y el registro índice (X) se utilizan como registros de trabajo del programa. El estado del registro de código de condición (CCR) se utiliza para indicar el estado actual del procesador del programa. El contador de programa (PC) contiene la dirección de memoria de la siguiente instrucción que el procesador ejecutará. El stack pointer (SP, apuntador a pila)

[23]Para más información sobre los registros de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A13, A74, A137.

contiene la dirección de la siguiente localidad vacía de la pila.

NOTA:

El tamaño del stack pointer y del program counter está determinado por el tamaño de la memoria que el dispositivo pueda accesar; por ejemplo, un mapa de memoria de 8K requiere 13 bits de stack pointer y de program counter.

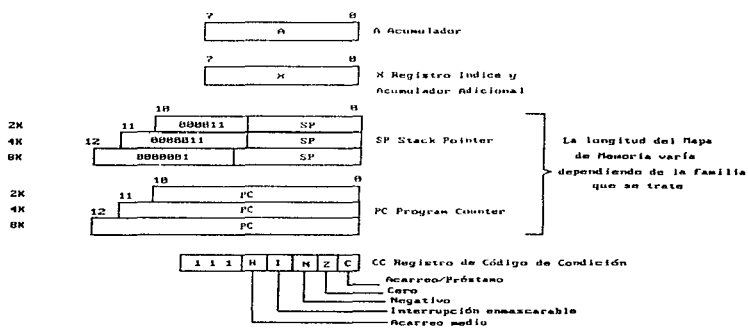


Figura 1.8.8 Arquitectura de lo registros de la familia M6805 HMOS/M146805 CMOS

Acumulador (A)

El registro A es un registro de propósito general de 8 bits que utiliza el programa para cálculos aritméticos y manipulación de datos. El conjunto completo de instrucciones de lectura/modificación/escritura, operan sobre el registro A. El acumulador se utiliza en las instrucciones de registro/memoria para la manipulación de datos y cálculos aritméticos. A continuación se muestra un ejemplo que utiliza el acumulador para sumar el contenido de dos localidades de memoria:

```
        ORG  $49
B6  50  LDA  $50    Carga el acumulador con el contenido de la localidad
                    de memoria $50
BB  87  ADD  $87    Suma el contenido de la localidad de memoria $87 al
                    acumulador
B7  3C  STA  $3C    Almacena el contenido del acumulador en la
                    localidad de memoria $3C
```

Registro de Índice (X)

El registro índice se utiliza en los modos de direccionamiento indexado o se utiliza como un acumulador auxiliar. Este registro es de 8 bits y puede cargarse directamente o desde memoria, su contenido puede almacenarse en memoria, o puede compararse con la memoria.

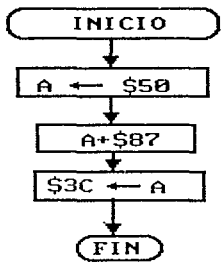


Diagrama de flujo No. 1

En instrucciones indexadas, el registro X proporciona un valor de 8 bits que se añade a una instrucción que proporciona un valor, para crear una dirección efectiva.

El registro X también se utiliza en la familia M6805 HMOS/M146805 CMOS para ciertos cálculos y manipulación de datos. Al igual que en el acumulador el conjunto completo de instrucciones de lectura/modificación/escritura también operan en el registro X . Algunas secuencias de instrucciones, las cuales no utilizan el registro X para direccionamiento indexado pueden usar X como una celda de almacenamiento temporal o acumulador.

El siguiente ejemplo muestra un uso típico de un registro de índice en uno de los modos de direccionamiento. El ejemplo ejecuta un movimiento de bloque de longitud

BCNT.

```
        LDX #BCNT           Devuelve la longitud
REPEAT  LDA SOURCE,X        Devuelve el dato
        STA DESTIN.X        Lo almacena
        DECX                Siguiente
        BNE REPEAT          Repite si no es cero
```

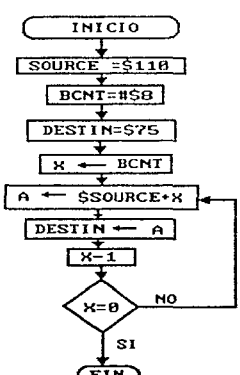


Diagrama de flujo No. 2

El registro X puede utilizarse en eventos de conteo, puede incrementarse o decrementarse. Las instrucciones INCX o DECX pueden utilizarse para controlar el conteo. Para cada decremento o incremento el registro X, comienza reconociendo el valor, y entonces compara el contenido del registro X con el contenido de la localidad de memoria (o un número específico). Después de un cierto número de eventos puede finalizarse un ciclo o puede realizarse una bifurcación.

La siguiente rutina usa el registro índice como un contador para una rutina que evita el rebote en el teclado de CNTX6 CMSO (o CNTX8, HMOS).

```
AE   FF   DBNCE   LDX  #CNT        CNT = 255 en este ejemplo
5A        AGAIN   DECX
26   FD           BNE AGAIN
```

Contador de programa (PC)

El PC contiene la dirección de memoria de la siguiente instrucción que se llamará y ejecutará. Normalmente el PC apunta a la siguiente instrucción, sin embargo, el PC puede alterarse mediante interrupciones o instrucciones. Durante una interrupción válida, el PC se carga con el vector de interrupción apropiado. Las instrucciones de salto y ramificación modifican el PC, así que la siguiente instrucción a ejecutar no es necesariamente la siguiente instrucción en memoria física. El tamaño actual de PC depende del tamaño del espacio de direccionamiento de los miembros de las familias y de los rangos actuales que van de 11 a 13 bits.

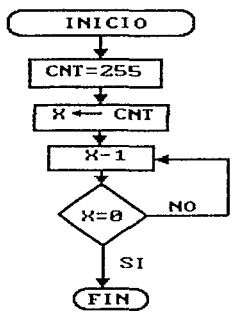


Diagrama de flujo No. 3

Apuntador de pila (SP)

El arreglo de pila es un área de memoria RAM que se utiliza para almacenar temporalmente información importante. Es una secuencia de registros (localidades de memoria) que se utilizan de forma LIFO (últimos en entrar primeros en salir). Un SP se utiliza para especificar donde se localiza el último registro o donde deberá ir. La pila debe leerse y localizarse en RAM.

Las interrupciones y subrutinas deben usar la pila para salvar temporalmente datos importantes. El SP se utiliza para almacenar automáticamente la dirección de

retorno en llamadas de subrutina (2 bytes del PC) y almacenar automáticamente todos los registros (5 bytes A, X, PC y CC) durante las interrupciones. Los registros salvados pueden colocarse en la pila, permitiendo también anidar subrutinas e interrupciones. las subrutinas pueden interrumpirse y las interrupciones pueden llamar subrutinas. La anidación de subrutinas e interrupciones sólo puede ocurrir un máximo de veces, esto se describe a continuación.

En la familia de dispositivos M6805 HMOS/ M146805 CMOS el tamaño actual del apuntador de pila puede variar con el tamaño de memoria de un miembro en particular de la familia. Pero desde la perspectiva del programador, el SP aparece de manera similar en los diferentes miembros. Ambos, el pin de $\overline{\text{RESET}}$ y la instrucción de reinicializar el SP (RSP) reinicializa el SP a su máximo valor ($7F en todos los miembros actuales). El stack pointer de la familia M6805 HMOS/M146805 CMOS siempre apunta a la siguiente localidad libre en la pila. Cada que se agrega (push) se decrementa el SP mientras que cada que se extrae (pull) se incrementa (pull y push no son válidas como instrucciones utilizables en la familia M6805 HMOS/M146805 CMOS).

Al anidar las llamadas a subrutina e interrupciones no se debe sobrecargar el SP. La longitud utilizable de la pila puede variar entre unos y otros dispositivos de las familias M6805 HMOS y M146805 CMOS. En la familia M6805 HMOS la longitud utilizable de la pila es 2^n-1 (donde n es el número de bits del apuntador de pila); sin embargo, en la familia M146805 CMOS la longitud utilizable es 2^n (donde n es el número de bits del apuntador de pila). Cuando la longitud de la pila se excede, el SP puede envolver alrededor del tope de la pila. Esta condición de sobrecarga de la pila

puede evitarse. Un ejemplo para calcular la longitud utilizable de la pila para un dispositivo de la familia M6805 HMOS con un apuntador de pila de 5 bits es: 2^5-1 = 31 bytes máximo. Sin embargo, para un dispositivo de la familia M146805 CMOS, con 6 bits de SP, el cálculo es 2^6 o 64 bytes máximo.

Un SP de un dispositivo de 5 bits de la familia M6805 HMOS acomoda 15 llamadas a subrutina anidadas (30 bytes), 6 interrupciones (30 bytes) o una combinación de ambas. El programador debe tener cuidado cuando se aproxima a la sobrecarga. Cuando se sobrecarga el SP se puede envolver alrededor, y el contenido puede perderse. El límite de la pila en la familia de 5 bits M6805 HMOS, por ejemplo, está alrededor de 31 bytes, no 32 bytes. El límite de la pila puede ir más allá de acuerdo a las necesidades del programa. Un máximo de 5 niveles de subrutinas anidadas (10 bytes) se acopla con un nivel de interrupción (5 bytes) ocupando solamente 15 bytes del espacio de la pila. La longitud de la pila permitida típicamente es independiente de la cantidad de RAM necesaria para datos.

En la familia M6805 HMOS/ M146805 CMOS, la pila trabaja en la dirección decreciente de la pila, sin embargo, el SP siempre apunta a la siguiente localidad vacía en la pila. El SP se decrementa cada vez que un dato se agrega a la pila y se incrementa cada vez que un dato se extrae de la pila. El SP solamente se modifica durante las operaciones positivas y, excepto por la instrucción RSP, esto no está directamente bajo el control del software. Durante el manejo externo del encendido del reset, y durante una instrucción de reinicialización del apuntador (RSP), el SP se va a su límite superior ($7F).

El orden en el cual los bytes se almacenan y se extraen de la pila se muestran en

la figura 1.8.9. Note que el PC tiene un número fijo de bits. El número de bits variables depende del tamaño disponible de memoria y es particular de cada miembro de la familia (ver figura 1.8.8 para estas relaciones).

Registro de códigos de condición (CCR)

Las familias M6805 HMOS/M146805 CMOS utilizan 5 bits para banderas del código de condición, etiquetadas H, I, N, Z y C, las cuales residen en el registro CCR. Los 3 bits MSB del registro CCR llenan el registro de 8 bits.

La función del código de condición es retener información concerniente a los resultados de la última ejecución de las instrucciones en los datos; cualquier bit o una combinación de bits, excepto el bit 1, se examinan utilizando las instrucciones condicionales de ramificación.

Carry (C)

El bit C es un conjunto de acarreos de la ALU de 8 bits que ocurren durante la última operación aritmética. Esto también ocurre durante la ejecución de instrucciones de corrimiento, rotación y de prueba de bits.

El bit C se coloca en uno de seis casos:

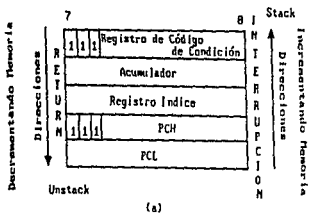
1. Se coloca durante una instrucción de adición si el resultado de las adiciones produce un acarreo de salida de 8 bits en la ALU (Unidad Aritmética y Lógica).
2. Para las instrucciones de resta y comparación, se coloca cuando el valor

absoluto del substraendo es más largo que el valor absoluto del minuendo. Generalmente esto implica un préstamo.

3. Se cambia durante las instrucciones de corrimiento y rotación. Para estas instrucciones el bit trasladado fuera del acumulador llega al bit C.

Notas:

1. En los dispositivos de todas las familias, el stack pointer se decrementa durante las operaciones "push", el PCL se almacena primero, seguido por el PCH, etc. Las operaciones "pull" se realizando en orden inverso a las anteriores.
2. En la familia M6805 HMOS siempre se coloca el bit PC, sin embargo, en la familia M146805 CMOS el bit OC siempre se limpia.



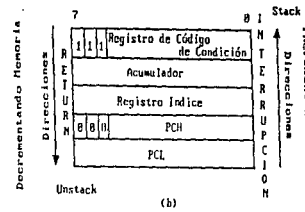


Figura 1.8.9 Orden de almacenamiento

4. Se coloca cuando una instrucción SEC se ejecuta.
5. Se coloca cuando una instrucción COM se ejecuta.
6. Se coloca si un bit de prueba y un bit de bifurcación se encienden.

Dos instrucciones: adición con acarreo (ADC) y sustracción con acarreo (SBC), utilizan el bit de acarreo como parte de la instrucción. Esto simplifica la adición o sustracción de números que son más grandes de 8 bits. El bit de acarreo puede examinarse con varias instrucciones de ramificación condicionales.

Cero (Z)

El bit Z se coloca si el resultado de la última manipulación de datos, aritmética o lógica, fue cero. El bit Z se coloca sólo si todos los 8 bits del resultado son cero; en otro caso, se limpia.

El bit Z puede utilizarse para causar una bifurcación con las instrucciones BHI, BLS, BNE o BEQ. Cuando se utiliza la instrucción BHI, ambos, el bit C y el bit Z, se usan para la bifurcación.

El bit Z puede utilizarse para iniciar una bifurcación después de que los contenidos de A o X sean iguales a los contenidos de una localidad de memoria. Por ejemplo, el acumulador puede comparar los contenidos de una localidad de memoria y entonces los 8 bits resultantes son todos ceros (el bit Z se coloca), de la instrucción

BEQ puede resultar una bifurcación. Recíprocamente, si se realiza la misma comparación fue hecha y se utiliza una instrucción BNE, puede resultar una bifurcación después de cada comparación mientras que los 8 bits resultantes son todos ceros (bit Z se coloca).

Negativa (N)

El bit N se coloca cuando el bit 7 del resultado de la última manipulación de datos, operación lógica o aritmética se realiza. Esto indica que el resultado de la operación fue negativo. El bit N se limpia mediante las instrucciones CLR y LSR. Otras instrucciones afectan el bit N, esta condición se determina por el bit 7 del resultado.

El bit N puede usarse para causar una bifurcación si se coloca mediante el uso de la instrucción BMI. Igualmente el bit N puede utilizarse por una bifurcación si se limpia utilizando la instrucción BPL. En un caso se examina para un resultado negativo y en el otro se examina para un resultado positivo.

El bit N puede utilizarse para iniciar una bifurcación después de una comparación de 2 números. Por ejemplo, el contenido del registro X puede compararse con los contenidos de la localidad de memoria M y hacer una bifurcación si N = 1. Usando la instrucción CPX, el bit N puede permanecer limpio y no efectuar la bifurcación, si la longitud del contenido del registro X es mayor o igual al contenido de M. Sin embargo, si el contador del registro X es menor que el contenido de M, el bit N queda en 1 y la ramificación puede iniciarse (usando la instrucción BMI).

Medio Acarreo (H)

El bit N se coloca cuando ocurre un acarreo entre los bits 4 y 3 bits y durante una instrucción ADD o ADC. La bandera H puede utilizarse en subrutinas de adición BCD puesto que cada dígito binario codificado en decimal está contenido en los bits 0-3 (menos significativos) o 4-7. Así, cuando la suma de 2 BCD menos significativos da como resultado un acarreo del bit 3 en la posición 4, el bit H se coloca.

Interrupción enmascarada (I)

Cuando se coloca el bit I, la interrupción externa y la interrupción del timer se enmascaran. Limpiando el bit I se logra habilitar las interrupciones. Si una interrupción ocurre mientras el bit I se coloca, la interrupción se cierra internamente y se mantiene mientras el bit I se limpia. Entonces el vector de interrupción se ejecuta normalmente.

Excepto para cuando se aplica una interrupción externa ($\overline{INT}$ o $\overline{IRQ}$), el bit I se controla mediante instrucciones del programa. Algunas instrucciones de programa cambian el bit I sólo como un resultado en la instrucción, mientras que otras realizan estos cambios como parte de la instrucción. Por ejemplo, CLI limpia el bit I y SEI coloca el bit I, sin embargo, SWI coloca automáticamente el bit I como parte de la instrucción de interrupción. En la familia M146805 CMOS las instrucciones STOP y WAIT colocan automáticamente el bit I como parte de la instrucción.

NOTA

La instrucción SWI y $\overline{\text{RESET}}$ son las únicas interrupciones no enmascarables en las familias M6805 HMOS/M146805 CMOS.

1.8.3 Modos de direccionamiento[24]

La potencia de cualquier computadora consiste en su habilidad para accesar a memoria. Los modos de direccionamiento del procesador permiten esta capacidad. La familia M6805 HMOS/ M146805 CMOS tienen un conjunto de modos de direccionamiento que satisface estos criterios extremadamente bien.

Los modos de direccionamiento definen la manera en la cual una instrucción obtiene el dato requerido para su ejecución. Una instrucción , a causa de los diferentes modos de direccionamiento, puede accesar su operando en uno de cinco diferentes modos de direccionamiento. En esta forma, el modo de direccionamiento extiende las 59 instrucciones básicas de la familia M6805 HMOS (61 para la familia M146805 CMOS) en 207 operaciones separadas (209 para la familia M146805 CMOS). Algunos modos de direccionamiento requieren que los 8 bits de código se acompañen por uno de dos bytes adicionales. Estos bytes contienen el dato para las operaciones, la dirección del datos o ambos.

En la descripción el modo de direccionamiento que sigue, se usa el término dirección efectiva (EA). La EA es la dirección en memoria desde la cual el argumento

[24]Para más información sobre los modos de direccionamiento de los MCU's 68705P3, R3 y U3, consultar el apéndice A en las páginas A45, A109, A169.

para una instrucción se toma o almacena. En instrucciones de dos operandos, tal como sumar al acumulador (ADD), uno de los operandos efectivos (el acumulador) es inherente y no se considera un modo de direccionamiento para este.

La descripción y ejemplos de los diferentes modos de direccionamiento de las familias M6805 HMOS/M146805 CMOS se dan en los siguientes párrafos. Para cada modo se muestran varios ejemplos de programas en ensamblador, y uno de los ejemplos se describe en detalle (ORG, EQU y FCB son directivas de ensamblador y no parte del conjunto de instrucciones).

Se usan paréntesis en estos ejemplos para indicar "el contenido de " la localidad o registro referido a; por ejemplo (PC) indica el contenido de la localidad a la cual apunta el PC. El símbolo (:) indica una concatenación de bytes.

En los siguientes ejemplos se asume que inicialmente el contador de programa (PC) apunta a la localidad del primer byte del código de operación.

El primer PC+1 es el primer resultado incremental y muestra que el PC apunta a la localidad inmediata siguiendo el primer byte del código de operación.

La información provista en los ejemplos del programa ensamblador utiliza varios símbolos para identificar los diferentes tipos de números que ocurren en un programa. Estos símbolos incluyen:

1. Un blanco o no símbolo que indica un número en decimal.
2. Un $ precediendo un número inmediatamente indica que es un número hexadecimal; por ejemplo $24 es 24 en hexadecimal o el equivalente de 36 en decimal.
3. Un # indica un operando inmediato y el número se encuentra en la locación

siguiente al código de operación.

Hay 7 diferentes modos de direccionamiento utilizados en las familias M6805 HMOS/M146805 CMOS y se denominan: inherente, inmediato, directo, extendido, indexado, relativo y de manipulación de bit. Los modos de direccionamiento de manipulación de bit e indexado contienen subdivisiones adicionales para incrementar su flexibilidad; por ejemplo 3 adicionales para el modo indexado y 2 para manipulación de bit. Cada uno de estos modos de programación se discuten en los párrafos que siguen.

Modo de direccionamiento inherente

En este modo de direccionamiento no se utiliza dirección efectiva (EA). Las instrucciones del direccionamiento inherente se usan cuando toda la información requerida para la instrucción está ya dentro del CPU, y no se trata de operandos externos, de memoria o del programa. Puesto que toda la información necesaria para acarrear fuera la instrucción está contenida en el código de operación y no se necesitan operandos externos, las instrucciones inherentes sólo requieren un byte.

El siguiente es un ejemplo de una subrutina que limpia todos los registros (acumulador e índice) más el bit C y entonces retorna.

05B9 4F	CLEAR	CLRA	Limpia el acumulador
05BA	97	TAX	Transfiere el contenido del acumulador al Registro índice

05BB 98	CLC	Limpia el bit de acarreo
05BC 81	RTS	Retorno de subrutina

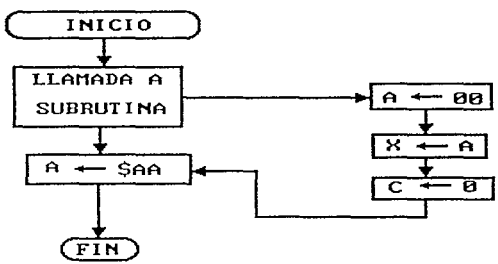


Diagrama de flujo No. 4

La figura 1.8.10 muestra un ejemplo de los pasos que se requieren para ejecutar la instrucción TAX en la subrutina.

Modo de direccionamiento inmediato

La EA de una instrucción en modo de direccionamiento inmediato es la localidad que sigue al código de operación. Este modo se utiliza para mantener un valor o constante el cual se conoce al tiempo en que el programa se escribe, y el cual no cambia durante la ejecución del programa.

Estas son instrucciones de 2 bytes, uno para el código de operación y otro para

el byte del dato inmediato. El direccionamiento inmediato puede usarse mediante cualquier instrucción de registro/memoria.

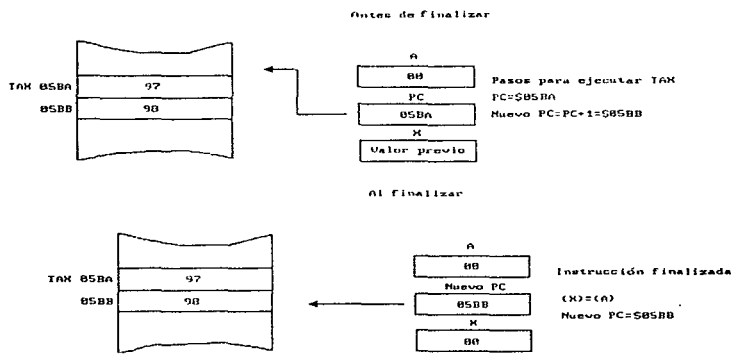


Figura 1.8.10 Ejemplo del modo de direccionamiento inherente

PC + 1 ---> PC
EA = PC
PC + 1 ---> PC

El siguiente es un ejemplo el cual sustrae 5 de los contenidos del acumulador y compara el resultado con 10. La figura 1.8.11 muestra un ejemplo de los pasos que se requieren para ejecutar la instrucción SUB.

05BC B6	4B	LDA $4B	Carga el acumulador desde RAM
05BE A0	05	SUB #5	Resta 5 al acumulador
05C0 A1	0A	CMP #10	Compara el acumulador con 10

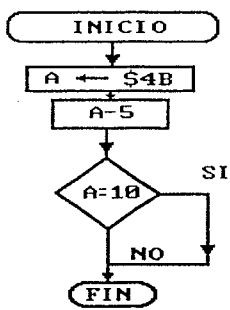


Diagrama de flujo No. 5

- **Modo de direccionamiento extendido**

La EA de una instrucción en modo de direccionamiento extendido está

contenida en los dos bytes siguientes al código de operación. El direccionamiento extendido hace referencia a cualquier localidad en las familias M6805 HMOS/M146805 CMOS; espacios de memoria, E/S, RAM y ROM. El modo de direccionamiento extendido permite que una instrucción accese a toda la memoria. Así, puesto que los 2 bytes siguientes al código de operación contienen 16 bits, el rango de direccionamiento de las familias M6805 HMOS/M146805 CMOS puede extenderse, sin afectar el conjunto de instrucciones o los modos de direccionamiento.

Las instrucciones del modo de direccionamiento extendido son de 3 bytes de longitud, un byte de código de operación más dos bytes de direccionamiento. Todas las instrucciones de registros/memoria, pueden usar direccionamiento extendido.

PC + 1 ---> PC
EA = (PC): (PC + 1)
PC + 2 ---> PC

El siguiente ejemplo carga el contenido de una localidad de memoria (etiquetada COUNT) en el registro índice y entonces salta a una subrutina que provee un retardo.

```
                0800  COUNT   EQU  $800
                1200  DELAY   EQU  $1200
0409  CE         0800         LDX  COUNT    Carga el Registro índice con el
                                             contenido de la localidad $800
040C  CD         1200         JSR  DELAY    Salto a subrutina localizada en
                                             $1200
```

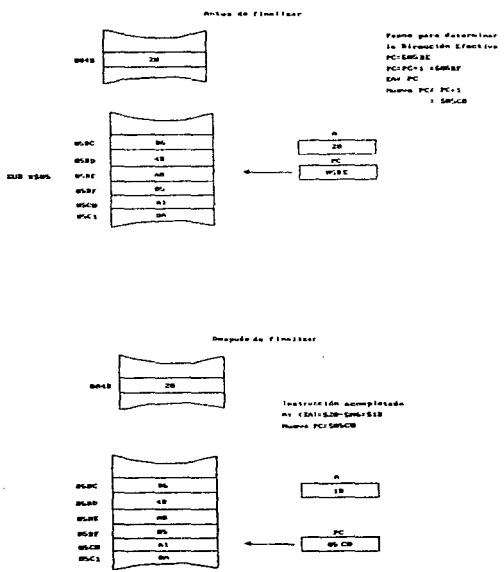


Figura 1.8.11 Ejemplo del modo de direccionamiento inmediato

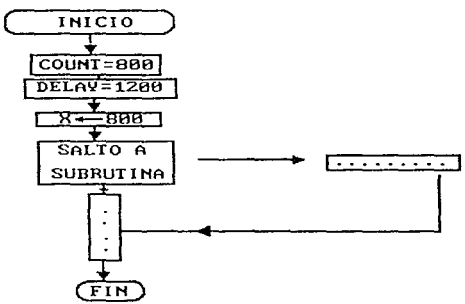


Diagrama de flujo No. 6

La figura 1.8.12 muestra un ejemplo de los pasos que se requieren para determinar la EA de la cual se cargará el registro índice.

Modo de direccionamiento directo

El modo de direccionamiento directo es similar al modo de direccionamiento extendido excepto en que un byte se utiliza para la EA. El direccionamiento directo permite que una instrucción sólo accese una localidad en la página 0 (locaciones $00 - $FF) con una instrucción de 2 bytes. Por lo tanto, los bits de la dirección alta se

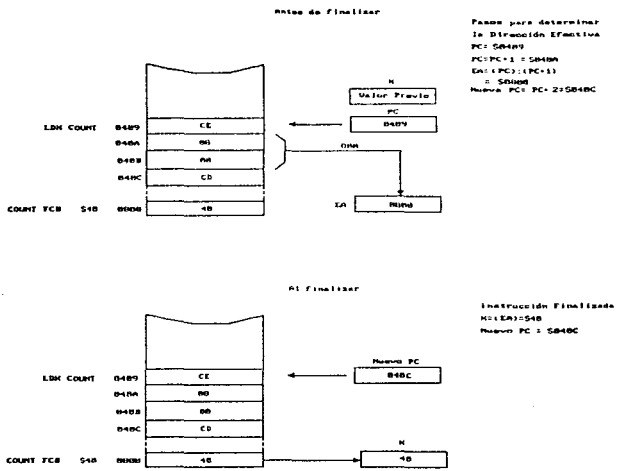


Figura 1.8.12 Ejemplo del modo de direccionamiento extendido

colocan en $00. El modo de direccionamiento directo puede usarse con cualquier instrucción de lectura-modificación-escritura o de registro/memoria y de manipulación

de bit.

El siguiente ejemplo agrega dos números de 16 bits. El resultado se coloca en la localidad del primer número, sin embargo, si el resultado excede los 16 bits, el bit C puede colocarse. La figura 1.8.13 ilustra los pasos que se requieren para determinar la EA de la cual cargará el acumulador con los contenidos de NUM1 (primer número).

```
                    ORG  $10
            NUM1    RMB  2
            NUM2    RMB  2
0527 B6  11         LDA  NUM1+1     Carga el acumulador con el
                                    contenido de la localidad $0011
0529 BB  13         ADD  NUM2+1     Suma el contenido de la
                                    localidad $0013 al acumulador
052B B7  11         STA  NUM1+1     Salva el resultado en la localidad
                                    $0011
052D B6  10         LDA  NUM1           Carga el acumulador con
                                        el contenido de la
                                        localidad $0010
052F B9  12         ADC  NUM2           Suma el contenido de la
                                        localidad $0012 y el bit C
                                        al acumulador
0531 B7  10         STA  NUM1           Salva el resultado en la
                                        localidad $0010
```

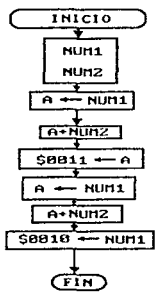


Diagrama de flujo No. 7

Modo de direccionamiento indexado

En el modo de direccionamiento indexado, la EA es variable y depende de 2 factores : (1) el contenido actual del registro índice (X) y (2) el offset contenido en el (los) byte(s) siguientes al código de operación. Existen 3 tipos de direccionamiento indexado en las familias M6805 HMOS/ M146805 CMOS: sin offset, 8 bits de offset y 16 bits de offset. Un buen ensamblador debe usar el modo de direccionamiento

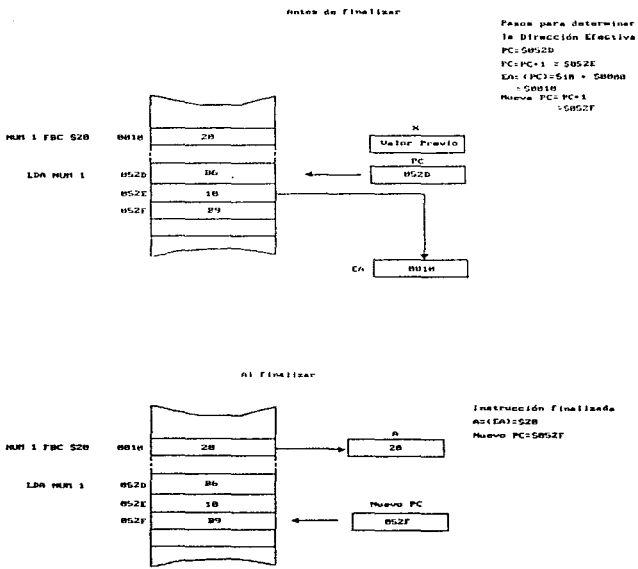


Figura 1.8.13. Ejemplo del modo de direccionamiento directo

indexado el cual requiere el último offset. Cualquiera de los modos de direccionamiento indexados puede utilizarse con cualquier instrucción de lectura-modificación-escritura o de registro/memoria. El direccionamiento indexado de 16 bits de offset utiliza instrucciones de registro/memoria.

Indexado- Sin offset

En este modo el contenido del registro X es la EA, por lo tanto, esta instrucción es de un byte. Este modo se usa para crear una EA la cual apunta al dato en los 256 bytes bajos del espacio de direcciones, incluyendo E/S, RAM y parte de la ROM. Puede utilizarse para mover un apuntador a través de una tabla, o a una localidad de referencia frecuente (por ejemplo una localidad de E/S). mantener la dirección de un dato que se calcula mediante un programa, las instrucciones en modo indexado sin offset sólo usan un byte el del código de operación.

EA = X + $0000
PC + 1 ---> PC

En el siguiente ejemplo, las localidades $45 a $50 se inicializan con blancos (ASCII $20). La figura 1.8.14 ilustra los pasos necesarios para determinar la EA en la cual almacenar el contenido del acumulador en una localidad de memoria apuntada mediante el registro índice.

```
05F0 AE   45          LDX #$45      Inicializa el registro índice con
                                    $45
05F2 A6   20          LDA #$20      Carca el acumulador con $20
05F4 F7       REPEAT  STA X         Almacena el contenido del
                                    acumulador en la localidad
                                    apuntada por el registro índice
05F5 5C               INCX          Siguiente localidad
05F6 A3   51          CPX #$51      Fin
05F8 26   FC          BNE REPEAT    Repite si no es igual a cero
```

Indexado de 8 bits de offset

Para determinar la EA en este modo de direccionamiento, el contenido del registro X se suma al contenido del byte siguiente al código de operación. Este modo de direccionamiento se utiliza en la selección del elemento K-ésimo de una tabla de h elementos.

Para usar este modo la tabla debe comenzar en las 256 localidades bajas de memoria, y puede extenderse a través de las primeras 511 localidades de memoria ($FE es la última localidad en la cual la instrucción puede comenzar) de la familia M6805 HMOS/M146805 CMOS. Todo direccionamiento indexado de 8 bits de offset eficiente de la ROM fomenta la inclusión de tantas tablas como sea posible en la

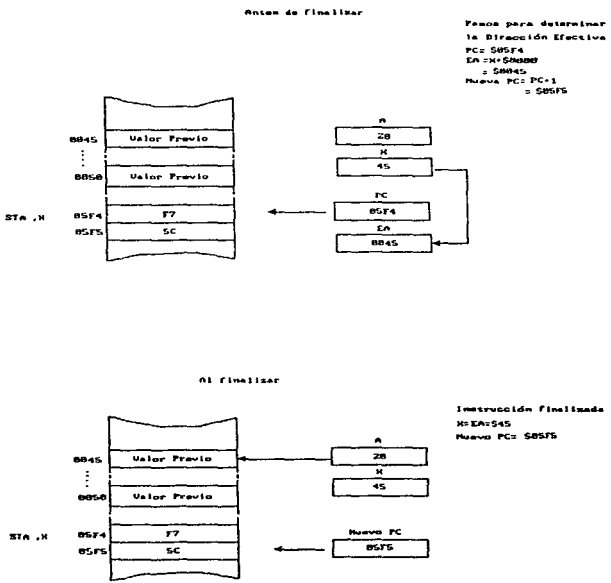


Figura 1.8.14 Ejemplo del modo de direccionamiento indexado sin offset

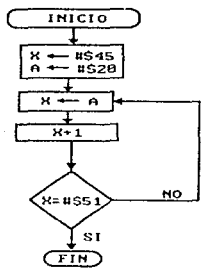


Diagrama de flujo No. 8

puede utilizarse para accesar la ROM, RAM o E/S. Este modo es de dos bytes por instrucción con el offset contenido en el byte siguiente al código de operación. El uso página cero y la uno.

La siguiente subrutina busca una lista, la cual contiene 256 términos separados, para la primer ocurrencia de un valor contenido en el acumulador. La búsqueda comienza en $80 y continua a través de $180 a menos que el contenido del acumulador se iguale con uno de los términos de la lista. La figura1.8.15 muestra los pasos que se requieren para determinar la EA del siguiente término a comparar.

```
                LIST    EQU  $80
                        ORG  $075A
075A 5F         FIND         CLRX        Limpia el registro índice
```

075B	E1	80	REPEAT CMP LIST,X	Compara el acumulador con el contenido de la localidad $80 + X
075D	27	03	BEQ RETURN	Retorna si no se encuentra más
075F	5C		INCX	De lo contrario pasa al siguiente término
0760	26	F9	BNE REPEAT	Si verificó los 256 términos entonces finaliza de lo contrario repite
0672	81		RETURN RTS	

Indexado de 16 bits de offset

La EA para este modo de direccionamiento de dos bytes se calcula sumando los contenidos concatenados de los dos bytes siguientes al código de operación, a los contenidos del registro X. Este modo de direccionamiento se usa de una manera similar al indexado de 8 bits de offset, excepto que, puesto que el offset es de 16 bits las tablas pueden referenciarse a cualquier lugar del espacio de direccionamiento en las familias M6805 HMOS/M146805 CMOS. Para más detalles referentes a la compatibilidad del indexamiento ver los párrafos siguientes. Este modo de direccionamiento es de 3 bytes de instrucción, uno para el código de operación y dos para el valor del offset.

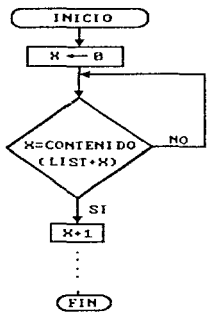


Diagrama de flujo No. 9

PC + 1 ---> PC
EA = (PC): (PC + 1) + X
PC + 2 ---> PC

En el siguiente ejemplo se mueve un bloque de datos de una tabla fuente a una tabla destino. El registro índice contiene la longitud del bloque. La figura 1.8.16 ilustra los pasos que se requieren para determinar la EA de la cual se almacena a memoria .

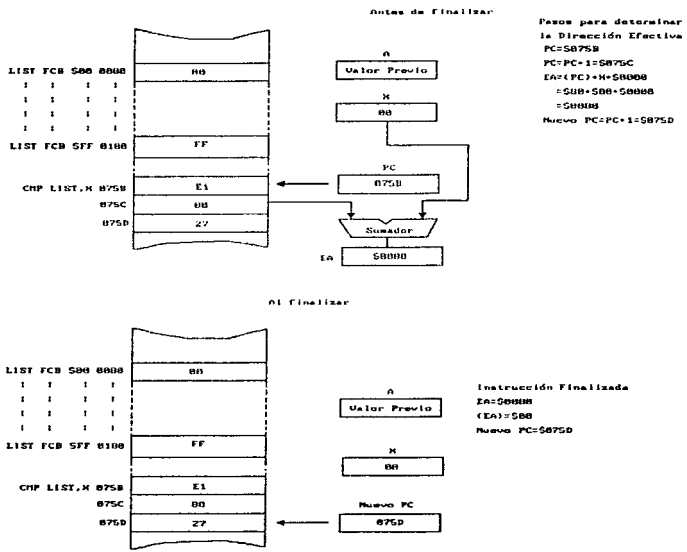


Figura 1.8.15 Ejemplo del modo de direccionamiento indexado de 8 bits de offset

```
                    SOURCE  EQU  $200
                    DESTIN  EQU  $40
0690  AE   04               LDX  #$04
0692  D6   0200   BLKMOV    LDA  SOURCE,X  Carga el acumulador con el
                                           contenido de la localidad
                                           SOURCE+X
0695  E7   40               STA  DESTIN,X  Almacena el contenido del
                                           acumulador en la localidad
                                           DESTIN+X
0698  5A                    DECX           Siguiente localidad
0699  2A   0692             BPL  BLKMOV    Repite si hay más
```

Compatibilidad de indexamiento

Dado que en la familia M6805 HMOS/M146805 CMOS el registro índice sólo es de 8 bits de longitud, y los valores del offset son cero, 8 o 16 bits, el MC6800 utilizado, puede así encontrar que el registro X en la familia M6805 HMOS/M146805 CMOS es mejor utilizado "hacia atrás" del MC6800. Esto es, el offset puede contener la dirección de la tabla y el registro índice contiene el desplazamiento en la tabla.

Modo de direccionamiento relativo

El direccionamiento relativo se usa para bifurcar instrucciones y especificar una

localidad relativa al valor actual del PC. La EA se forma mediante la suma de los contenidos del byte siguiente al código de operación y el valor del PC.

Dado que el PC siempre apunta a la siguiente declaración en línea, mientras la adición comienza a ejecutarse, un byte cero de offset relativo resulta al no bifurcar.

La EA resultante se usa si y sólo si, se toma una bifurcación relativa. Note que el byte siguiente al código de operación se suma al contenido del PC, esto es apuntando a la siguiente instrucción mientras la adición se ejecuta. Las instrucciones de bifurcación siempre contienen 2 bytes de código de máquina: uno para el código de operación y uno para el byte relativo de offset. Dado que es deseable la bifurcación

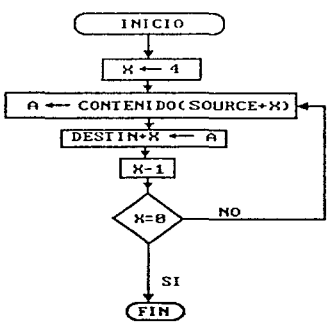


Diagrama de flujo No.10

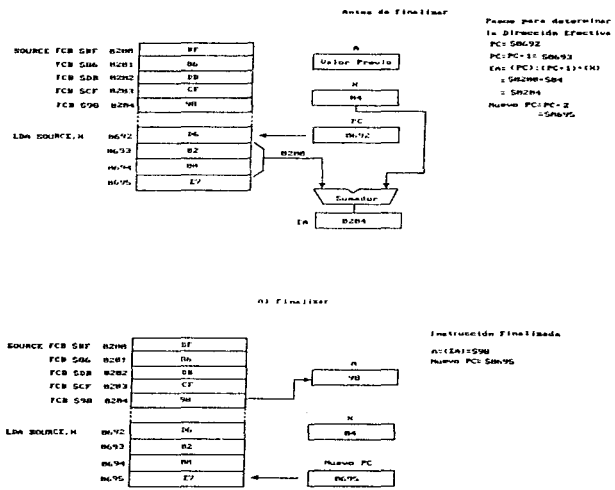


Figura 1.8.16 Ejemplo del modo de direccionamiento Indexado de 16 bits de offset

en cualquier dirección el byte offset es de signo extendido a un rango de -128 a +127 bytes. Sin embargo, el rango efectivo debe calcularse con respecto a la dirección de la siguiente instrucción en línea. Las instrucciones de bifurcaciones relativas constan de 2 bytes; por lo tanto, el rango efectivo de una instrucción de bifurcación al comienzo de la instrucción de bifurcación se define (donde R está definida) como la dirección de la instrucción de bifurcación).

(PC + 2) - 128 <= R <= (PC + 2) + 127

o

PC - 126 <=R <= PC + 129 (sólo para bifurcación condicional)

Un salto JMP o salto a subrutina JSR puede usarse si el rango de bifurcación se excede.

PC + 1 ---> PC

(PC) ---> TEMP

PC + 1 ---> PC

EA = PC + TEMP si se da la bifurcación

En el siguiente ejemplo, la rutina usa el registro índice como un contador para ejecutar la subrutina trabajando 50 ciclos.

La bifurcación condicional BNE examina el bit Z el cual se coloca si el resultado de la instrucción DECX limpia el registro X. La línea de código mostrada en la figura 1.8.17 contiene una instrucción para bifurcar a REPEAT, si el código de condición registra que el bit Z no se ha colocado por el paso previo del programa (DECX). Note

en la figura que el bit Z controla cual número se añade al contenido del PC. Si se da la bifurcación, el byte de offset relativo ($FA) se añade; sin embargo, se abandona la EA en PC+2. Note que en este caso el byte del offset relativo indica un retroceso dado que el bit más significativo es uno.

```
04A1 AE   50              LDX #50
04A3 CD   04C0    REPEAT  JSR  WORK
04A6 5A                   DECX
04A7 26   FA  04A3        BNE  REPEAT
```

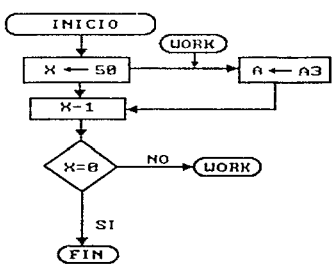


Diagrama de flujo No. 11

Manipulación de bit

La manipulación de bit consta de dos nodos diferentes de direccionamiento: bit

set/clear y bit examinado y bifurcado. El modo bit set/clear permite fijar bits de memoria en forma individual y bits E/S pueden colocarse o limpiarse bajo el control del programa. El modo bit examinado y bifurcado permite que cualquier bit en memoria sea examinado y que se ejecute una bifurcación como resultado. Cada uno de estos modos de direccionamiento se describe a continuación:

Modo de direccionamiento bit set/clear

El direccionamiento directo de byte y direccionamiento de bit están combinados en instrucciones en las cuales se colocan y limpian en memoria individual y en bits de E/S. En las instrucciones bit set y bit clear la localidad de dirección de memoria (contiene el bit que se modificará) se especifica como una dirección directa en la locación siguiente al código de operación. Como en el direccionamiento directo, las 256 localidades de memoria pueden direccionarse. El bit actual puede modificarse, dentro del byte, se especifica dentro del nibble bajo del código de operación. El bit colocado y las instrucciones de limpieza son instrucciones de 2 bytes uno para el código de operación (incluyendo el número de bit) y otro para la dirección del byte el cual contiene el bit de interés.

Precaución

En algunos dispositivos de la familia M6805 HMOS los registros de dirección de datos son registros de sólo escritura y pueden leerse como $FF. Por lo tanto, las instrucciones bit set/clear (o instrucciones lectura-modificación-escritura) no pueden usarse para manipular el registro de dirección del dato.

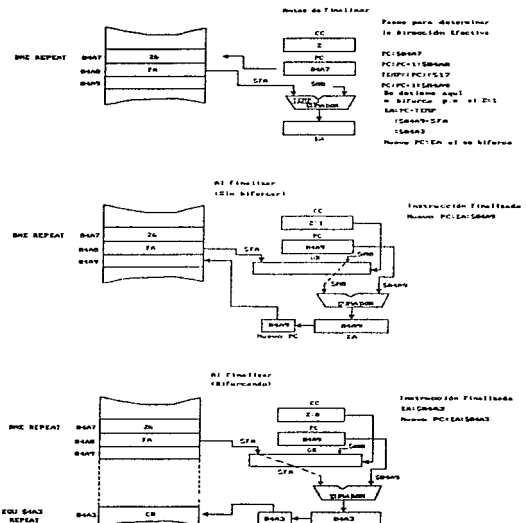


Figura 1.8.17 Ejemplo del modo de direccionamiento relativo

PC + 1 ---> PC
EA = (PC) + $0000
PC + 1 ---> PC

El siguiente ejemplo compara el bit de manipulación verdadero de la familia M6805 HMOS/M146805 CMOS mediante el método convencional de manipulación de bit. Este ejemplo usa la instrucción de manipulación de bit para apagar un LED usando el bit 2 del puerto B y 3 instrucciones convencionales para encender el LED. El ejemplo registra el bit requerido para el control de tiempo del registro de control (TCR bit 7) para determinar cuando debe encenderse el led.

```
0001        PORTB   EQU  $01   Define la dirección del puerto B
0009        TIMER   EQU  $09   Define la dirección del TCR
```

INSTRUCCIONES DE MANIPULACIÓN DE BITS

```
058F  15  01                   BCLR  2,PORTB        Apaga el LED
0591  0F  09     FC    REPT   BRCLR  7,TIMER,REPT
                               Verifica el estado del timer, repite si no está
                               fuera de tiempo
0594  14  01                   BSET  2,PORTB  Enciende el LED si el
                                              timer está fuera de tiempo
```

INSTRUCCIONES CONVENCIONALES

```
AGAIN   LDA  TIMER    Retorna el estado del timer
        BIT  #$80     Enmascara por fuera el bit
                      apropiado
        BNE  AGAIN    Verifica-y enciende si el timer
                      está fuera de tiempo
        LDA  PORTB    Retorna el dato del puerto B
        AND  #$FB     Limpia el bit apropiado
        STA  PORTB    Salva las modificaciones del dato
                      encendiendo el LED
        BRA  REPT
```

La figura 1.8.18 muestra un ejemplo del modo de direccionamiento bit set/clear. En este ejemplo, el ensamblado ejemplifica acerca de los contenidos de una instrucción para limpiar el bit 2 del puerto B (PRTB en este caso es igual al contenido de la localidad de memoria $00), la cual es el resultado de sumar el byte siguiente al código de operación a $0000.

Modo de direccionamiento bit examinado y bifurcado

Este modo es una combinación de direccionamientos directo, relativo y bit set/clear. El byte del dato a examinar se localiza mediante un direccionamiento directo en la localidad siguiente al código de operación. El bit actual a verificar, dentro del

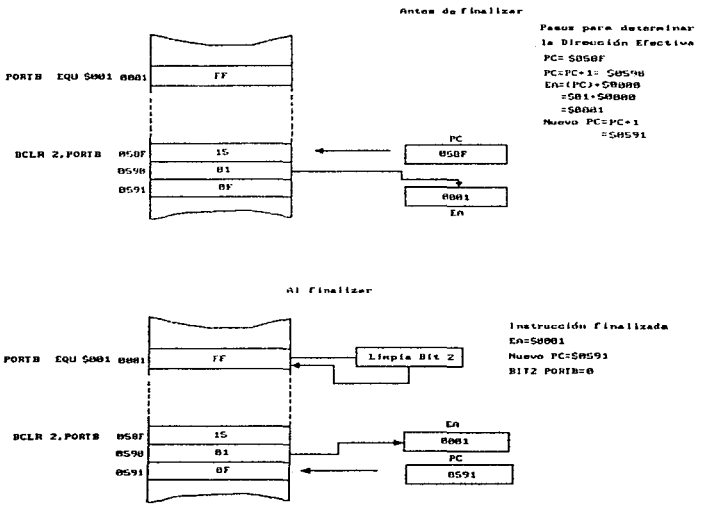


Figura 1.8.18 Ejemplo del modo de direccionamiento "Bit Set/Clear"

byte se especifica dentro del orden más bajo del nibble del código de operación.

El direccionamiento relativo para bifurcar está en el byte siguiente a la dirección directa (segundo byte siguiente al código de operación). Así, las instrucciones de bit examinado y bifurcado son instrucciones de 3 bytes (byte de código de operación, byte directo y byte relativo). Un bit examinado y bifurcado tiene un rango de direccionamiento relativo de PC-125<=R<=PC+130 desde el comienzo de la instrucción.

La rutina de manipulación de bit se muestra en párrafos previos usando un bit de prueba e instrucciones de bifurcación para recibir el timer, por ejemplo REPT, CRCL 7, TIMER, REPT. Esta instrucción causa que el bit 7 del timer se verifique mientras se limpia al tiempo que baja a través del encendido del led.

La figura 1.8.19 ilustra este ciclo mostrando ambos estados , el bifurcado y el no bifurcado. Note que si el bit 7 del timer se limpia $FD se añade a $0594 y este signo del bit es negativo.

Cuando el timer está fuera de tiempo, el bit 7 del timer se coloca en el bit C y el programa desciende a través de $0594. Note en el mismo ejemplo que el bit de prueba y las instrucciones de bifurcación convencionales requieren 3 instrucciones separadas para ejecutar la misma función.

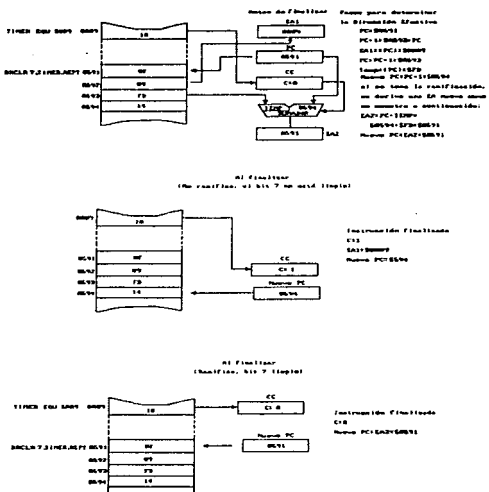


Figura 1.8.19 Ejemplo del modo de direccionamiento "Bit Test and Branch"

Capítulo 2

Planteamiento del Problema

Capítulo 2
Planteamiento del Problema

2.1 Diferencias entre los microcontroladores 68705P3, R3 y U3

Las diferencias principales entre los MCU's las podemos clasificar por:

- Capacidad
- Direccionamiento
  - Mapa de memoria
  - Página Cero

2.1.1 Capacidad

Las diferencias en cuanto a la capacidad de almacenamiento de los MCU's, dan la flexibilidad en su variedad de usos y aplicaciones (los diferentes tamaños de EPROM, cantidad de líneas de E/S, tamaño de memoria bootstrap), permiten libertad de elección de cualquiera de los MCU's ya que podríamos clasificarlos en cuanto a su capacidad en básicos (MCU 68705P3), con los elementos indispensables para la ejecución de un programa, capacidad media (MCU 68705U3) para realizar aplicaciones grandes y de gran capacidad (MCU 68705R3) para elaborar aplicaciones

además de grandes complejas dado que es el que incluye el Convertidor Analógico Digital[1].

Las diferencias se muestran en la tabla 2.1.1.

Tabla 2.1.1 Diferencias de Capacidad de los MCU's

Descripción	MCU 68705P3	MCU 68705R3	MCU 68705U3	Unidades
EPROM	1804x8	3776x8	3776x8	bytes
RAM	112x8	112x8	112x8	bytes
Lineas E/S	20	24	24	pines
ROM bootstrap	115x8	191x8	191x8	bytes
Puerto A	8	8	8	pines/bits
Puerto B	8	8	8	pines/bits
Puerto C	4	8	8	pines/bits
Puerto D	---	8	8	pines/bits
Pines	28	40	40	pines
Convertidor A/D	---	8	---	bits

2.1.2 Direccionamiento

Las diferencias en el direccionamiento están establecidas en cuanto a sus direcciones en el mapa de memoria y en la página cero.

[1]Para mayor información sobre el Convertidor analógico digital, consultar el apéndice A en la sección del MCU 68705R3.

2.1.2.1 Mapa de memoria[2]

Las diferencias se muestran en la tabla 2.1.2.

Tabla 2.1.2 Diferencias de direccionamiento en el mapa de memoria de los MCU's

Descripción	MCU 68705P3		MCU 68705R3		MCU 68705U3	
	Dec	Hex	Dec	Hex	Dec	Hex
Página Cero	000-255	$000-$0FF	000-255	$000-$0FF	000-255	$000-$0FF
EPROM usuario	256-1923	$100-$783	256-3895	$100-$F37	256-3895	$100-$F37
Registro de Opción Enmascarable	1924	$784	3896	$F38	3896	$F38
No usado			3897-3967	$F39-$F7F	3897-3967	$F39-$F7F
ROM Bootstrap	1925-2039	$785-$7F7	3968-4087	$F80-$FF7	3968-4087	$F80-$FF7
Interrupción Timer	2040-2041	$7F8-$7F9	4088-4089	$FF8-$FF9	4088-4089	$FF8-$FF9
Interrupción Externa	2042-2043	$7FA-$7FB	4090-4091	$FFA-$FFB	4090-4091	$FFA-$FFB
SWI	2044-2045	$7FC-$7FD	4092-4093	$FFC-$FFD	4092-4093	$FFC-$FFD
Reset	2046-2047	$7FE-$7FF	4094-4095	$FFE-$FFF	4094-4095	$FFE-$FFF

[2]Para mayor información sobre mapa de memoria de los MCU's 68705P3, U3 y R3 consultar el apéndice A.

2.1.2.2 Página Cero

Las diferencias se muestran en la tabla 2.1.3.

Tabla 2.1.3 Diferencias de direccionamiento en la página cero de los MCU's

Descripción	MCU 68705P3		MCU 68705R3		MCU 68705U3	
	Dec	Hex	Dec	Hex	Dec	Hex
Puerto A	0	$000	0	$000	0	$000
Puerto B	1	$001	1	$001	1	$001
Puerto C	2	$002	2	$002	2	$002
Puerto D	3	No usado	3	$003	3	$003
Puerto A DDR	4	$004	4	$004	4	$004
Puerto B DDR	5	$005	5	$005	5	$005
Puerto C DDR	6	$006	6	$006	6	$006
No usado	7	$007	7	$007	7	$007
Registro de Datos del Timer	8	$008	8	$008	8	$008
Registro Control Timer	9	$009	9	$009	9	$009
Registro Misceláneo	10	No usado	10	$00A	10	$00A
Registro Control Programa	11	$00B	11	$00B	11	$00B
No usado	12	No usado	12	$00C	12	No usado
No usado	13	No usado	13	$00D	13	No usado
Registro de Control del A/D	14	No usado	14	$00E	14	No usado
Registro A/D	15	No usado	15	$00F	15	No usado
RAM (112 bytes) Stack (máx 31 bytes)	16-127	$010-$07F	16-127	$010-$07F	16-127	$010-$07F

2.2 Requerimientos para el sistema de software

- Microsoft Windows 3.x
- Visual C++
- Sistema Operativo 6.0 o mayor

2.3 Visual C++

2.3.1 Características de Visual C++

Microsoft Visual C++ cuenta con una **interfaz de dispositivos gráficos de Windows (GDI)**, con lo cual ya no tiene que proporcionar programas controladores para cada tarjeta de video y para cada modelo de impresora, ya que el GDI es una capa de abstracción que proporciona los controladores para la visualización y para la impresora, de tal manera que el programa no necesita conocer el tipo de tarjeta de video ni la impresora que está conectada al sistema, es decir, en lugar de acceder al equipo, el programa llama a unas funciones de la GDI, las cuales acceden a una estructura llamada contexto de dispositivo. Windows establece una correspondencia entre la estructura contexto de dispositivo y un dispositivo físico y emite las instrucciones de E/S apropiadas. La GDI es casi tan rápida como el acceso directo al video y permite además que distintas aplicaciones escritas para Windows puedan compartir la pantalla.

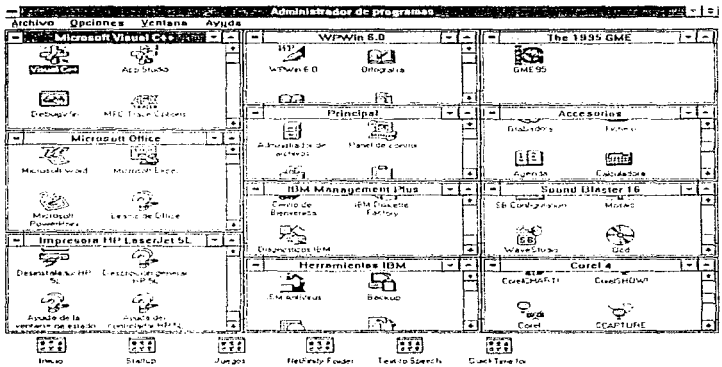


Figura 2.3.1 Presentación de Visual C++ en Windows

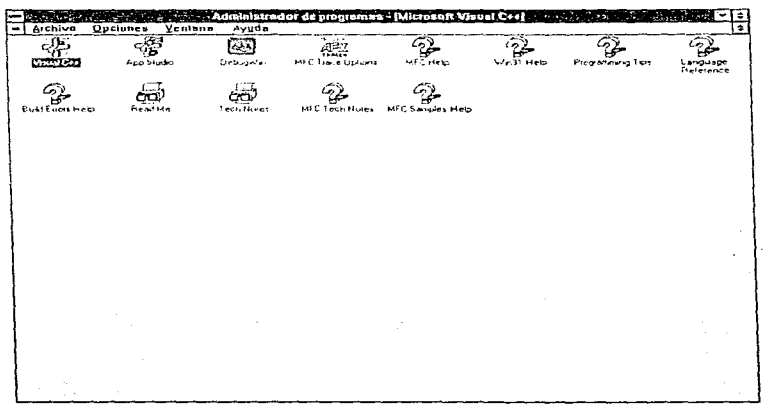


Figura 2.3.2 Componentes de Visual C++

Visual C++ (figura 2.3.1 y figura 2.3.2) tienen una **programación basada en recursos**, para hacer una programación controlada por datos bajo MS-DOS, los datos tienen que ser codificados como constantes inicializadas, o bien se deben proporcionar archivos de datos separados para que el programa los pueda leer. Cuando se programa para Windows, los datos se almacenan en un archivo de recursos usando diversos formatos. Windows se encarga de refundir el archivo de recursos en un programa enlazado, por medio de un proceso denominado "ligadura". Los archivos de recursos pueden incluir mapas de bits, iconos, definiciones de menús, disposiciones de cuadros de diálogo, y cadenas. Incluso pueden contener formatos de recursos personalizados que hayan sido definidos.

Se usa un editor de textos para editar el programa, pero generalmente se emplean **herramientas de tipo "what you see is what you get"**- lo que se ve es lo que se obtiene (wysiwyg) para editar recursos.

Visual C++ emplea una **gestión de memoria**; en los viejos tiempos, el límite de 640 kilobytes de la memoria convencional del MS-DOS restringía el tamaño de los programas. Se podían usar varias técnicas de gestión de recubrimientos y administradores de memoria extendida/expandida para permitir programas más grandes, pero todos tenían sus inconvenientes. Una computadora basada en el 80386SX (o superior) usualmente tiene 4 megabytes o más de memoria, y su CPU tiene un equipo incorporado de gestión de memoria. Windows, conjuntamente con el compilador Visual C++, ofrecen características adicionales de gestión de memoria. El resultado final es que la memoria usualmente ha dejado de ser un problema.

Las **bibliotecas de enlace dinámico (DLL)** son otra de sus ventajas. En el

entorno MS-DOS, todos los módulos objeto de un programa quedaban enlazados estáticamente durante el proceso de construcción. Windows permite un enlace dinámico, lo cual significa que unas bibliotecas especialmente construidas pueden ser cargadas y enlazadas en tiempo de ejecución. Múltiples aplicaciones pueden compartir Bibliotecas de Enlace Dinámico, con lo cual **se ahorra memoria y espacio en disco. El enlace dinámico incrementa la modularidad de un programa porque se pueden compilar y comprobar las DLL en forma separada.**

Los diseñadores originalmente crearon las DLL para ser utilizadas con el lenguaje C, y el C++ ha añadido algunas complicaciones. Después de un considerable esfuerzo, los desarrolladores de la biblioteca de clases(Microsoft Foundation Class Library) lograron combinar todas las clases del marco de la aplicación en una sola DLL. De esta manera, **en una aplicación se pueden enlazar las clases del marco de la aplicación de forma estática o dinámica. Además, se pueden crear DLL propias a partir de las clases de la biblioteca Microsoft Foundation Class Library.**

OLE y TrueType

El sistema operativo Windows ha estado evolucionando a lo largo de los años. Dos nuevas adiciones son la Vinculación e Incrustación de Objetos (OLE) y las fuentes TrueType. Por medio de OLE se pueden mejorar los programas combinando objetos creados en distintas aplicaciones. Por ejemplo, se pueden incluir gráficos, sonidos, dibujos, y otros, en un solo documento. Cuando el usuario activa un objeto OLE, se ejecuta el programa Windows que creó dicho objeto, con lo cual se permite al usuario

poderlo manipular. La programación OLE con las herramientas del SDK de Windows era extremadamente difícil, pero la biblioteca de clases "Microsoft Foundation Class Library" simplifica enormemente la tarea.

Las fuentes TrueType mejoran radicalmente el aspecto de los textos visualizados por Windows tanto en la pantalla como en la impresora. Puesto que estas fuentes se pueden ajustar a cualquier tamaño y porque trabajan prácticamente con cualquier tipo de impresora, el programador queda relevado de la carga de tener que equiparar una fuente que aparece en la pantalla con una de la impresora.

2.3.2 Windows NT

Cuando Microsoft lanzó a la venta Visual C++, Windows NT estaba en las últimas fases de la comprobación beta. Este nuevo sistema operativo tiene un avanzado sistema de archivos con características de seguridad, subprocesos múltiples, una multitarea con una verdadera conmutación forzada de tareas, un acceso a redes mejorado, y portabilidad a un grupo seleccionado de computadoras de tipo RISC. Windows NT puede ejecutar tanto las aplicaciones para Windows de 16 bits existentes como las nuevas aplicaciones para Windows de 32 bits, de elevadas características.

Debido a la necesidad de contar con parámetros de 32 bits, la mayoría de los prototipos de las funciones Win32 son diferentes a sus equivalentes de 16 bits. Además, muchas funciones son nuevas, especialmente en el área de E/S del disco. Las aplicaciones para Windows de 32 bits acceden a los archivos por medio de la API Win 32 en vez de hacerlo mediante la API de MS-DOS.

La SDK Win32 tiene una extensión llamada Win32s, que permite la creación de aplicaciones de 32 bits que se pueden ejecutar tanto bajo Windows 3.1 como bajo Windows NT. Estas aplicaciones aún no pueden sacar partido de algunas características NT avanzadas y por tanto quedan restringidas a un subconjunto de la API Win32

Las aplicaciones existentes en lenguaje C para Windows de 16 bits requerirán una extensa conversión para poder convertirse en verdaderas aplicaciones de 32 bits. Por otra parte, una aplicación hecha con las clases de la Microsoft Foundation Class Library fue diseñada teniendo en mente la API Win32.

La edición Microsoft Visual C++ de 32 bits incluye un código generador de wizards y un sobresaliente editor. Visual C++ 1.5 no es la mejor herramienta para el desarrollo del DOS, y la edición de 32 bits tampoco la soporta.

2.3.3 Visual Workbench

Visual Workbench es un entorno de desarrollo interactivo residente en Windows que es un descendiente directo de QuickC para Windows de Microsoft. A aquellos que estén acostumbrados a ejecutar un compilador desde la línea de órdenes.

Visual Workbench trabaja con proyectos. Un proyecto es una colección de archivos fuente relacionados entre sí, que son compilados, enlazados y ligados para constituir un programa funcional para Windows. Los archivos fuente de proyectos

generalmente se almacenan en un subdirectorio aparte. Un proyecto depende de muchos archivos que están fuera del subdirectorio del proyecto, tales como los archivos de cabecera y los archivos de bibliotecas.

Un archivo make expresa todas las interrelaciones que hay entre los archivos fuente. Un archivo de código fuente necesita archivos de cabecera específicos; un archivo ejecutable necesita ciertos módulos objeto, bibliotecas, y otros. El programa make lee el archivo make y entonces invoca al compilador, ensamblador, enlazador, y al compilador de recursos para producir el resultado final que generalmente es un archivo ejecutable. El programa make utiliza unas "reglas de inferencia" incorporadas que le instruyen, por ejemplo, qué debe invocar el compilador para generar un archivo OBJ a partir de un archivo CPP que se especifica.

En un entorno de línea de órdenes es necesario codificar el archivo make de forma manual. En cambio, Visual Workbench genera de forma automática el archivo make, que se conoce como "archivo de proyecto". En la mayoría de los casos, se quiere que el proyecto incluya todos los archivos fuente que hay en el subdirectorio, pero se pueden excluir archivos del subdirectorio del proyecto y usar archivos de otros subdirectorios. Una vez que se crea el proyecto, se pueden editar los archivos de código objeto en ventanas hijas individuales. Visual Workbench "recuerda" cuáles eran los archivos de código objeto con los cuales estaba trabajando y mantiene una lista de los proyectos más recientemente usados. Como una parte del proyecto, se pueden guardar las configuraciones de los conmutadores del compilador y del enlazador, que se especifican por medio de cuadros de diálogo. Para generar el programa ejecutable, basta con escoger la orden Build (Construir) del menú Project

Workbench.

El usuario le indica al director de proyecto donde se desean las filas fuentes y después la aplicación en conjunto con este construirá un proyecto interno llamado mediante un archivo MAKE que rastrea cada archivo por dependencias. Las filas de proyectos MAKE están en formato NMAKE y pueden utilizarse para una línea de comandos en caso de que se escoja.

Visual Workbench contiene un útil editor de textos que sigue las normas de interfaz de Windows y usa color para resaltar la sintaxis C++. Lamentablemente no se puede personalizar completamente este editor, ni instalar un editor del usuario. Si el programador decide utilizar su propio editor, se perderá la integración uniforme que proporciona el entorno de desarrollo integrado. Por ejemplo, cuando se crea un proyecto Visual Workbench ilumina las líneas que contienen errores en los archivos de código fuente y permite que se fijen puntos de ruptura.

Por lo que no se pueden agregar archivos C/C++ a los archivos internos de proyecto. Si un programa usa módulos escritos en otros lenguajes, tal como ensamblador, se deben usar archivos MAKE externos o ensamblarlos por separado y entonces añadir los archivos OBJ al proyecto.

Los archivos MAKE que el mismo usuario haya escrito para cualquier Visual Workbench, deben mantenerse en forma manual.

Visual Workbench combina: **editor de código fuente, editor de recursos (App Studio), menú de código fuente, depurador, una herramienta para generar los programas básicos de trabajo en grupo de Windows (App Wizard), una herramienta para automatizar tareas de programación comunes de Windows**

(Class Wizard), y una extensa ayuda en línea.

Visual C++ 1.5 incluye wizards para crear aplicaciones OLE 2.0 y ODBC (Open Database Connectivity). Los wizards generan un código C++ que se utiliza en Microsoft Foundation Classes (MFC).

2.3.4 El editor de recursos App Studio

El editor de recursos, App Studio, es una de las mejores partes de Visual Workbench, ya que **maneja virtualmente cualquier recurso y su diseño visual lo hace muy intuitivo de usar.** La creación del menú está bien diseñada, y **la característica de multinivel hace muy fácil la corrección de errores.**

La SDK primitiva para Windows incluía herramientas separadas para editar cuadros de diálogo, mapas de bits y fuentes. En cambio, en Visual C++ se usa App Studio para editar la mayoría de los recursos. App Studio incluye tanto un editor de menús wysiwyg como un poderoso editor de cuadros de diálogo que es muy superior al antiguo programa DIALOG del SDK de Windows. Se puede utilizar App Studio como editor de recursos para hacer una programación estilo SDK para Windows, pero cuando se usa App Studio para hacer una programación estilo Biblioteca de Clases, entonces se puede insertar en forma interactiva, controles de Microsoft Visual Basic en los cuadros de diálogo, para vincularlos posteriormente al código C++.

El formato original de App Studio es el formato de archivo de recursos ASCII (RC) para Windows, y cada proyecto usualmente tiene un archivo RC con sentencia #include para incorporar recursos de otros subdirectorios. No es recomendable editar

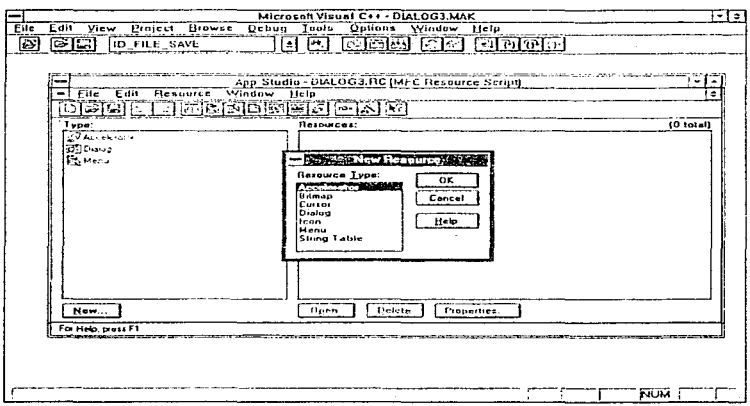


Figura 2.3.3 AppStudio

el archivo RC fuera de App Studio. App Studio también puede procesar archivos EXE y DLL, de manera que se puede utilizar el portapapeles para "pedir prestados" recursos, tales como iconos y mapas de bits, a otras aplicaciones para Windows.

App Studio se compara favorablemente con las mejores aplicaciones que han sido escritas para Windows, de modo que es muy significativo que App Studio haya sido escrito utilizando herramientas de Visual C++ y la Biblioteca de Clases. App Studio puede editar incluso sus propios recursos.

2.3.5 El compilador C/C++

El compilador de Visual C++ (figura 2.3.4) puede procesar tanto código fuente en C como también código fuente en C++. El compilador determina cuál es el lenguaje en cuestión examinando la extensión de nombre de archivo de código fuente. Una extensión C indica código fuente C, y una extensión en CPP o CXX indica código fuente C++. El compilador cumple con la versión ANSI 2.1 y tiene extensiones adicionales de Microsoft. La opción "Use Microsoft Foundation Classes" (del cuadro de diálogo de Project Options) de Visual Workbench determina si el compilador utilizará los archivos de inclusión de la biblioteca Microsoft Foundation Class Library.

El compilador C++ en ambas versiones es **rápido**, más rápido que los compiladores predecesores de Microsoft. Ambos **proveen programas pequeños y rápidos de ejecutar y pueden optimizar códigos para procesador INTEL** arriba de 486. La edición de 32 bits agrega una optimización para los procesadores PENTIUM.

Desafortunadamente ambos productos tienen un retraso en la sintaxis de C++,

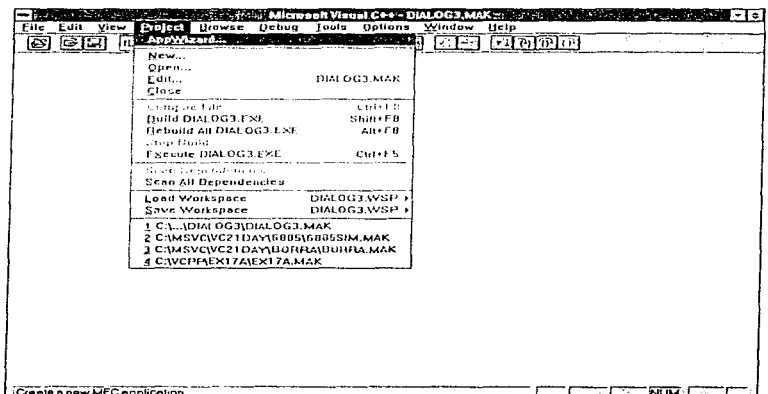


Figura 2.3.4 El compilador de Visual C++

ni las plantillas (templates) ni la sintaxis de excepción están soportadas, y Microsoft aún no ha agregado un soporte o ha manejado una excepción, pero ambas están planeadas para la siguiente versión.

2.3.6 El enlazador

Para generar un código EXE, el enlazador de Visual C++ procesa los archivos OBJ que produce el compilador. Si se especifica la opción "Use Microsoft Foundation Classes" de Visual Workbench, el enlazador utilizará el archivo de la biblioteca Microsoft Foundation Class Library para obtener el modelo apropiado de memoria.

2.3.7 El compilador de recursos

El compilador de recursos de Visual C++ opera ya sea en modo compilar o en modo ligar. En modo compilar, un archivo de recursos ASCII (RC) de App Studio se compila en un archivo binario RES. En modo ligar, el archivo RES se fusiona con un archivo ejecutable (EXE). Si posteriormente se actualiza un archivo RES, éste se puede volver a ligar a su archivo EXE sin tener que volverlo a enlazar.

2.3.8 El depurador

Si un programa funciona la primera vez, el programador no necesita un depurador. Los demás necesitamos uno de vez en cuando. El depurador de Visual C++ es el primer entorno de depuración para C++ que funciona bajo Windows. El depurador trabaja estrechamente con Visual Workbench para asegurar que los puntos

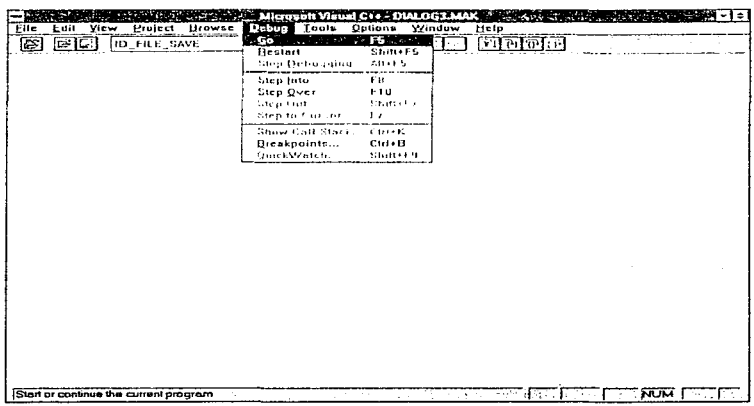


Figura 2.3.5 El depurador de Visual C++

de depuración quedan guardados en disco. Los botones de la Barra de herramientas (figura 2.3.5) sirven para conmutar los puntos de depuración y controlan la ejecución paso a paso. Es importante mencionar que la ventana llamada Locals puede ampliar un puntero a un objeto para mostrar todos los miembros de datos de la clase derivada y de las clases base. Para depurar un programa, se debe construir el programa configurando las opciones del compilador y del enlazador de manera que generen información para la depuración.

Con la Edición Profesional de Visual C++ se obtiene el depurador modo carácter Code View, junto con el depurador que trabaja bajo Windows. Code View para Windows sirve para depurar código-p, y tienen otras varias características útiles.

2.3.9 AppWizard

AppWizard (figura 2.3.6) es un generador de código que crea un esqueleto funcional de una aplicación para Windows, con características, nombres de clases y nombres de archivos de código fuente que se especifican por medio de cuadros de diálogo. No se debe confundir AppWizard con los generadores de código convencionales tales como CASE:W de Caseworks y Windows MAKER de Blue Sky. El código de AppWizard es un código mínimo; la funcionalidad está dentro de las clases base de Marco de la aplicación. Su propósito es ayudar al programador a iniciar rápidamente una nueva aplicación.

2.3.10 ClassWizard

ClassWizard (figura 2.3.7) es un programa (implementado como una DLL) que

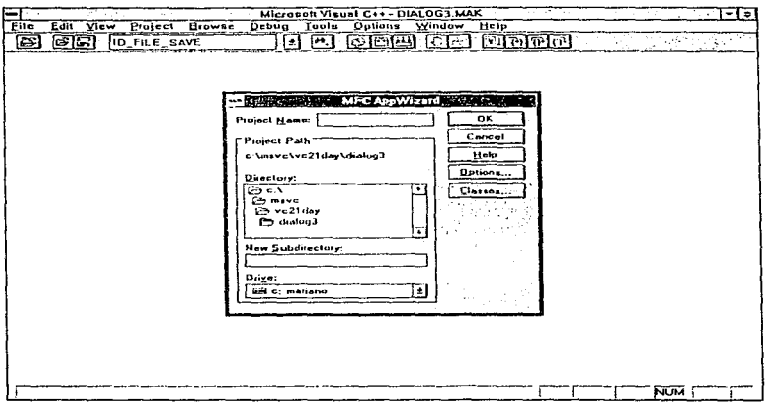


Figura 2.3.6 AppWizard

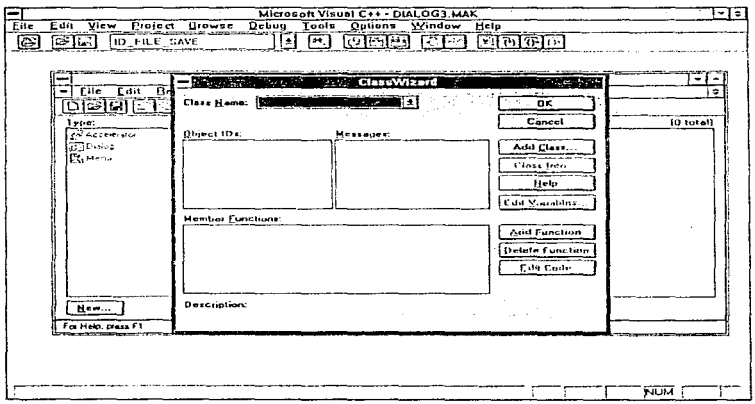


Figura 2.3.7 ClassWizard

opera tanto en el interior de Visual Workbench como también en el interior de AppStudio. ClassWizard quita el trabajo pesado al mantenimiento del código de las clases de Visual C++. ¿Acaso se necesita una nueva clase o una nueva función para manejar un mensaje de Windows? ClassWizard se encarga de escribir los prototipos, el cuerpo de las funciones y el código para vincular los mensajes con el Marco de la aplicación. ClassWizard puede actualizar el código de las clases que escribe el programador, así evita los problemas de mantenimiento que son comunes a todos los generadores de código usuales.

Las herramientas de Visual C++ reducen el trabajo pesado de la programación. AppStudio, AppWizard y ClassWizard reducen significativamente el tiempo que se necesita para escribir código que es específico a la aplicación. Por ejemplo, AppStudio crea un archivo de cabecera que contiene valores asignados a las constantes de tipo #define. AppWizard genera el esqueleto del código para toda la aplicación y ClassWizard genera los prototipos y los cuerpos de las funciones para los controladores de mensaje.

2.3.11 El Source Browser

Cuando se escribe una aplicación partiendo desde cero, probablemente se obtiene un buen cuadro mental de los archivos de código fuente, de las clases y de las funciones miembro. En cambio, cuando uno se hace cargo de una aplicación de un tercero, entonces necesitará alguna ayuda. El Source Browser (analizador) de Visual C++ (más brevemente el Browser) permite examinar (y editar) una aplicación desde el punto de vista de un archivo (figura 2.3.8). Se asemeja un poco a la herramienta

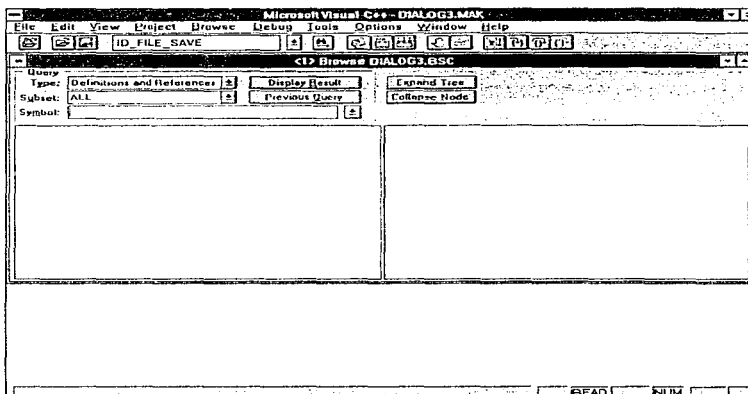


Figura 2.3.8 Source Browser

"inspector" que se encuentra disponible en otras bibliotecas orientadas a objetos como la de Smalltalk. El Browser tiene los siguientes modos de visualización:

- Definiciones y referencias: Se escoge cualquiera función, variable, tipo, macro o clase y luego se ve donde está definida y utilizada en el programa.
- Gráfico de llamada/Gráfico de llamadora: Dada una función seleccionada, se obtiene una representación gráfica de las funciones que ella llama, o de las funciones que la invocan.
- Gráfico de clase derivada/Gráfico de clase base: Estos son diagramas de jerarquía gráfica de clases. Dada una clase seleccionada, se ven las clases derivadas o las clases base. Se puede controlar la extensión de la jerarquía por medio del ratón.

NOTA:

Si se cambia el orden de las líneas de cualquier archivo de código fuente, entonces se debe reconstruir la base de datos del analizador.

2.3.12 Ayuda en línea

Todo el contenido de los manuales de referencia del SDK de Windows y los manuales de referencia de la Biblioteca de Clases están incluidos en la ayuda en línea (figura 2.3.9) de visual de Visual C++. También existe ayuda para App Studio, AppWizard y ClassWizard. No hay que subestimar el valor de Ayuda. Muchos programadores de Microsoft la usan casi exclusivamente. Si se desea ayuda con

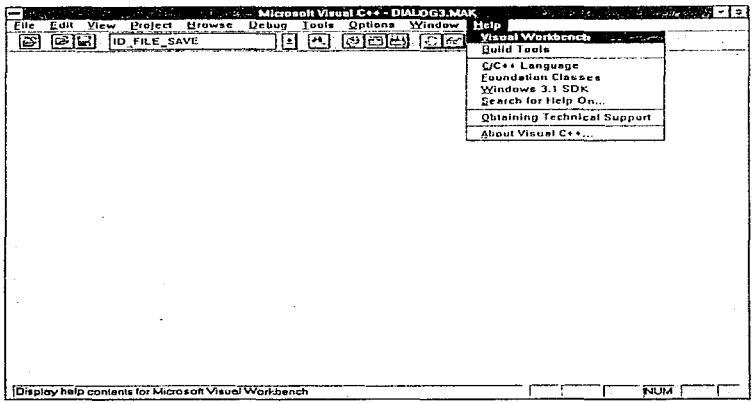


Figura 2.3.9 Ayuda en Línea

respecto a una función, simplemente se pulsa sobre (o bien se desplaza el cursos hacia) la función en el editor de Visual Workbench y se pulsa F1, con lo cual aparecerá una ventana de ayuda.

La Ayuda de Visual C++ resuelve los conflictos entre los nombres de las funciones de la SDK de Windows y nombres idénticos pertenecientes a las clases de la Microsoft Foundation Class Library. Si se selecciona el nombre de una función que corresponde a funciones miembro de varias clases, se puede escoger la clase en un cuadro de lista. Si se pide ayuda con respecto de una clase, entonces podrá ver una función miembro y una lista de miembros de datos en un orden funcional.

2.3.13 Herramientas de diagnóstico para Windows

La edición Profesional de Visual C++ contiene el mismo juego de herramientas de diagnóstico que las que fueron incluidas en el SDK de Windows, cuando era un producto separado: SPY para observar los mensajes de Windows, HEAP-WALK para examinar la memoria, HC31 para compilar archivos de Ayuda, STRESS para limitar la memoria de forma artificial, y un programa perfilador que da la alarma sobre la presencia de cuellos de botella en el código.

Ambas la Edición Estándar y la Edición Profesional, incluyen la utilidad DBWIN, que muestra una salida de diagnóstico, y el programa NMAKE, que procesa archivos make codificados manualmente. NMAKE se utiliza para construir versiones no estándar de la Microsoft Foundation Class Library.

2.3.14 La Microsoft Foundation Class Library versión 2.0

La Microsoft Foundation Class Library versión 2.0 (que llamaremos la Biblioteca de Clases) define el Marco de la aplicación (figura 2.3.10). Pero, ¿por qué usar el Marco de la aplicación?: si se van a desarrollar aplicaciones para Windows, entonces se debe escoger un entorno de desarrollo. Asumiendo que ya han sido rechazadas las opciones interactivos tales como Microsoft Visual Basic, entonces se debe escoger entre las siguientes opciones:

- El probado y verificado SDK para Windows (Kit de Desarrollo de Software).
- El nuevo Marco de la aplicación de la biblioteca de clases Microsoft Foundation Class Library.
- Otros marcos de trabajo para Windows tales como el Object Windows Library (OWL) de Borland.

El Marco de la aplicación de la Biblioteca de Clases está repleto de características. Las clases de la Biblioteca de Clases versión 1.0 que fueron suministradas en la versión 7.0 del C/C++ de Microsoft, esencialmente constituyeron una interfaz de programación de C++ para Windows. No obstante, se añadieron algunas características significativas:

- Unas clases de propósitos generales (no específicas a Windows), entre las cuales se incluyen:
  - Clases de colecciones de listas, arrays y mapas.
  - Una clase cadena útil y eficiente.
  - Clases relativas al transcurso de tiempo y de fecha.

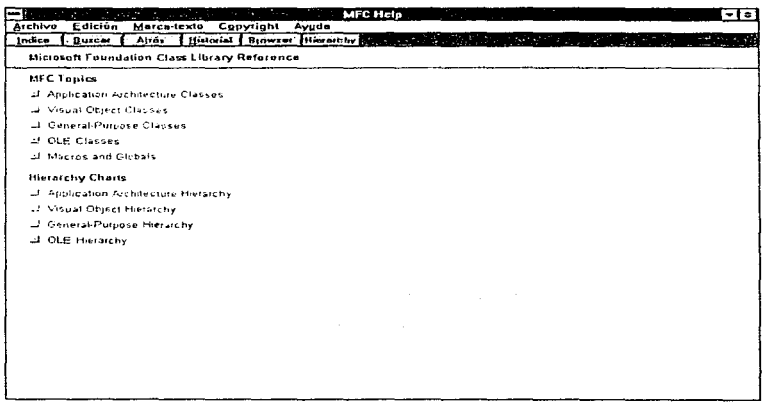


Figura 2.3.10 Microsoft Fundation Class Library v. 2.0

- Clases de acceso a archivos para obtener independencia del sistema operativo.
- Un soporte para el almacenamiento sistemático de objetos hacia y desde un disco.

- Una jerarquía de clases de "objeto raíz común".
- Soporte de aplicación racionalizado para la Interfaz de Documentos Múltiples (MDI).
- Un soporte efectivo OLE (Vinculación e Incrustación de Objetos).

Las clases de la Biblioteca de Clases versión 2.0 siguen adelante desde el punto hasta donde llegaron las clases de la versión 1.0 prestando apoyo a muchas características de interfaz de usuario que se encuentran en las aplicaciones actuales para Windows. Sin contar con la arquitectura del Marco de la aplicación, éste es un resumen de las nuevas características importantes:

- Un soporte amplio a los elementos de menú: Abrir archivo (File Open), Guardar (Save) y Guardar como (Save As) de la lista de archivos más recientemente usada.
- Presentación preliminar de impresión y soporte de impresora.
- Ventajas desplazables y ventanas divisibles.
- Barras de herramientas y barras de estado.
- Acceso a los controles de Microsoft Visual Basic.
- Ayuda sensible al contexto.
- Procesamiento automático de datos introducidos en un cuadro de diálogo.
- Una interfaz OLE fácil de programar.

- Un soporte a las DLL.

2.3.15 El Marco de la aplicación frente a la Biblioteca de Clases

Una de las razones por la cual C++ es un lenguaje tan difundido, se debe a que puede ser "extendido" por medio de bibliotecas de clases. Algunas bibliotecas de clases son entregadas junto con los compiladores de C++, otras son comercializadas por empresas de software independientes, y otras son desarrolladas dentro de la misma empresa. Una biblioteca de clases es un conjunto de clases C++, relacionadas entre sí, que se pueden utilizar en una aplicación. Por ejemplo, una biblioteca de clases de matrices puede encargarse de efectuar las operaciones matemáticas usuales que se efectúan con las matrices, y una biblioteca de clases de comunicaciones puede dar soporte al transporte de datos por un enlace serie. Algunas veces se construyen objetos de las clases que han sido proporcionadas; en otras se deriva de ellas unas clases propias. Todo esto depende del diseño de la biblioteca de clases en particular.

Un Marco de la aplicación es un superconjunto de una biblioteca de clases. En efecto, una biblioteca de clases normal es un conjunto aislado de clases diseñadas para ser incorporadas en cualquier programa, en cambio un Marco de la aplicación define la estructura del programa mismo. Esto suena como una distinción precisa, y lo es. La mayoría de las bibliotecas de clases de desarrollo para Windows, incluyendo en ellas la Microsoft Foundation Class Library versión 1.0, la OWL de Borland y la Microsoft Foundation Class Library versión 2.0 son consideradas como Marcos de aplicación. No obstante, la Microsoft Foundation Class Library versión 2.0 proporciona de forma

significativa muchas más características que las otras.

2.3.16 Lo nuevo de la versión 1.5

Lo más importante dentro de las nuevas características de Visual C++ 1.5 es el soporte para OLE 2.0 y ODBC. El OLE 2.0 SDK y porciones claves de los ODBC SDK están ahora disponibles con el producto.

Sin embargo los MFC 2.5 y Visual Workbench wizards vienen a ser un soporte tanto para OLE 2.0 como para ODBC.

La visión de las compañías acerca de **OLE** es que viene a ser una rutina incorporada dentro de las aplicaciones de Windows, pero las especificaciones de OLE 2.0 son extremadamente complejas y revueltas. El MCF 2.5 ahora está provisto de un buffer entre el programador y los detalles del OLE 2.0

Las tareas son hechas fácilmente mediante las nuevas opciones de OLE sin Class Wizard y App Wizard. La clase Wizard está rediseñada con tabulador de índice para hacer más fácil su uso.

Microsoft agrega el soporte ODBC para el Visual C++ 1.5 para bases de datos externas a las aplicaciones. ODBC está provisto mediante de un API para accesar datos en cualquier formato; todo lo que se necesita es un manejador apropiado. Manejadores para Access, Btrieve, dBase, Excel, FoxPro, Oracle Server, Paradox, SQL Server, y archivos textos se proveen con el producto.

El producto puede correr bajo el Windows NT. También puede correr la versión del comando en línea del compilador para OS/2 y si se cambia el

DPMIDOS_APPI colocando en "habilitado" y el DPMI_MEMORY_LIMIT a 10 MB.

2.3.17 Nueva edición de 32 bits

La edición de 32 bits puede crear aplicaciones WIN32 NT y WIN32. Sólo la plataforma INTEL se soporta actualmente pero Microsoft está trabajando en su versión para MIPS y DEC Alpha.

Así como el Visual C++ 1.5 la edición de 32 bits está construida alrededor del Visual Workbench pero si se prefiere tiene líneas de comandos de herramientas.

La utilería Spy (escudriñar) provista con el Visual C++ 1.5 es usada para enviar mensajes a una ventana, pero se pueden decodificar los parámetros del mensaje manualmente. En la edición de 32 bits el SPY se reemplaza por el SPY++ que adicionalmente muestra el flujo de información, despliega totalmente la decodificación de los parámetros de cada mensaje e información importante mediante ventanas, procesos y ligas.

El perfil de la edición de 32 bits como la que viene en Visual C++ 1.5 da opción a analizar los programas basándose en función del tiempo, función del conteo, función del muestreo, función del alcance, el contador de línea y alcance de línea.

En resumen podemos listar las ventajas y desventajas de Visual C++ de la siguiente manera:

2.3.18 Ventajas de Compilación:

- ✓ Optimización de tamaño y velocidad
- ✓ Optimización 386, 486
- ✓ Encabezados de precompilación
- ✓ Ensamblado en línea
- ✓ Fondo de compilación
- ✓ Soporta plataformas: DOS16, Microsoft Windows 3.1, Microsoft NT
- ✓ Funciones de usuario en línea
- ✓ Tipos anidados
- ✓ Editor
- ✓ Teclas de código de color
- ✓ Documentación en línea

2.3.19 Ventajas del ambiente de desarrollo:

- ✓ Ayuda sensible al contexto
- ✓ Wizards, experts o equivalentes
- ✓ Importación de ejemplos desde la ayuda en línea
- ✓ El usuario puede programar y salvar su ambiente de trabajo
- ✓ Lanzador de programas
- ✓ Herramientas de arrastrar y dejar
- ✓ Menús definidos por el usuario
- ✓ Depurador integrado
- ✓ Versiones de control de atajos

2.3.20 Ventajas de las herramientas de programación:

- ✓ Menú de clases
- ✓ Perfil
- ✓ Programación con Windows o OS/2:
  - Recursos de compilación
  - Ayuda de compilación
  - Recursos de edición
  - Aplicación de clases de trabajo en grupo

2.3.21 Ventajas de Debugging:

- ✓ Debugging target:
  - Windows
  - DOS 16
  - Menús EXE o DDL
  - Escudriñador de mensajes
  - Escudriñador DDE

2.3.22 Ventajas de su miscelánea:

- ✓ Creación de programas de varias capas (multinivel)
- ✓ Rutinas de verificación:

- Apuntadores nulos
- Sobrecarga de la pila

✓ Clases y librerías
- OLE 2.0
- VBX acceso de control

2.3.23 Desventajas

× La curva de aprendizaje es árida ya que se requiere de experiencia trabajando con C++ y con técnicas de programación orientada a objetos.

× Aunque Visual C++ realiza una gestión de memoria consume mucha de esta así que si no se cuenta con el equipo apropiado las tareas se alentarán considerablemente.

× No maneja excepciones

× No tiene teclas remapeables

× No es la mejor herramienta para el desarrollo del DOS

× Desafortunadamente no se pueden agregar archivos C/C++ a los archivos internos de proyecto.

× Visual C++ 1.5 no tiene un depurador que funcione en forma independiente y Microsoft no planea agregarlo.

2.4 Generación de código y optimización

2.4.1 Introducción

El código de nuestros programas se basa en la P.O.O. (Programación Orientada a Objetos), se hizo de este modo dado que Visual C++ está desarrollado totalmente en esta técnica de programación.

De las clases que tiene MFC de Visual C++ se utilizaron:

CString que es la clase que se encarga de definir cadenas de caracteres y que gracias a los operadores que tiene definidos se pueden realizar de manera fácil operaciones comunes entre cadenas como son concatenación (por medio del operador +), asignación (por medio del operador =), obtener un elemento de una cadena (operador []), entre otros. El tamaño de caracteres que acepta esta clase es de 32,767.

CMenu: Mediante esta clase se pueden manejar los menús, lo cual incluye la creación, actualización, seguimiento y eliminación de un recurso de menú asociado con una ventana.

CEdit: Esta clase implementa un control de edición que soporta texto de una línea y de líneas múltiples, también tiene la capacidad de cortar, pegar,

copiar, borrar y limpiar el texto, así como de deshacer los últimos cambios al texto e intercambiar texto con el portapapeles.

Subjerarquías de MFC

Ventanas enmarcadas Tiene a la clase **CFrameWnd** como su raíz. Este grupo de clases modela la ventana de la interfaz básica de un solo documento (SDI) o las ventanas más complejas de interfaz de documento múltiple (MDI).

Cuadros de dialogo Incluye la clase de cuadros de diálogo de propósito general, **CDialog**, así como las clases que soportan los cuadros de diálogo comunes para la selección de archivos, la selección de colores, la selección de tipos de letra, la impresión y la búsqueda y el reemplazo de texto. La clase CDialog modela a los cuadros de diálogo modales y sin modo, y es la raíz de esta subjerarquía.

CDialog Es descendiente de **CWnd**, tiene un constructor de clase y varias funciones miembro, incluida la función Create. Sirven para desplegar información o para dar entrada a datos. Los cuadros de diálogo modales requieren que se cierren antes de continuar con la aplicación, pues están orientados a ejecutar un importante intercambio de datos, de hecho, los cuadros de diálogo modales desactivan a sus

ventanas madre cuando tienen el foco. Los cuadros de diálogo sin modo no necesitan cerrarse para continuar utilizando la aplicación.

De los tipos de datos que utilizamos de Visual C++ fue:

FILE el tipo de dato para manejar archivos y realizar operaciones en ellos.

De las funciones que trae consigo Visual C++ utilizamos[3]:

atoi int atoi(const char **cadena*);
recibe una cadena y la convierte a su valor entero.

delete delete *variable*;
desasigna memoria dinámica de una variable.

itoa char *_itoa(int *valor*, char **cadena*, int *radix*); declarada en stdlib.h
convierte un número entero en una cadena de caracteres terminada en nulo y a continuación almacena el resultado (de hasta 17 bytes) en el argumento *cadena*. El argumento *radix* especifica la base en la que se va a convertir el *valor*. Esta base debe de encontrarse dentro del rango de 2 a 36. Si el argumento radix vale 10 y *valor*

[3] Microsoft Visual C++, Funciones y aplicaciones.

	es negativo, el primer carácter de la cadena resultante será el signo menos (-).
new	new *variable*; asigna memoria dinámica a una variable del tamaño de algún tipo de dato ya existente o creado por el usuario.
strupr	Esta función convierte los caracteres en minúscula de la cadena origen a mayúsculas.
MessageBox	Esta función permite la comunicación con el usuario final de la aplicación de Windows.

2.4.2 Ensamblador

2.4.2.1 Introducción

Debido a que un programa en ensamblador consta de una serie de sentencias de una línea, parece natural tener un ensamblador que lea una sentencia, la traduzca a lenguaje de máquina y escriba el código de máquina generado en un archivo y la porción del listado correspondiente, si la hay, en otro. El proceso se repetiría hasta que todo el programa se haya traducido. Pero en la práctica este método no funciona.

Considérese la situación en la que la primera sentencia sea un salto a E. El ensamblador no puede ensamblar esta instrucción hasta que conozca la dirección de la sentencia E. La sentencia E puede estar cerca del fin del programa, lo que impide que el ensamblador encuentre la dirección sin leerse primero casi todo el programa. Esta dificultad se llama referencia adelantada, porque el símbolo E se ha usado antes de que se haya definido; es decir, se ha hecho referencia a un símbolo cuya definición aparecerá más tarde.

Las referencias adelantadas pueden tratarse de la siguiente manera: La primera es que el ensamblador lea el programa fuente dos veces. Cada lectura del programa fuente se llama una pasada, y todo traductor que lea el programa de entrada dos veces se llama traductor de dos pasadas. En la pasada 1, el ensamblador de dos pasadas colecciona todas las definiciones de símbolo, incluyendo las etiquetas de las sentencias, y las almacena en una tabla. En el momento en que empieza la segunda pasada, ya se conocen los valores de todos los símbolos, ya no existe el problema de las referencias adelantadas y puede leerse cada sentencia, ensamblarla y obtener una salida.

Dividimos el programa en tres secciones principales:

- Editor
- Ensamblador:
    - Primera pasada
    - Segunda pasada

- Validación de datos y detección de errores

2.4.2.2 Editor

Esta aplicación despliega texto que pueda almacenarse y/o recuperarse de un archivo.

El editor realiza las funciones básicas de:

- Crear archivo
- Abrir archivo
- Cerrar archivo
- Salvar archivo

Optimización del editor:

Construcción de ventanas hijas MDI:

Esto permite que se abran ventanas hijas para tareas específicas de archivo, como la edición de texto, manejo de una base de datos o el trabajo en una hoja de cálculo

2.4.2.3 Ensamblador

Las instrucciones se dividieron por tipo como sigue:

Tipo A:	Instrucciones de lectura, modificación y escritura
Tipo B:	Instrucciones de control
Tipo C:	Instrucciones de "bit set/clear"
Tipo D:	Instrucciones "bit test and branch"
Tipo E:	Instrucciones de registro y memoria
Tipo F:	Instrucciones de ramificación

También se dividieron los tipos de direccionamiento:

Tipo 1:	Direccionamiento Inherente
Tipo 2:	Direccionamiento Inmediato
Tipo 3:	Direccionamiento Directo
Tipo 4:	Direccionamiento Extendido
Tipo 5:	Direccionamiento Indexado sin offset
Tipo 6:	Direccionamiento Indexado con un byte de offset
Tipo 7:	Direccionamiento Indexado con dos bytes de offset
Tipo 8:	Direccionamiento de instrucciones del tipo "Bit set/clear"
Tipo 9:	Direccionamiento de instrucciones del tipo "Bit test and branch"
Tipo 10	Direccionamiento relativo

Se definieron dos clases principales una para almacenar las instrucciones del programa fuente del usuario y otra que contiene el conjunto de instrucciones permitidas.

En la primera pasada se toma cada línea del programa fuente, separando la etiqueta, el nmónico, el operando y el comentario si es que lo tiene.

Una vez que se han identificado las partes de cada línea del programa fuente, se realiza la validación de datos y detección de errores

En general el proceso de ensamblado que realiza el programa puede describirse de la siguiente manera:

1.- Carga archivo
2.- Separa cada línea del programa fuente
3.- Identifica, para cada línea del programa fuente, el nmónico, la etiqueta, el operando, y el comentario respectivamente y en caso de que exista.
4.- Verifica si el nmónico pertenece al conjunto de instrucciones permitidas
5.- Identifica el tipo de instrucción
6.- Verifica el operando
7.- Incrementa el PC
8.- Identifica el tipo de direccionamiento
9.- Si existe comentario lo identifica

Los puntos 2 al 9 representan el proceso denominado "primera pasada".

Durante la segunda pasada se detecta si el tipo de direccionamiento de la instrucción es permitido para dicha instrucción o no.

Ensambla las referencias adelantadas, ya que como mencionamos anteriormente, en este punto el ensamblador ya conoce todas las direcciones.

Posteriormente se genera el código ensamblado y se crean los archivos de trabajo ".prn" y ".mik".

Optimización:

Para optimizar la validación del nmónico utilizamos búsqueda binaria, la cual aprovecha el orden del arreglo,. El método busca un valor concordante usando el método de intervalos decrecientes. El intervalo de búsqueda inicial incluye todos los elementos del arreglo. El método compara el elemento medio del intervalo con el valor buscado. Si los dos concuerdan, la búsqueda termina, en caso contrario el método determina que subintervalo usar como siguiente intervalo de búsqueda.

De esta manera se garantiza que cada intervalo de búsqueda es de la mitad del tamaño del anterior.

2.4.3 Simulador

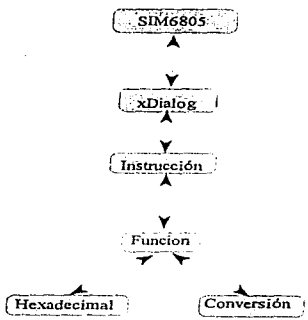


Figura 2.4.1 Diagrama modular del simulador

El simulador está dividido en las siguientes partes:

-Simulador
-CxDialog
-Instrucción
-Función

-Hexadecimal
-Conversión

Simulador

De esta parte se encarga el programa SIM6805.CPP y es el que se encarga de crear la aplicación y de la creación del objeto cuya clase es CxDialog.

xDialog

Los archivos que se encargan de esta parte son XDIALOG.H, XDIALOG.CPP, FUNCION.CPP.

Esta parte es la encargada de crear la clase CxDialog y se encarga de crear la ventana principal y de manejar los mensajes de menú, de ventana de botones, cuadros de edición y todos los elementos que componen la ventana de simulación.

Además que es la que manda crear un objeto de tipo CxResDlg que es la ventana de dialogo modal con la cual se especifica el archivo a simular, su directorio y el tipo de MCU a simular.

Es el encargado de abrir el archivo texto de extensión ***prn*** que fue indicado en la ventana de dialogo de "abrir archivo" e iniciar los datos del simulador y la tabla de

instrucciones a ejecutar, el número de instrucciones máxima a ejecutarse es de 190..

Y de manipular los mensajes de botones de control para la simulación como son: Paso a paso (ejecuta línea por línea), Simula (ejecuta de forma continua), Reset (inicializa todo el simulador), Paro (detiene Simula), Salir (termina la aplicación).

Ya que se ejecuta cualquiera de los botones ya sea "Paso a paso" o "Simula" se encarga de ejecutar cada una de las instrucciones, actualizar registros, memoria, puertos o lo que vaya asociado con la instrucción ejecutada.

También es el encargado de cualquier tipo de modificación que se haga en las partes editables del simulador para que sea en forma dinámica y automática.

Trabaja en coordinación con Instrucción ya que de ahí toma las instrucciones a ejecutar en Hexadecimal, como apoyo para realizar las operaciones en hexadecimal básicas ya declaradas.

Instrucción

Esta parte es la encargada de crear la clase CInstrucción. El archivo en donde se encuentra es en INSTRU.H.

Hexadecimal

Es la parte encargada de realizar las operaciones en hexadecimal y para ello se creó la clase CHexadecimal que se encuentra en el archivo HEXA.CPP

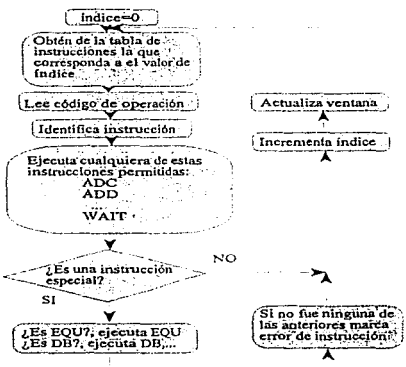


Figura 2.4.2 Diagrama de flujo para la ejecución de una instrucción

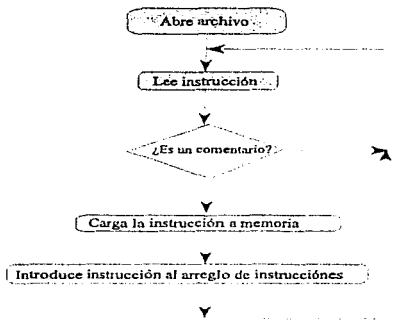


Figura 2.4.3 Diagrama de Flujo para cargar el programa a memoria

Conversión

Es la parte encargada de realizar la conversión de hexadecimal y binario a entero y de hexadecimal a binario y viceversa. El archivo en el que se encuentran es Conver.cpp..

Se mostraran los siguientes elementos en la ventana de ejecución:

- Nombre y directorio del programa **.prn* a ejecutar.

- Nombre del MCU a programar.

- Número de pines del MCU.

- Interrupción externa
  INT2

- Línea que se está ejecutando

- Del CPU:

  Acumulador (valor binario y hexadecimal).
  Registro Índice X (valor binario y hexadecimal).
  Registro TDR (valor binario y hexadecimal).
  Registro TCR (valor binario y hexadecimal).
  Registro de Códigos de Condición (H, I, N, C, Z).

-De la memoria

  Se muestran todas las localidades de la página cero.
  Se muestra el programa cargado en memoria.
  Se muestra las localidades que ocupan el MOR, vector de Interrupciones

(internas y externas), vector de reset.

- Los puertos

Puerto A, B, C y D según sea el caso del MCU.
DDR A, B, C según sea el caso del MCU.

- Del control de simulación

Paso a paso
Simula
Paro
Reset
Salir

2.5 Presentación y formato del programa fuente y objeto para generar archivos de trabajo del ensamblador.

Programa Fuente:

El programa fuente deberá ser un archivo tipo texto, que separe mediante espacios cada elemento de una línea de instrucción.

Programa .prn (programa de impresión):

Es un archivo tipo texto generado por el ensamblador y que contiene el programa fuente ensamblado con sus direcciones correspondientes y los códigos de operación que el microcontrolador reconocerá.

Programa .mik (programa objeto):

Es el archivo objeto del programa fuente que contiene el código ensamblado en forma continua. Cada línea es de 18 bytes, los primeros dos bytes indican la dirección de memoria de la instrucción. Se genera un cambio de línea cuando encuentra la instrucción "ORG" o bien cuando se completan los 18 bytes de longitud.

2.6 Requerimientos para el sistema de hardware

Los elementos necesarios para la realizar la tableta programadora son listados a continuación.

Tabla 2.6.1 Lista de partes para la construcción de la tarjeta programadora

R1	100 kΩ	Q1	2N2222
R2	4.7 kΩ	Q2	2N2222

R3	4.7 kΩ
R4	510 Ω
R5	510 Ω
R6	4.7 kΩ
R7	4.7 kΩ
C1	0.1 µF
C2	1.0 µF
C3	100 pF
C4	1.0 µF
C5	1.0 µF
C6	10 µF
C7	10 µF
D1	1N4001
D2	22V Zener (1N4748A)
D3	1N4001
D4	1N4001

Y1	1 Mhz (100Ω máx.)
U1	MC68705P3
U2	MC68705R3/U3
U3	MCM2732
U4	MC14040B
VR1	ASTEC Convertidor de voltaje 26A05
VR2	MC78L12
DS1	LED Rojo
DS2	LED Verde
PCB1	Tableta fenólica 15x15cm
Misc:	1 Base 40 pines con palanca
	1 Base 28 pines con palanca
	1 Base 24 pines con palanca
	1 Base 16 pines con palanca
	2 Switches

Donde:

R# Resistencia número #

C# Capacitor número #

D# Diodo número #

Y# Cristal de cuarzo número #

Q# Transistor bipolar número #

U1 y U2 Microprocesadores

U3 Memoria EPROM

U4 Contador binario

VR# Convertidor de voltaje número #

DS# LED (Diodo Emisor de Luz) número #

Capítulo 3

Diseño

y

Construcción

Capítulo 3
Diseño y Construcción

3.1 Implementación de los circuitos de conteo, tiempo y voltajes de programación

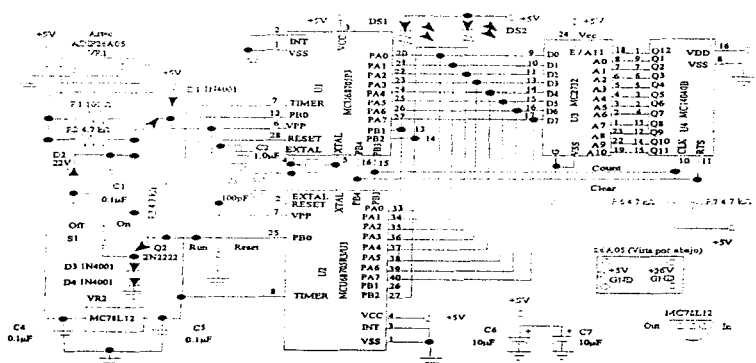


Figura 3.1.1 Diagrama esquemático de la tarjeta programadora de los MCU 68705P3, R3 y U3.

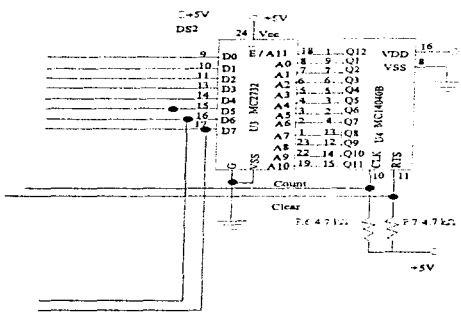


Figura 3.1.2 Circuito de conteo

Los pines de salida (Q1-Q11) del circuito de conteo están conectados a los pines de la memoria EPROM (A0-A11). Cuando el circuito de conteo sea activado se encargaran de hacer la cuenta para que con ella se direccione a la memoria de la EPROM y su contenido se vea reflejado en los pines de salida (D0-D7) y estos a su vez entren a los pines (PA0-PA7) del MCU correspondiente

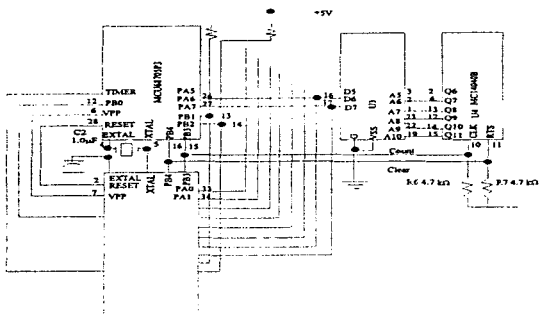
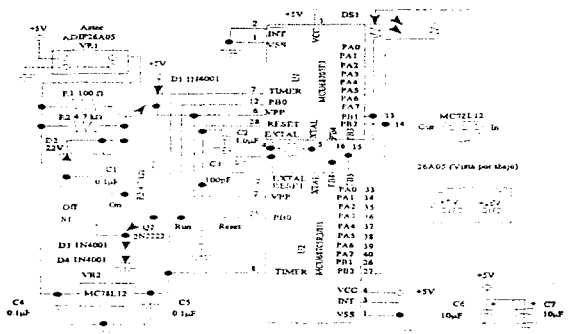


Figura 3.1.2 Circuito de tiempos de programación

El circuito de tiempo de programación consiste en la conexión del pin 15 del MCU (PB3) con el pin 10 del contador (CLK) con ello el MCU indica al contador cuando empezar a contar y cuando comenzar a transferir el contenido de la memoria EPROM hacia el MCU.



3.1.3 Circuito de voltajes

Este circuito consta de 2 partes la que genera el voltaje de 5 volts y la que genera los 26 volts.

El circuito de voltaje de 26 V consiste en el convertidor CD/CD 26A05 de Astec que convierte un voltaje de entrada de 5 V digital para obtener un voltaje de salida de 26 volts, con este mismo convertidor y conectando la salida de 26 volts al convertidor 78L12 se obtienen los 12 V para la programación de los MCU.

NOTA

Solamente programar un MCU a la vez, para evitar daños o la pérdida de la tarjeta programadora.

3.2 Desarrollo de las rutinas de compilación, ensamblado y preparación de los archivos de trabajo en Visual C++3.2

3.2.1 Programa CLASES.CPP

Contiene la declaración de todas las funciones utilizadas para realizar el proceso de ensamblado.

3.2.1.1 Definición de la clase "instruc"

class instruc : Almacena el conjunto de instrucciones permitidas.

Sus miembros privados son:

char nmo[6]: Contiene el mnemónico
char opcod[3]: Contiene el código del mnemónico en lenguaje ensamblador
char tipins: Tipo de instrucción

Sus funciones miembro son:

void init(char *Nmo[]*, char *Opcod[]*, char *Tipins*):

Entrada: Recibe como parámetros el mnemónico, el código de operación y el tipo de instrucción.
Salida: No retorna ningún valor.
Procesamiento: Esta función inicializa los miembros privados de su clase.

char veriftip(void):
Entrada: No recibe ningún parámetro.
Salida: Retorna el tipo de instrucción.
Procesamiento: Esta función identifica el tipo de instrucción del cual se trata.

int friend verifnmo(tablacomp **Nmo*, instruc **INSTRUCC*):
Entrada: Recibe dos objetos uno de tipo instruc y el otro de tipo tablacomp.
Salida: Retorna un valor entero. Un "1" si la comparación fue exitosa y un "0" en caso contrario.
Procesamiento: Verifica si los mnemónicos del programa fuente pertenecen al conjunto de instrucciones permitidas.

char * verifopcod(void):
Entrada: No recibe ningún parámetro
Salida: Retorna el código de operación (código en lenguaje ensamblador de determinado mnemónico).
Procesamiento: Esta función indica que código le pertenece a cada mnemónico del programa fuente.

3.2.1.2 Definición de la clase "tablacomp"

class tablacomp: Esta clase contiene los miembros apropiados para almacenar la tabla de símbolos y el resultado del proceso de ensamblado.

Sus miembros privados son:

int tipdir;	Valor de tipo entero que almacena el tipo de direccionamiento.
char tipins;	Cadena que almacena el tipo de instrucción.
int pc;	Valor entero que almacena el Program Counter.
int pcet;	Valor entero que almacena la dirección de una etiqueta.
char pc_ens[10];	Cadena que almacena el PC en hexadecimal.
char etiq[10];	Cadena que almacena una etiqueta.
char nmo[6];	Cadena que almacena el mnemónico de cada línea del programa fuente.
char opcod[10];	Cadena que almacena el código de operación de cada mnemónico.
char cod_ens[10];	Cadena que almacena el código ensamblado en la segunda pasada.
char oper[L_OP];	Cadena que almacena el operando de las instrucciones en caso de que lo requieran.
char coment[TAMI];	Cadena que almacena el comentario de cada línea del

programa fuente, si es que existe.

Sus funciones miembro son:

void pci(int):

Entrada: Ninguna.

Salida: Ninguna.

Procesamiento: Esta función inicializa el valor del PC.

int friend verifnmo(tablacomp **Nmo*, instruc **INSTRUCC*, int &):

Entrada: Recibe dos mnemónicos, el del programa fuente y cada uno de los mnemónicos permitidos además de un valor entero que indica si la cadena1 a comparar es mayor que, menor que o igual que la cadena 2.

Salida: Retorna un valor entero, "1" si la verificación fue exitosa o "0" en caso contrario.

Procesamiento: Con los datos de entrada realiza comparaciones para verificar si el mnemónico de cierta línea del programa fuente es permitido o no.

void asig_etiq(char *cad[]*):

Entrada: Recibe una cadena que contiene la etiqueta de determinada línea del programa fuente.

Salida: No retorna ningún valor.

Procesamiento: Esta función almacena el contenido del parámetro cad en la variable privada etiqueta de la clase.

void asig_pcet(int):

Entrada: Recibe un valor entero que contiene el valor del PC de una etiqueta.

Salida: No retorna ningún valor.

Procesamiento: Esta función almacena el contenido del parámetro que recibe en la variable privada pcet de la clase.

int verifop(tablacomp **tabla1*, int *&sum*):

Entrada: Recibe las líneas del programa fuente y las almacenadas en el parámetro tabla1 y un valor entero que contiene el valor en que se incrementará el siguiente PC.

Salida: Retorna un valor entero, "1" si la verificación fue exitosa y "0" en caso contrario.

Procesamiento: Verifica operandos que comiencen con "#", ","'"$" y de instrucciones del tipo C,D,F,A,E.

void verifop2(int *I*, tablacomp **tabla1[]*, int *&sum*, CString&, int&):

Entrada: Recibe un entero que contiene el valor actual de la línea a ensamblar, las líneas del programa fuente, un entero que contiene el valor en que se incrementará el siguiente PC, una cadena tipo

CString que contiene los mensajes de error en caso de que existan y un entero que contiene la línea en la cual se detectó el error.

Salida: No retorna ningún valor.

Procesamiento: Verifica los operandos de instrucciones del tipo A y E.

int bus_et(char *Oper[]*):

Entrada: Recibe un arreglo de caracteres que contienen el operando.

Salida: Retorna un valor entero "1" si la búsqueda fue exitosa y "0" en caso contrario.

Procesamiento: Esta función realiza la búsqueda de etiquetas a lo largo del programa fuente.

int veriflb(char *cad[]*);

Entrada: Recibe un arreglo de caracteres que contienen la cadena a verificar.

Salida: Retorna un valor entero "1" si la verificación es verdadera y "0" en caso contrario.

Procesamiento: Esta función verifica si determinada línea del programa fuente es una línea en blanco.

int verifet(char []):

Entrada: Recibe un arreglo de caracteres que contienen la cadena a verificar.

Salida: Retorna un valor entero "1" si la verificación es verdadera y "0" en caso contrario.

Procesamiento: Esta función verifica si determinada línea del programa fuente contiene o no una etiqueta.

int recorre(int *i*, char[]):

Entrada: Recibe un valor entero que contiene la posición actual de la cadena y la cadena.

Salida: Retorna un valor entero que representa la siguiente posición de interés.

Procesamiento: Mediante esta función se va recorriendo la cadena que representa cada línea del programa fuente.

void inc_sig_pc(int *val*, int):

Entrada: Recibe dos valores enteros, uno contiene el valor actual del PC y el otro el valor en que se incrementará el siguiente PC.

Salida: No retorna ningún valor.

Procesamiento: Esta función incrementa el valor del siguiente PC.

int ver_val(int *n*, int & *sum*, int):

Entrada: Recibe tres enteros.

Salida: Retorna un valor entero que representa el valor encontrado.

Procesamiento: De los datos que recibe el primero representa el valor a verificar, el segundo representa el valor en que se incrementará el PC, el tercero representa el valor con el cual se comparará.

void toma_lin2(int *i*,int *Lin*, tablacomp * *Tabla1[]*, CString&, int&):

Entrada: Recibe un valor entero.

Salida: No retorna ningún valor.

Procesamiento: Del dato que recibe representa el índice de su tercer parámetro Tabla1, el segundo parámetro es un valor entero que representa el número de líneas ensambladas hasta el momento de la llamada a esta función, el tercer parámetro es un objeto que contiene el conjunto de líneas del programa fuente recorridas hasta ese momento, el siguiente parámetro del tipo CString contiene los mensajes de error en caso de que existan, el último parámetro es un valor entero que representa la línea en la cual se detectó el error.

int salto_rel(int ,int, int &):

Entrada: Recibe 3 valores enteros, el primero contiene la dirección de la etiqueta, el segundo el valor actual del PC y el tercero el número de bytes que saltará.

Salida: Retorna un valor entero que representa el valor del salto.

Procesamiento: Esta función realiza el cálculo de saltos relativos.

int val_asc(char *C[]*, int *i*);

Entrada: Recibe la cadena a convertir y el índice de dicha cadena.

Salida: Retorna un valor entero que contiene el valor entero de la cadena.

Procesamiento: Mediante esta función se obtiene el valor ASCII de un carácter.

int conv_dec(char **Car*, int *&entero*):

Entrada: Recibe la cadena a convertir y una variable en la cual se almacenará su valor decimal en caso de que sea un número hexadecimal válido.

Salida: Retorna un valor entero que indica si la conversión fue posible o no, "1" si fue exitosa y "0" en caso contrario.

Procesamiento: Esta función convierte una cadena que para este caso representará un número en hexadecimal a su correspondiente valor entero. Dentro de esta función se realiza la validación para determinar si la cadena a convertir es un valor hexadecimal o no y dependiendo de este resultado se realiza o no la conversión.

void asig_nmo(char *cad[]*);

Entrada: Recibe una cadena que contiene el mnemónico.

Salida: No retorna ningún valor.

Procesamiento: Esta función almacena un mnemónico en la variable nmo (mnemónico) de su clase.

void asig_opcod(char *cad[]*):

Entrada: Recibe una cadena que contiene el código de operación.

Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el código de operación de cada mnemónico del programa fuente.

void asig_tipins(char):
Entrada: Recibe una cadena que contiene el tipo de instrucción.
Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el tipo de instrucción correspondiente a cada mnemónico del programa fuente.

void asig_oper(char cad[]):
Entrada: Recibe una cadena que contiene el operando.
Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el operando de cada línea del programa fuente.

void asig_tipdir(int *d*):
Entrada: Recibe un valor entero que contiene el tipo de direccionamiento.
Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el tipo de instrucción de determinada línea de código fuente.

void direcI(void):

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el código ensamblado para el tipo de direccionamiento no. 1.

void direc2(int *Lin*, tablacomp **tabla1[]*):

Entrada: Recibe un entero que contiene el número total de líneas de programa fuente, un objeto tablacomp que contiene las líneas del programa fuente.
Salida: No retorna ningún valor.
Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 2.

void direc3(int *Lin*, tablacomp **tabla1[]*, int lin, CString&, int&):

Entrada: Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.
Salida: No retorna ningún valor.
Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento

no. 3.

void direc4(int *Lin*, tablacomp **tabla1[]*, int *lin*, CString&, int&):

Entrada:	Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.
Salida:	No retorna ningún valor.
Procesamiento:	Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 4

void direc5(void):

Entrada:	No recibe ningún valor.
Salida:	No retorna ningún valor.
Procesamiento:	Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 5.

void direc6(int *Lin*, tablacomp **tabla1[]*, int *lin*, CString&, int&):

Entrada:	**Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en**

la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.

Salida: No retorna ningún valor.

Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 5

void direc7(int *Lin*, tablacomp **tabla1[]*, int lin, CString&, int&):

Entrada: Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.

Salida: No retorna ningún valor.

Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 7.

void direc8(int *Lin*, tablacomp **tabla1[]*, int lin, CString&, int&):

Entrada: Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una

cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.

Salida: No retorna ningún valor.

Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 8.

void direc9(int *Lin*, tablacomp **tabla1[]*, int lin, CString&, int&):

Entrada: Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se generen y un entero que indica si se generó o no algún error.

Salida: No retorna ningún valor.

Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 9.

void direc10(int *Lin*, tablacomp **tabla1[]*, int lin, CString&, int&):

Entrada: Recibe un entero que contiene el total de líneas ensambladas, un objeto del tipo tablacomp que contiene las líneas ensambladas en la primera pasada, un entero que indica la línea a ensamblar, una cadena de tipo CString que contiene los mensajes de error que se

generen y un entero que indica si se generó o no algún error.

Salida: No retorna ningún valor.

Procesamiento: Esta función se ejecuta durante la segunda pasada y genera el código ensamblado correspondiente al tipo de direccionamiento no. 10.

int ext_pc(void):

Entrada: No recibe ningún parámetro.

Salida: Retorna el valor entero que contiene el valor del PC.

Procesamiento: Mediante esta función se extrae el PC de la línea ensamblada que se desee.

int ext_pcet(void):

Entrada: No recibe ningún parámetro.

Salida: Retorna el valor entero que contiene el valor del PC.

Procesamiento: Mediante esta función se extrae el PC de la etiqueta o variable que se desee.

int ext_tipdir(void):

Entrada: No recibe ningún parámetro.

Salida: Retorna el valor entero que contiene el tipo de direccionamiento.

Procesamiento: Mediante esta función se extrae el tipo de direccionamiento de la línea ensamblada que se desee.

void salvar(FILE *_arch_, int _b_):

Entrada: Recibe un apuntador de tipo FILE que contiene la dirección del archivo a almacenar, y un valor entero que indica si la línea a almacenar contiene código ensamblado o no.

Salida: No retorna ningún valor.

Procesamiento: Mediante esta función se almacena el archivo prn.

void salvar_mik(FILE *_arch_, int _b_, _int_ &):

Entrada: Recibe un apuntador de tipo FILE que contiene la dirección del archivo a almacenar, un valor entero que indica si la línea a almacenar contiene código ensamblado o no y un valor entero que indica el total de caracteres por línea.

Salida: No retorna ningún valor.

Procesamiento: Mediante esta función se almacena el archivo mik.

void tabla_simb(FILE *_arch_):

Entrada: Recibe un apuntador de tipo FILE que contiene la dirección del archivo en el cual almacenará datos.

Salida: No retorna ningún valor.

Procesamiento: Mediante esta función se almacena la tabla de símbolos en el archivo prn.

void asig_pc_ens(char *_pc_):

Entrada: Recibe la cadena que contiene el valor del PC en valor hexadecimal.
Salida: No retorna ningún valor.
Procesamiento: Esta función almacena el PC en valor hexadecimal en el campo pc_ens (PC ensamblado) durante la segunda pasada.

void ext_nmo(char *aux_nmo[]*);

Entrada: Recibe una cadena vacía que contendrá un mnemónico.
Salida: No retorna ningún valor.
Procesamiento: Entra el mnemónico deseado almacenándolo en aux_nmo.

void direc_END(char *cad[]*, **int** *&i*);

Entrada: Recibe una cadena que contiene la cadena que se está validando y un entero que contiene la posición actual en la cadena.
Salida: No retorna ningún valor.
Procesamiento: Recibe la cadena y la recorre verificando así si tiene comentario o no.

void direc_ORG(char *cad[]*, **int** *&i*, **tablacomp** **tabla1[]*, **int lin, CString&, int&);**

Entrada: Recibe una cadena que contiene la cadena que se está validando, un entero que contiene la posición actual en la cadena, el conjunto de líneas del programa validadas hasta el momento , el número de líneas ensambladas, una cadena que contendrá un mensaje en caso

de existir un error y un valor entero que contendrá "1" en caso de que se detecte un error y "0" en caso contrario.

Salida: No retorna ningún valor.

Procesamiento: Al igual que la función anterior recibe la cadena y la recorre verificando si contiene comentario y operando.

void direc_EQU(char *cad[]*, int *&i*, tablacomp **tabla1[]*, int lin, CString&, int&);

Entrada: Ver función direc_ORG

Salida: Ver función direc_ORG

Procesamiento: Ver función direc_ORG

void direc_DW_DB(char *cad[]*, int *&i*, tablacomp **tabla1[]*, int *lin*, char *aux[]*,CString&, int&);

Entrada: Similar a la función direc_ORG excepto que además recibe el parámetro "char aux[]" el cual es una cadena que contiene la directiva a validar.

Salida: No retorna ningún valor.

Procesamiento: Similar a la función_ORG excepto en que con esta función se reconocen dos directivas DW y DB, esto se logra comparando cual de las dos se recibe mediante el parámetro aux.

void asig_coment(char *cad[]*);

Entrada: Recibe una cadena que contiene el comentario.

Salida: No retorna ningún valor.
Procesamiento: Almacena la cadena recibida en la variable comentario de la clase.

void vacio(void);
Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Esta función se utiliza en casos en que la línea que se valida sea un comentario y una línea con espacios o tabuladores, en estos casos coloca todos los valores enteros en -1 y las cadenas las copia con la cadena vacía.

Además se declararon las siguientes funciones:

int ut_aux(int *I*, char *cad[]*, char *aux[]*, char *c_fin*);
Entrada: Recibe un entero que contiene la posición de la cadena, una cadena que contiene una línea del programa fuente, una cadena auxiliar en la cual se almacenará el parámetro cad y un carácter de terminación.
Salida: Retorna un entero que representa la nueva posición en el arreglo.
Procesamiento: Esta función recibe cada línea del programa fuente almacenando parte de esta cadena en aux, es decir almacena la cadena hasta la posición en la cual se cumple que cad[I]=c_fin.

void limp_aux(char *aux[]*, **int** *tam*);

Entrada:	Recibe una cadena que contiene la cadena a limpiar y un entero que indica el tamaño de la misma.
Salida:	No retorna ningún valor.
Procesamiento:	Esta función limpia el contenido de la cadena que recibe como parámetro tomando en cuenta su tamaño.

void msj_err(int *No_msj*, **int** *Linea*, **CString&, int &);**

Entrada:	Recibe un entero que contiene el número de error, un entero que indica la línea en la cual se generó el error, una cadena que contendrá el mensaje de error generado y un entero que toma un valor de "1" si se detectó error y "0" en caso contrario.
Salida:	No retorna ningún valor.
Procesamiento:	Esta función permite reconocer los diferentes mensajes que se le enviarán al usuario al terminar el proceso de ensamblado.

3.2.1.3 Definición de las constantes utilizadas

const NO_INSTRUC = 85;	Número de instrucciones almacenadas por default.
const L_OP=20;	Número de caracteres permitidos para el operando.
const TAMI =90;	Número de caracteres permitidos para el nombre del archivo y para la cadena que contiene la instrucción.

const NO_INST_USR = 180; Número de líneas permitidas en el programa fuente.

3.2.2 Programa ENS6805.CPP

Funciones para el manejo del editor de textos y ejecución del proceso de ensamblaje.

3.2.2.1 Definición de la clase "CxDialog"

class CxDialog : public CDialog: Es derivada de la clase CDialog, se utiliza para manejar el cuadro de diálogo "abrir archivo".

Sus miembros son:

enum { IDD = ID_ABRIR }: Identificador enumerado que contiene el nombre del identificador utilizado para abrir archivo.

CString m_archivo: Cadena que contiene el nombre del archivo.

CString m_directorio: Cadena que contiene la ruta en la cual se localiza el archivo.

Sus funciones miembro son:

CxDialog():

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Es su constructor estándar; inicializa el nombre y la ruta del archivo.

virtual void DoDataExchange(CDataExchange* *pDX*):
Entrada: Recibe un apuntador a las variables de mensajes.
Salida: No retorna ningún valor.
Procesamiento: Es una función virtual sobrepasada. Se utiliza para el intercambio de datos, es decir, hace posible la transferencia de datos desde los miembros de datos hasta los controles de diálogo y viceversa.

3.2.2.2 Definición de la clase "CxGDialog"

class CxGDialog : public CDialog: Al igual que la clase CxDialog, es derivada de la clase CDialog, y soporta el cuadro de diálogo utilizado para almacenar archivos..

Sus miembros son:

enum { IDD = ID_GUARDAR }: Identificador enumerado que contiene el nombre del identificador utilizado para guardar el archivo.

CString m_arch: Cadena que contiene el nombre del archivo.
CString m_direc: Cadena que contiene la ruta en la cual se localiza el archivo.

Sus funciones miembros son:

CxGDialog():
Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Constructor estándar. Ver clase CxDialog.

virtual void DoDataExchange(CDataExchange* pDX):
Entrada: Recibe un apuntador a las variables de mensajes.
Salida: No retorna ningún valor.
Procesamiento: Función de intercambio de datos. Ver clase CxDialog.

3.2.2.3 Definición de la clase "CxEdit"

class CxEdit: public CEdit: Clase utilizada para manipular la edición y formateo de textos.

Su función miembro es:

int GetLineLength(int *nLineNumber*):

Entrada: Recibe un entero que almacena el número de línea.
Salida: Retorna la longitud de la línea.
Procesamiento: Esta función calcula la longitud de una línea.

3.2.2.4 Definición de la clase "CWindowApp"

class CWindowApp: public CWinApp: Es una clase derivada de la clase CWinApp y se utiliza para manipular los mensajes de ventana y los mensajes de órdenes del programa.

Sus miembros son:

Tiene una función miembro:

virtual BOOL InitInstance():

Entrada: No recibe ningún valor.
Salida: Retorna un valor BOOLEANO "1" si la operación fue exitosa y "0" en caso contrario.
Procesamiento: Inicia el proceso de cargar un documento y de visualizar las ventanas.

3.2.2.5 Definición de la clase "CAppMDIChild"

class CAppMDIChild: public CMDIChildWnd: Es una clase derivada de la clase

CMDIChildWnd, se utiliza para almacenar la vista.

Sus miembros son:

int nChildNum;	Entero que almacena el número de ventana MDI abierta.
CxEdit *TextBox	Apuntador al control de cuadro de edición.
char archv[MaxEditLen+1];	Cadena que contiene el nombre del archivo abierto.

Sus funciones miembro son:

CAppMDIChild():

Entrada:	No recibe ningún valor.
Salida:	No retorna ningún valor.
Procesamiento:	Es el constructor estándar, inicializa la caja de texto con NULL.

~CAppMDIChild():

Entrada:	No recibe ningún valor.
Salida:	No retorna ningún valor.
Procesamiento:	Es el destructor estándar, destruye la caja de texto.

BOOL Create(LPSTR *szTitle*, int *nChildNum*):

Entrada: Recibe un valor FAR que contiene el título de la ventana y un entero que contiene el número de ventana hija MDI.

Salida: Retorna un valor BOOLEANO si la operación fue exitosa almacena un "1" y "0 " en caso contrario.

Procesamiento: Crea una ventana hija MDI.

afx_msg void OnClose():

Entrada: No recibe ningún valor.

Salida: No retorna ningún valor.

Procesamiento: Maneja el cierre de la ventana hija MDI.

afx_msg void OnDestroy():

Entrada: No recibe ningún valor.

Salida: No retorna ningún valor.

Procesamiento: Maneja la destrucción de una ventana hija MDI

afx_msg int OnCreate(LPCREATESTRUCT *lpCS*):

Entrada: Recibe un apuntador FAR a la estructura que contiene toda la información del recurso a crear

Salida: Retorna un valor entero "0"como valor de la función

Procesamiento: Esta función se utiliza para calcular el tamaño y la ubicación del control de edición, crea un control de edición accesado por el miembro TextBox. Maneja la creación de controles

afx_msg void CMGuardar():

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Guarda la ventana hija MDI activa.

afx_msg void CMAbrir():

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Abre una nueva ventana hija MDI.

void CMEnsamblar():

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Realiza el proceso de ensamblado del archivo abierto.

3.2.2.6 Definición de la clase "CAppMDIFrame"

class CAppMDIFrame: public CMDIFrameWnd: Esta clase es derivada de la clase CMDIFramWnd y se utiliza para controlar las ventanas hijas y los mensajes.

Sus miembros son:

BOOL bExpressClose:	Valor booleano indicador para cerrar rápidamente todas las ventanas hijas MDI.
int nLastMDIChild:	Valor entero que almacena el índice más alto de una ventana hija MDI.
CMenu *pMenu:	Apuntador al menú asociado con la ventana marco MDI.
int nNumMDIChildren:	Valor entero que almacena la cantidad actual de ventanas hijas MDI.

Sus funciones miembro son:

CAppMDIFrame():

Entrada:	No recibe ningún valor.
Salida:	No retorna ningún valor.
Procesamiento:	Constructor estándar, el cual llama a la función miembro Create para especificar el título, el estilo (maximizada o traslapada) y el recurso de menú MAINMENU de la ventana. También inicializa los datos miembro bExpressClose, nLastMDIChild y nNumMDIChildren.

~CAppMDIFrame():

Entrada:	No recibe ningún valor.

Salida: No retorna ningún valor.

Procesamiento: Es el destructor de la clase y su función es borrar el menú dinámico accesado por el apuntador pMenu.

afx_msg int OnCreate(LPCREATESTRUCT *lpCS*):

Entrada: Recibe un apuntador FAR a la estructura que contiene toda la información del recurso a crear

Salida: Retorna un valor entero cero como resultado de la función.

Procesamiento: Esta función responde al mensaje WM_CREATE, enviado por la función Create. Mediante esta función se crea una nueva área de ventana cliente.

afx_msg void CMCreateChild():

Entrada: No recibe ningún valor.

Salida: No retorna ningún valor.

Procesamiento: Crea una ventana hija MDI.

afx_msg void CMCascadeChildren()

Entrada: No recibe ningún valor.

Salida: No retorna ningún valor.

Procesamiento: Pone en cascada a las hijas MDI.

afx_msg void CMTileChildren():

Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Coloca en mosaico a las hijas MDI.

afx_msg void CMArrageIcons():
Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Acomoda los iconos hijos MDI.

afx_msg void CMCloseChildren():
Entrada: No recibe ningún valor.
Salida: No retorna ningún valor.
Procesamiento: Cierra todas las hijas MDI.

int GetChildCount():
Entrada: No recibe ningún valor.
Salida: Retorna un entero que contiene el número de ventanas hijas MDI.
Procesamiento: Obtiene el número de hijas MDI.

afx_msg void CMCountChildren():
Entrada: **No recibe ningún valor.**
Salida: **No retorna ningún valor.**
Procesamiento: **Maneja el comando para la cuenta de hijas MDI.**

afx_msg LONG OnChildDestroy(UINT *wParam*, LONG *lParam*):

Entrada:	Para uso interno de la función no son visibles al usuario
Salida:	Retorna el nuevo número de la ventana siguiente
Procesamiento:	Maneja el mensaje de destrucción de la ventana MDI hija

afx_msg void OnExit():

Entrada:	No recibe ningún valor
Salida:	No retorna ningún valor
Procesamiento:	Maneja la salida de la aplicación

virtual void OnClose():

Entrada:	No recibe ningún valor
Salida:	No retorna ningún valor
Procesamiento:	Maneja el cierre de una ventana marco MDI

Otras funciones declaradas son:

void makeRect(int *nX*, int *nY*, int *nW*, int *nH*, CRect *&r*):

Entrada:	Recibe cuatro valores enteros, el primero almacena la posición en la coordenada "x", el segundo almacena la posición en la coordenada "y", el tercero almacena la longitud, el cuarto el alto. El quinto parámetro es de tipo CRect que define un rectángulo.
Salida:	No retorna ningún valor.

Procesamiento: Calcula el tamaño y la ubicación de un control utilizando las coordenadas y tamaños recibidos como parámetros.

3.2.2.7 Definición de las constantes utilizadas:

const char *MenuRezName="MAINMENU"	Identificador del menú
const Hctl=200;	Longitud horizontal máxima de un control
const Wctl=400;	Longitud vertical máxima de un control
const MaxEditLen=70	Longitud máxima de algunas cadenas utilizadas durante el programa

En algunas de las funciones (como se podrá observar en el listado del programa fuente), se utiliza la función **DECLARE_MESSAGE_MAP()**, la cual se utiliza para manipular el mapeo de los mensajes generados por las funciones.

3.3 Desarrollo de las rutinas de simulación y preparación de los archivos de trabajo en Visual C++

Tabla 3.3.1 Descripción de las clases principales que manejan el simulador

Clase	Descripción
CxDialog	Es la encargada de la ventana principal del simulador y realizar todas las operaciones que en ella se aplican.
CxResDlg	Es la encargada de la ventana que recibe el nombre del archivo que se va a simular y la de seleccionar el tipo de MCU a simular.
CAboutDlg	Es la clase encargada de mencionar quien diseño el simulador y mencionar la versión de este.
CHexadecimal	Esta clase fue creada para que en ella se realizaran las operaciones aritméticas y lógicas entre números hexadecimales y también de ella se obtiene el número binario correspondiente al resultado de la operación.

Clase	Descripción
CInstruccion	Esta clase contiene los datos básicos para ejecutar las instrucciones por línea como son: PC (contador del programa) Etiqueta Código de Operación Operando y es la base para construir el arreglo de instrucciones por programa para ejecutarse.

3.3.1 Programa principal SIM6805.CPP

Programa principal para la creación y manejo del simulador de los MCU's 68705P3, R3 y U3 y contiene los siguientes elementos:

3.3.1.1 Declaración de la clase "CWindowApp"

class CWindowApp: public CWinApp — Derivada de CWinApp. Inicializa la construcción de la aplicación.

Su función miembro es:

virtual BOOL InitInstance();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Crea el cuadro de dialogo sin modo de la ventana principal. Pone el fondo de este en color gris. Ejecuta la ventana principal.

Se ejecuta la siguiente línea:

CWindowApp NEAR WindowApp;	El constructor de la aplicación inicializa y ejecuta la aplicación

3.3.2 Programa XDIALOG.H

Programa que maneja las clases CxDialog, CxResDlg y CAboutDlg.

3.3.2.1 Declaración de la clase "CxDialog"

class CxDialog : public CDialog

Datos privados:

long m_lCount;	Maneja el conteo de las instrucciones que se ejecutan con el

botón "Simula"

long m_lMaxCount; Maneja el máximo valor de instrucciones que se pueden ejecutar con el botón "Simula"

Constructor:

CxDialog();

Datos públicos:

CButton	**m_pasoapaso;**	Control del botón "Paso a paso"
CButton	**m_simula;**	Control del botón "Simula"
CButton	**m_reset;**	Control del botón "Reset"
CString	**m_archivo;**	Contiene el nombre del archivo
CString	**m_C;**	Contiene el valor de el bit C del CC
long	**m_ciclos;**	Contiene el número de ciclos de ejecución.
CString	**m_cp;**	Contiene el contador del programa
CString	**m_DDRAb;**	Contiene el valor del DDR del puerto A en binario
CString	**m_AAb;**	Contiene el valor del acumulador en binario
CString	**m_AAh;**	Contiene el valor del acumulador en hexadecimal
CString	**m_DDRAh;**	Contiene el valor del DDR del puerto A en hexadecimal
CString	**m_DDRBb;**	Contiene el valor del DDR del puerto B en binario.
CString	**m_DDRBh;**	Contiene el valor del DDR del puerto B en

		hexadecimal
CString	**m_DDRCb;**	Contiene el valor del DDR del puerto C en binario.
CString	**m_DDRCh;**	Contiene el valor del DDR del puerto C en hexadecimal
CString	**m_directorio;**	Contiene el directorio del archivo de instrucción a ejecutar
CString	**m_H;**	Contiene el valor del bit H del CC
CString	**m_I;**	Contiene el valor del bit I del CC
CString	**m_INDXb;**	Contiene el valor del registro INDX en binario
CString	**m_INDXh;**	Contiene el valor del registro INDX en hexadecimal
CString	**m_N;**	Contiene el valor del bit N del CC
CString	**m_lineaprog;**	Contiene la línea del archivo que se esta ejecutando
CString	**m_PORTAb;**	Contiene el valor del puerto A en binario
CString	**m_PORTAh;**	Contiene el valor del puerto A en hexadecimal
CString	**m_PORTBb;**	Contiene el valor del puerto B en binario
CString	**m_PORTBh;**	Contiene el valor del puerto B en hexadecimal
CString	**m_PORTCb;**	Contiene el valor del puerto C en binario
CString	**m_PORTCh;**	Contiene el valor del puerto C en hexadecimal
CString	**m_PORTDb;**	Contiene el valor del puerto D en binario
CString	**m_PORTDh;**	Contiene el valor del puerto D en hexadecimal

CString	**m_procesador;**	Contiene el nombre del MCU que se va a simular
CString	**m_SP;**	Contiene el valor de la dirección del apuntador de la pila
CString	**m_TCRh;**	Contiene el valor del TCR en hexadecimal
CString	**m_TCRb;**	Contiene el valor del TCR en binario
CString	**m_TDRb;**	Contiene el valor del TDR en binario
CString	**m_TDRh;**	Contiene el valor del TDR en hexadecimal
CString	**m_Z;**	Contiene el valor del bit Z del registro CC
UINT	**m_pins;**	Contiene el número de pines del MCU
CString	**m_mem;**	Contiene toda la información de la memoria del MCU
CString	**m_programa;**	Contiene todo el programa a simular
CString	**m_segundos;**	Contiene el tiempo de simulación
CString	**m_frec;**	Contiene la frecuencia del reloj que se está simulando
CString	**m_INT;**	Contiene el valor del pin INT para interrupciones externas
CString	**m_timer;**	Contiene el valor del pin TIMER para interrupción por temporizador
CInstruccion pInstruccion[190];		Contiene todas las líneas del programa de instrucciones a ejecutar
CString	**PC;**	Contiene el valor del contador del programa
CString	**CO;**	Contiene el valor del código de operación
CString	**E;**	Contiene el nombre de la etiqueta

CString	**Nmo;**	Contiene el mnemónico
CString	**Op;**	Contiene el operando
BOOL	**Et;**	Indica si hay o no una etiqueta en la línea a ejecutar
CString	**Org;**	Contiene el valor de la dirección de la instrucción "ORG"
CString	**Org1;**	Contiene el origen del programa
int	**Indexant;**	Contiene el valor del índice anterior
BOOL	**EsOrg;**	Indica si la instrucción a ejecutar es "ORG" o no
BOOL	**EsOrg1;**	Indica si la instrucción a ejecutar es "ORG" o no y si es el de inicio de programa
int	**índice;**	Contiene el índice de la instrucción que se está ejecutando
int	**bandera;**	Indica si se ha realizado alguna modificación en algún cuadro de edición
int	**count;**	Contiene el total de líneas a ejecutar
double	**segundos;**	Contiene en valor real el número de segundos de ejecución que han transcurrido
double	**fval;**	Contiene en valor real la frecuencia a la que se va a simular

Sus funciones miembro son:

void ObtenSubcadena(char *C);

Entrada:	Una cadena de caracteres
Salida:	Ninguna
Procesamiento:	Obtiene una subcadena de una cadena y la almacena en la variable *arreglo*

BOOL AbreArchivo(const char **archivo*);

Entrada:	Una cadena de caracteres que contiene el directorio y el nombre del archivo a abrir
Salida:	Un valor lógico que indica 0 no hubo problema para abrir el archivo y 1 si hubo algún problema
Procesamiento:	Abre el archivo a simular y lo carga en memoria

void IniciaDatos();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Inicializa los valores del simulador

void EjecutaInstruccion();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	**Dependiendo del código de operación y al tipo de direccionamiento se ejecuta la instrucción asociada, p.e. Codigo de operación=A9, Instrucción=ADC y se efectua la operación ADC**

con direccionamiento inmediato..

void IniciaMemoria();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Inicia la página cero de la memoria.

void CargaProgramaAMemoria(CString *a0*, CString *a1*);

Entrada:	Cadena *a0* que contiene el valor del contador del programa y *a1* que contiene el valor en hexadecimal del código de operación y el operando .
Salida:	Ninguna
Procesamiento:	Introduce a memoria el valor del código de operación y el operando en las localidades de memoria correspondientes a la dirección que da *a0* (contador del programa)

int CxDialog::BuscaDireccionMemoria(CString *a0*);

Entrada:	Cadena *a0* que contiene el contador del programa
Salida:	Valor entero que indica el índice de la cadena que maneja la memoria que es m_mem en donde encontró la dirección
Procesamiento:	Busca si existe el contador del programa y si no existe agrega a m_mem la línea en la que viene la dirección base de memoria y las 16 localidades siguientes y devuelve el índice en donde se localiza

la localidad indicada por el contador de programa

CString CxDialog::ObtenIndice(CString *dir*, int *&index*);

Entrada: La cadena *dir* contiene una dirección de memoria y el valor *index* por referencia en el que se regresa el índice en donde fue encontrada la localidad indicada por *dir*

Salida: Devuelve la cadena que encontró en la localidad indicada por *dir*.

Procesamiento: Busca en la memoria (m_mem) el índice en donde se encuentra la localidad indicada por *dir*

CString CxDialog::ObtenIndice(CString *dir*);

Entrada: La cadena *dir* contiene una dirección de memoria

Salida: Devuelve la cadena que encontró en la localidad indicada por *dir*.

Procesamiento: Busca en la memoria (m_mem) el índice en donde se encuentra la localidad indicada por *dir*.

void CxDialog::CalculaSegundos(void);

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Obtiene contiene el total de segundos transcurridos de la simulación

void CxDialog::IntroduceaMemoria(CString *dir*, CString *hex*, CString *bin*);

Entrada: Las cadenas dir (dirección de memoria), hex (valor en hexadecimal que se va a introducir en alguno de los puertos o DDR), bin (valor en binario que se va a introducir en alguno de los puertos o DDR)

Salida: Ninguna

Procesamiento: Introduce el valor correspondiente en binario y en hexadecimal de algún resultado que haya ocurrido en las instrucciones que manipulan o introducen datos en la página cero de la memoria, explícitamente si es en alguno de los puertos, DDR, TCR o TDR.

void CxDialog::CambiaDirMOReINT();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se esta simulando el MCU 68705R3 o el U3 cambia las direcciones del MOR, interrupciones, y reset de la memoria.

void CxDialog::Obtenfval();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Obtiene el valor de la frecuencia en la que se está simulando, dependiendo del valor contenido del registro MOR.

void CxDialog::EjecutaTimer();

Entrada: Ninguna

Salida: Ninguna
Procesamiento: Ejecuta una interrupción por timer cuando en la ventana principal en la parte de Timer se coloca un 1

void CxDialog::EjecutaIntExt();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Ejecuta una interrupción externa cuando en la ventana principal en la parte del -(INT2) se coloca un 0

void CxDialog::ADC(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: Efectúa la operación ACCA←ACCA+M+C [1]

void CxDialog::ADD(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si -(INT)=0

[1] Ver el glosario para abreviaturas

void CxDialog::AND(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Efectúa la operación ACCA←ACCA·M

void CxDialog::ASLoLSL(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Efectúa la operación traslación a la izquierda, introduciendo el bit más significativo en el bit C del registro CC y en el bit menos significativo un 0 (cero)

void CxDialog::ASR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Efectúa la operación traslación a la derecha, introduciendo el bit menos significativo en el bit C del registro CC

void CxDialog::BCCoBHS(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento

de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si C=0

void CxDialog::BEQ(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si Z=1

void CxDialog::BHCC(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si H=0

void CxDialog::BHCS(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si H=0

void CxDialog::BHI(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si (C o Z)=0

void CxDialog::BIH(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si -(INT)=1

void CxDialog::BIL(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: PC←PC+0002+Rel. Salta si -(INT)=0

void CxDialog::BIT(int *tipins*);
Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: ACCA · M (donde · representa la AND lógica)

void CxDialog::BLOoBCS(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si C=1

void CxDialog::BLS(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si C o Z =1

void CxDialog::BMC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si I=0

void CxDialog::BMI(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si N=1

void CxDialog::BMS(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC← PC+0002+Rel. Salta si I=1

void CxDialog::BNE(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si Z=0

void CxDialog::BPL(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel. Salta si N=0

void CxDialog::BRA(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instruccion

Salida: Ninguna

Procesamiento: PC←PC+0002+Rel

void CxDialog::BRN(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Nunca saltes equivale a 4 ciclos NOP

void CxDialog::BSR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+0002
(SP)←PCL; SP←SP-0001
(SP)←PCH; SP←SP-0001
PC←PC+Rel

void CxDialog::CLC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: bit C←0

void CxDialog::CLI(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento

	de la instrucción
Salida:	Ninguna
Procesamiento:	bit I←0

void CxDialog::CLR(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	X←00 o ACCA←00 o M←00

void CxDialog::CMP(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	ACCA-M

void CxDialog::COM(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	X←~X o ACCA←~ACCA o M←~M (donde ~ es negación)

void CxDialog::CPX(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: X-M

void CxDialog::DEC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: X←X-01 o ACCA←ACCA-01 o M←M-01

void CxDialog::EOR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: ACCA←ACCA ⊕ M

void CxDialog::INC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: X←X+01 o ACCA←ACCA+01 o M←M+01

void CxDialog::JMP(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←el operando contiene la dirección a saltar

void CxDialog::JSR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: PC←PC+N
(SP)←PCL; SP←SP-0001
(SP)←PCH; SP←SP-0001
PC← el operando contiene la dirección a saltar

void CxDialog::LDA(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: ACCA←M

void CxDialog::LDX(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento

de la instrucción

Salida: Ninguna

Procesamiento: X←M

void CxDialog::LSR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Desplazamiento lógico a la izquierda, el bit más significativo se introduce en el bit C del CC y en el bit menos significativo se introduce un 0 (cero).

void CxDialog::NEG(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: X←-X o ACCA←-ACCA o M←-M (complemento a dos)

void CxDialog::NOP(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Solamente el PC se incrementa y se efectúan 2 ciclos de operación,

no se modifica ningún registro.

void CxDialog::ORA(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: ACCA←ACCA v M (donde v es la operación lógica OR)

void CxDialog::ROL(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Rotación de ACCA, X o M a la izquierda con acarreo. El bit más significativo se introduce en el bit C del CC y el bit C del CC se introduce en el bit menos significativo y se trasladan los bits del registro que se esté operando a la izquierda

void CxDialog::ROR(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: Rotación de ACCA, X o M a la derecha con el bit C. El bit menos significativo se introduce en el bit C del CC y el bit C del CC se

introduce en el bit mas significativo y se trasladan los bits del registro que se esté operando a la derecha

void CxDialog::RSP(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	SP←S7F (limpia la pila (SP))

void CxDialog::RTI(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	SP←SP+0001; CC←SP SP←SP+0001; ACCA←SP SP←SP+0001; X←SP SP←SP+0001; PCH←SP SP←SP+0001; PCL←SP Retorna de subrutina

void CxDialog::RTS(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna
Procesamiento: SP←SP+0001; PCH←SP
SP←SP+0001; PCL←SP
Retorna de subrutina

void CxDialog::SBC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: ACCA←ACCA-M-C (resta con acarreo)

void CxDialog::SEC(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: bit C←1 (limpia bit de acarreo)

void CxDialog::SEI(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida: Ninguna
Procesamiento: bit I←1 (fija el bit de interrupción enmascarada)

void CxDialog::STA(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: M←ACCA (almacena el acumulador en memoria)

void CxDialog::STOP(int *tipins*);

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Válido sólo para CMOS

void CxDialog::STX(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: M←X (almacena el registro índice en memoria)

void CxDialog::SUB(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: ACCA←ACCA-M (resta)

void CxDialog::SWI(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	PC←PC+0001 SP←PCL; SP←SP-0001 PCH←PCH; SP←SP-0001 SP←X; SP←SP-0001 SP←ACCA; SP←SP-0001 SP←CC; SP←SP-0001 bit I←1, (interrupción por software)

void CxDialog::TAX(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	X←ACCA (transfiere el acumulador al registro índice X)

void CxDialog::TST(int *tipins*);

Entrada:	El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción
Salida:	Ninguna
Procesamiento:	X-00 o ACCA-00 o M-0 (verifica para negativo o cero)

void CxDialog::TXA(int *tipins*);

Entrada: El valor entero de *tipins* que indica el modo de direccionamiento de la instrucción

Salida: Ninguna

Procesamiento: ACCA←X (transfiere el registro índice X al acumulador)

void CxDialog::WAIT(int *tipins*);

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Valido sólo para CMOS

void CxDialog::especial();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Manda a ejecutar si la instrucción es DB o DW o EQU u ORG o END o BNCLR o BRSET o BSET o BCLR si no es ninguna de las anteriores manda un mensaje de error.

void CxDialog::DB();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: **Se efectúa cuando carga el programa a memoria**

void CxDialog::DW();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Se efectúa cuando carga el programa a memoria

void CxDialog::END();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Se efectúa cuando carga el programa a memoria

void CxDialog::EQU();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Se efectúa cuando carga el programa a memoria

void CxDialog::ORG();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Se efectúa cuando carga el programa a memoria

void CxDialog::BRCLR(int n);
Entrada: El número del bit a verificar
Salida: Ninguna

Procesamiento: PC←PC+0003 +Rel. Si el bit n de M es cero

void CxDialog::BRSET(int n);

Entrada: El número del bit a verificar

Salida: Ninguna

Procesamiento: PC←PC+0003 +Rel. Si el bit n de M no es cero

void CxDialog::BSET(int n);

Entrada: El número del bit a verificar

Salida: Ninguna

Procesamiento: M[n]←1

void CxDialog::BCLR(int n);

Entrada: El número del bit a verificar

Salida: Ninguna

Procesamiento: M[n]←0

Función miembro protegida:

DECLARE_MESSAGE_MAP()

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Maneja el mapa de mensajes de Windows y ejecuta una función

asociada a éste.

3.3.2.2 Declaración de la clase "CAboutDlg"

class CAboutDlg : public CDialog Crea la clase encargada de enviar la información de la versión del programa, quien lo diseñó y a que proyecto está asociado

Constructores:

CAboutDlg();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Constructor por defecto

Datos públicos:

enum { IDD = ID_ABOUTBOX }; El valor entero que le es asociado a ID_ABOUT BOX es asignado a IDD

3.3.2.3 Declaración de la clase "CxResDlg"

class CxResDlg : public CDialog Clase encargada de recibir el nombre del archivo y directorio del programa a ejecutar, así como especificar el MCU a simular

Constructor:

CxResDlg();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Por defecto

Datos públicos:

enum { IDD = ID_SEARCH };		El valor entero que le es asociado a ID_SEARCH es asignado a IDD
CString	**m_arch;**	Cadena que almacena el nombre del archivo
CString	**m_direc;**	Cadena que almacena el nombre del directorio
int	**m_radio;**	Valor entero que asigna 0 al MCU 68705P3, 1 al MCU 68705R3 y 2 al MCU 68705U3

Su función miembro protegida:

virtual void DoDataExchange(CDataExchange* pDX);

Entrada:	Un apuntador al tipo de dato que ha sido modificado
Salida:	Ninguna
Procesamiento:	Actualiza el valor que es introducido en la pantalla a la variable miembro asignada al control y si se modifica la variable miembro asignada a un control actualiza el valor en la pantalla.

3.3.3 Programa CONVERT.CPP

Programa que contiene la funciones de conversión de hexadecimal y binario a entero, y de las hexadecimal a binario y viceversa.

3.3.3.1 Funciones de conversión

int ConvierteHexadecimalaEntero(const char **auxmem*);

Entrada:	Cadena de caracteres que contiene un valor en hexadecimal
Salida:	Un valor entero
Procesamiento:	Convierte la cadena *auxmen* en su equivalente valor entero

CString ConvierteBinarioaHexadecimal(const char *auxmem);

Entrada:	Cadena de caracteres que contiene un valor en binario
Salida:	Cadena de caracteres en hexadecimal
Procesamiento:	Convierte la cadena *auxmem* en su equivalente valor en hexadecimal

CString ConvierteHexadecimalaBinario(const char *auxmem);

Entrada:	Cadena de caracteres que contiene un valor en hexadecimal
Salida:	Cadena de caracteres en binario
Procesamiento:	Convierte la cadena *auxmem* en su equivalente valor en binario

int ConvierteBinarioaEntero(const char *auxmem);

Entrada:	Cadena de caracteres que contiene un valor en binario
Salida:	Un valor entero
Procesamiento:	Convierte la cadena *auxmen* en su equivalente valor entero

3.3.4 Programa HEXA.CPP

Programa que contiene las funciones miembro de la clase Hexadecimal que permite el manejo de operadores sobrecargados para operaciones en binario y hexadecimal.

3.3.4.1 Declaración de la clase "CHexadecimal"

class CHexadecimal: public CObject	Declaración de la clase que maneja datos en hexadecimal y en binario y realiza operaciones básicas

Datos públicos:

char binario[9];	Cadena en binario
char hexadecimal[3];	Cadena en hexadecimal
char c;	Acarreo como un carácter

Constructores:

CHexadecimal();

Entrada:	Ninguna
Salida:	Un apuntador a un tipo de dato CHexadecimal
Procesamiento:	Crea un dato tipo CHexadecimal con un valor en hexadecimal y en su equivalente valor en binario

CHexadecimal(const char **hex*);

Entrada:	Un apuntador a carácter de un valor en hexadecimal
Salida:	Un apuntador a un tipo de dato CHexadecimal
Procesamiento:	Crea un dato tipo CHexadecimal con un valor en hexadecimal y en su equivalente valor en binario

CHexadecimal(CString *hex*);

Entrada:	Una cadena de caracteres de tipo CString de un valor en hexadecimal
Salida:	Un apuntador a un tipo de dato CHexadecimal
Procesamiento:	Crea un dato tipo CHexadecimal con un valor en hexadecimal y en

su equivalente valor en binario

CHexadecimal(CHexadecimal *&b*);

Entrada: Un valor por referencia a un tipo de dato CHexadecimal

Salida: Un apuntador a un tipo de dato CHexadecimal

Procesamiento: Crea un dato tipo CHexadecimal con un valor en hexadecimal y en su equivalente valor en binario

Funciones miembro públicas:

void Iniciadato(const char **hex*)

Entrada: Un apuntador a una cadena de caracteres de un valor hexadecimal

Salida: Ninguna

Procesamiento: Introduce a una variable de tipo CHexadecimal el valor en hexadecimal de un dato y obtiene su correspondiente valor en binario

CHexadecimal& operator=(const CHexadecimal *&rArg*);

Entrada: Un tipo de dato CHexadecimal

Salida: Un tipo de dato CHexadecimal

Procesamiento: Iguala una variable de tipo CHexadecimal a otra del mismo tipo

CHexadecimal operator+(CHexadecimal *&rArg*);

Entrada: Un tipo de dato CHexadecimal
Salida: Un tipo de dato CHexadecimal
Procesamiento: Suma una variable de tipo CHexadecimal a otra del mismo tipo

CHexadecimal operator+(int *rA*);
Entrada: Un tipo de dato CHexadecimal
Salida: Un tipo de dato CHexadecimal
Procesamiento: Suma una variable de tipo CHexadecimal a otra de tipo entero

CHexadecimal operator-(int *rA*);
Entrada: Un tipo de dato CHexadecimal
Salida: Un tipo de dato CHexadecimal
Procesamiento: Resta una variable de tipo CHexadecimal a otra de tipo entero

CHexadecimal operator-(CHexadecimal *&rArg*);
Entrada: Un tipo de dato CHexadecimal
Salida: Un tipo de dato CHexadecimal
Procesamiento: Resta una variable de tipo CHexadecimal a otra de tipo entero

CHexadecimal operator&&(CHexadecimal *&rArg*);
Entrada: Un tipo de dato CHexadecimal
Salida: Un tipo de dato CHexadecimal
Procesamiento: Efectúa la operación lógica AND de una variable de tipo

CHexadecimal a del mismo tipo

CHexadecimal operator||(CHexadecimal &*rArg*);

Entrada: Un tipo de dato CHexadecimal

Salida: Un tipo de dato CHexadecimal

Procesamiento: Efectúa la operación lógica OR de una variable de tipo CHexadecimal a del mismo tipo

CHexadecimal operator^(CHexadecimal &*rArg*);

Entrada: Un tipo de dato CHexadecimal

Salida: Un tipo de dato CHexadecimal

Procesamiento: Efectúa la operación lógica XOR de una variable de tipo CHexadecimal a del mismo tipo

CHexadecimal operator~(void);

Entrada: Ninguna

Salida: Un tipo de dato CHexadecimal

Procesamiento: Efectúa la operación lógica NEGACIÓN de una variable de tipo CHexadecimal a del mismo tipo

3.3.5 Programa INSTRU.H

Programa donde se localiza la definición de la clase CInstruccion

CIII - 264

3.3.5.1 Declaración de la clase "CInstruccion"

class CInstruccion	Crea la clase para almacenar las instrucciones a ejecutar

Datos públicos:

CString PC;	Contiene el contador del programa
CString CodHex;	Contiene el código en hexadecimal
CString Etiqueta;	Contiene la etiqueta
CString Nmonico;	Contiene el Mnemónico
CString Operando;	Contiene el operando
BOOL Betiqueta;	Contiene si hay o no etiqueta

Constructor:

CInstruccion();

Entrada:	Ninguna
Salida:	Apuntador a un tipo de dato CInstrucción
Procesamiento:	Crea un tipo de dato CInstruccion e inicia todos sus datos como se muestra: PC=""; CodHex=""; Etiqueta=""; Nmonico=""; Operando=""; BEtiqueta=FALSE;

Su función miembro:

BOOL IniciaInstruccion(CString PC, CString CH, CString E, CString Nmo, CString Oprn, BOOL BE)

Entrada:	Cadenas correspondientes al contador del programa, código en hexadecimal, etiqueta, mnemónico, operando, y uno para indicar si hay o no etiqueta.
Salida:	Valor lógico que indica que se realizó la asignación
Procesamiento:	Inicia los datos miembro de la clase CInstrucción

3.4 Desarrollo de las rutinas de ambientación y despliegue en Visual C++

3.4.1 Ensamblador

Las rutinas de ambientación son las siguientes:

BOOL Create(LPSTR *szTitle*, int *nChildNum*): Nueva ventana hija
afx_msg void OnClose(): Cierra ventana hija
afx_msg int OnCreate(LPCREATESTRUCT *lpCS*): Crea control de edición
afx_msg void CMGuardar(): Cuadro de diálogo para guardar ventana activa.

afx_msg void CMAbrir():Cuadro de diálogo para abrir ventana activa.
afx_msg int OnCreate(LPCREATESTRUCT *lpCS*): Crea una nueva área de ventana cliente.
afx_msg void CMCreateChild():Crea una ventana hija MDI.
afx_msg void CMCascadeChildren() Pone en cascada a las hijas MDI.
afx_msg void CMTileChildren():Coloca en mosaico a las hijas MDI.
afx_msg void CMArrageIcons():Acomoda los iconos hijos MDI.
afx_msg void CMCloseChildren():Cierra todas las hijas MDI.
afx_msg void OnExit():Maneja la salida de la aplicación.
virtual void OnClose():Maneja el cierre de una ventana marco MDI.

3.4.2 Simulador

El simulador muestra una ventana principal que es la que se muestra en la figura 3.4.2.1.

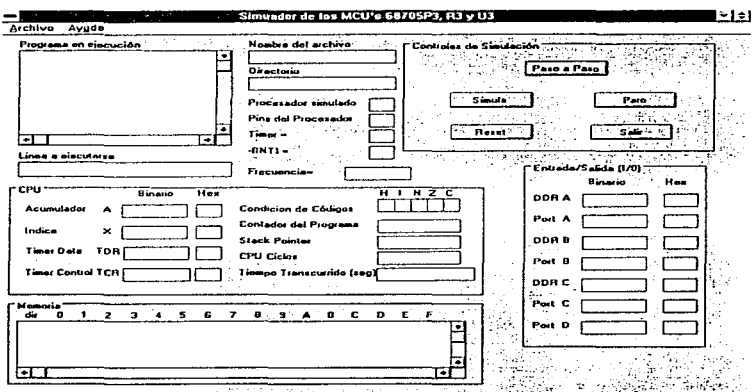


Figura 3.4.2.1 Ventana principal del simulador de los MCU 68705P3, R3 y U3

3.4.2.1 Formato de la ventana principal de ejecución

Programa a ejecutarse

El simulador trabaja con el archivo *.prn* que es creado por el ensamblador y lo carga en una ventana de edición mostrando únicamente las instrucciones a ser ejecutadas en el siguiente formato.

No. línea Etiqueta Mnemónico Operando

CPU

Del CPU muestra los registros internos de los MCU en el siguiente formato.

Nombre del registro Contenido en binario Contenido en hexadecimal

la excepción es el registro de condición de código que tiene el formato

H	I	N	C	Z

Memoria

La memoria de los microcontroladores se muestra con el siguiente formato:

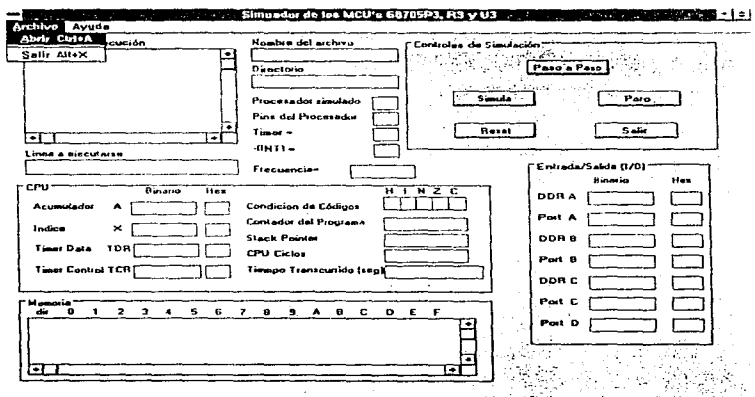


Figura 3.4.2.2a Opciónes del menú. Archivo

Dirección	0	1	2	3	4	5	...	D	E	F
0000:	00	00	00	00	00	00	...	00	00	00
0010:	00	00	00	00	00	00	...	00	00	00
0020:	00	00	00	00	00	00	...	00	00	00

La dirección y sus 16 localidades siguientes.

E/S

Los puertos de E/S y sus DDR se muestran en el siguiente formato:

Nombre del puerto	**Contenido en Binario**	**Contenido en Hexadecimal**
Nombre del DDR	**Contenido en Binario**	**Contenido en Hexadecimal**

3.4.2.2 Funciones para manipulación y actualización de la ventana de CxDialog

virtual void DoDataExchange(CDataExchange* pDX); Explicada anteriormente

afx_msg void CMAbout();
Entrada: Ninguna
Salida: Ninguna
Procesamiento: Ejecuta y muestra la ventana Acerca de..

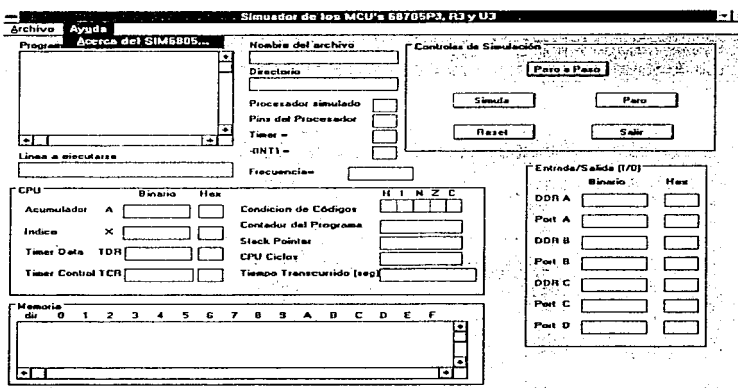


Figura 3.4.2.2b Opciónes del menú. Ayuda

afx_msg void CMDialog();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Ejecuta y muestra la ventana principal

afx_msg void OnExit();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Manda un mensaje si se desea salir de la aplicación

afx_msg void OnClose();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Manda un mensaje si se desea salir de la aplicación

afx_msg void OnClickedSalir();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Es llamada cuando se oprime el botón "Salir" y se llama a la función OnClose

afx_msg void OnClickedSimula();

Entrada: Ninguna

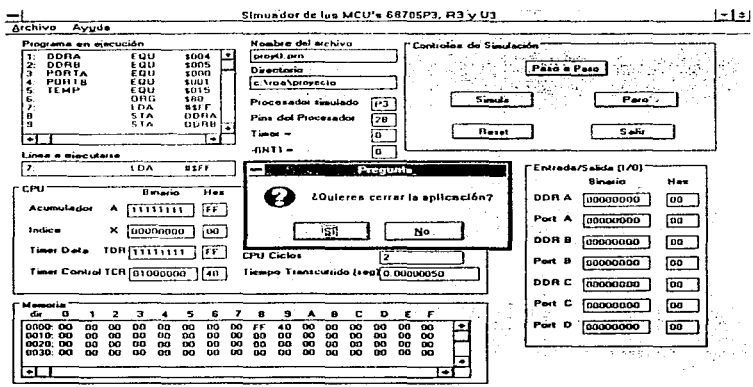


Figura 3.4.2.2c Botón Salir

Salida: Ninguna

Procesamiento: Ejecuta la simulación de las instrucciones de forma continua hasta que el botón "Paro" detiene la simulación o hasta que se cumpla el valor máximo de simular que es de 1500 instrucciones

afx_msg void OnClickedReset();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Inicializa todo el simulador

afx_msg void OnClickedPasoapaso();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Ejecuta cada una de las instrucciones paso a paso

afx_msg void OnChangeMem();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor de la memoria en pantalla y estos se efectúan en la página 0, explícitamente en las direcciones de puertos o DDR se actualizan los valores de estos que también son mostrados en otras secciones del simulador

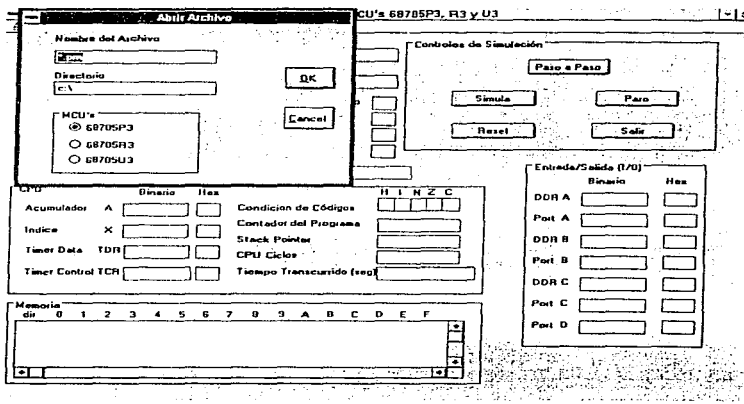


Figura 3.4.2.3a Opciónes del menú. Abrir

afx_msg void OnChangeDDRAb();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRAb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangeDDRAh();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRAh en pantalla se actualiza su valor en binario y su valor en memoria

afx_msg void OnChangeDDRBb();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRBb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

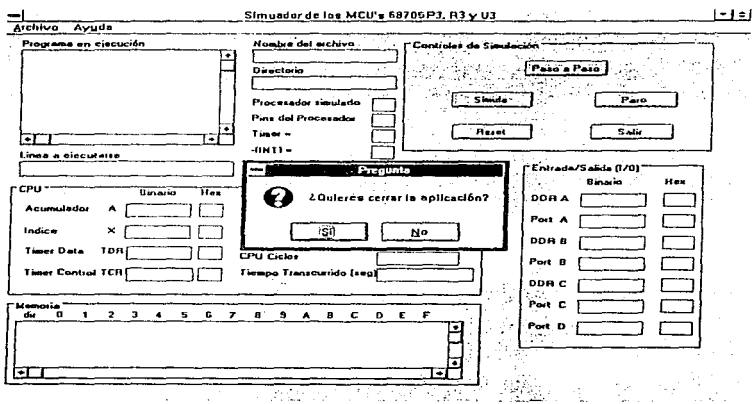


Figura 3.4.2.3b Opciónes del menú. Salir

afx_msg void OnChangeDDRBh();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRBh en pantalla se actualiza su valor en binario y su valor en memoria

afx_msg void OnChangeDDRCb();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRCb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangeDDRCh();

Entrada:	Ninguna
Salida:	Ninguna
Procesamiento:	Si se modifica algún valor del DDRCh en pantalla se actualiza su valor en binario y su valor en memoria

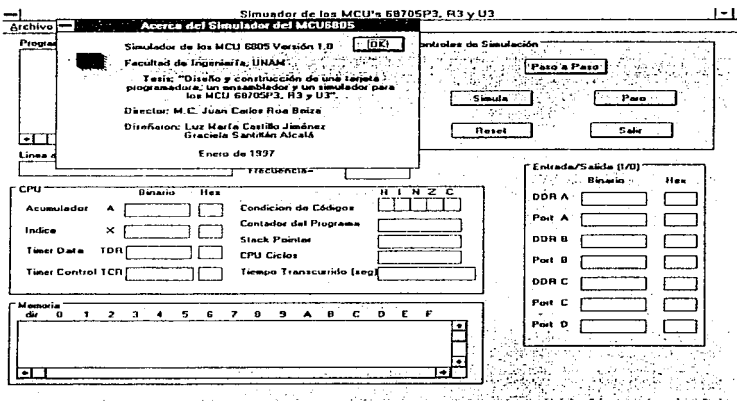


Figura 3.4.2.3c Opciónes del menú. Acerca del Simulador...

afx_msg void OnChangePORTAb();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTAb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangePORTAh();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTAh en pantalla se actualiza su valor en binario y su valor en memoria

afx_msg void OnChangePORTBb();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTBb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangePORTBh();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTBh en pantalla se actualiza su valor en binario y su valor en memoria

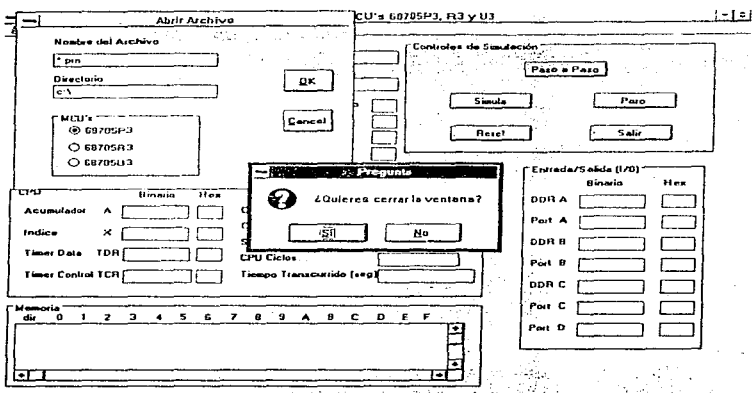


Figura 3.4.2.3d Opciónes del menú. Abrir archivo. Botón Cancel

afx_msg void OnChangePORTCb();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTCb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangePORTCh();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTCh en pantalla se actualiza su valor en binario y su valor en memoria

afx_msg void OnChangePORTDb();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTDb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangePORTDh();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del PORTDh en pantalla se actualiza su

valor en binario y su valor en memoria

afx_msg void OnChangeAcuab();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del ACCAb en pantalla se actualiza su valor en hexadecimal

afx_msg void OnChangeAcuah();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del ACCAh en pantalla se actualiza su valor en binario

afx_msg void OnChangeIndxb();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del INDXb en pantalla se actualiza su valor en hexadecimal y su valor en memoria

afx_msg void OnChangeIndxh();

Entrada: **Ninguna**

Salida: **Ninguna**

Procesamiento: Si se modifica algún valor del INDXh en pantalla se actualiza su valor en binario

afx_msg void OnChangeCp();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica algún valor del contador del programa en pantalla se efectúa el salto hasta que el PC de la instrucción corresponda a la del PC indicado si no es así no salta.

afx_msg void OnChangeC();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor en el dato miembro m_C

afx_msg void OnChangeH();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor en el dato miembro m_H

afx_msg void OnChangeI();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor en el dato miembro m_I

afx_msg void OnChangeN();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor en el dato miembro m_N

afx_msg void OnChangeZ();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor en el dato miembro m_Z

afx_msg void OnTimer(UINT *nIDEvent*);

Entrada: El valor entero del identificador del evento a ejecutarse

Salida: Ninguna

Procesamiento: Manipula el timer interno de la computadora para que se ejecuten las instrucciones en un ciclo de 10 ms cada una

afx_msg void OnClickedPara();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Detiene la ejecución de simula

afx_msg void OnChangeTimer();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Actualiza el valor de m_timer y ejecuta el servicio de interrupción por timer

afx_msg void OnChangeInt();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Si se modifica el valor de -(INT) en pantalla, actualiza el valor de la variable m_INT

3.4.2.2 Funciones para manipulación y actualización de la ventana de CxResDlg

virtual void OnCancel();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Cancela la operación y pregunta si se cierra la ventana

virtual void OnOK();

Entrada: Ninguna

Salida: Ninguna

Procesamiento: Afirma la operación y actualiza los valores de las variables m_arch, m_direc y m_radio

3.4.2.3 Programa SIM6805.RC

Este programa es creado por Microsoft App Studio que genera los recursos de pantalla. Cuando uno crea cualquiera de los recursos como son el menú, teclas aceleradoras, cuadros de dialogo, o elementos que vienen en App Studio, automáticamente crea este programa con extensión RC del programa principal, uno también puede crearlo y visualizarlo en la ventana de App Studio sin ningún problema, la asociación de las variables con los recursos del diálogo se realiza por medio de Class Wizard que es con el que se puede crear las funciones para manipulación de todos los recursos que podamos añadir.

Declaración de un ícono

```
6805            ICON    DISCARDABLE    "RES\\6805.ICO"
```

Declaración del cuadro de diálogo para recibir el nombre del archivo, el directorio y el tipo de MCU (ver figura 3.4.2.3a)

```
ID_SEARCH DIALOG DISCARDABLE 10, 10, 201, 150
STYLE DS_SYSMODAL | DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
WS_CLIPSIBLINGS | WS_CAPTION | WS_SYSMENU
CAPTION "Abrir Archivo"
FONT 8, "MS Sans Serif"
BEGIN
 LTEXT        "Nombre del Archivo",ID_ARCHIVO_TXT,20,10,100,10,NOT
              WS_GROUP
 EDITTEXT     ID_ARCHIVO_EDIT,20,25,100,12,ES_LOWERCASE
 LTEXT        "Directorio",ID_DIRECTORIO_TXT,20,45,100,8,NOT WS_GROUP
 EDITTEXT     ID_DIRECTORIO_EDIT,20,55,100,12,ES_LOWERCASE
 GROUPBOX     "MCU's",ID_MCU_GRP,20,80,90,60,WS_GROUP
 CONTROL      "68705P3",ID_P3_RBT,"Button",BS_AUTORADIOBUTTON |
              WS_GROUP | WS_TABSTOP,30,90,60,15
 CONTROL      "68705R3",ID_R3_BTN,"Button",BS_AUTORADIOBUTTON|
              WS_TABSTOP,30,106,60,15
 CONTROL      "68705U3",ID_U3_RBT,"Button",BS_AUTORADIOBUTTON|
```

```
                    WS_TABSTOP,30.120,60,15
  DEFPUSHBUTTON     "&OK",IDOK,161,41,30,20
  PUSHBUTTON        "&Cancel",IDCANCEL,161,79,30,20,WS_GROUP
END
```

Declaración del cuadro de diálogo Acerca de... (ver figura 3.4.2.3c)

```
ID_ABOUTBOX DIALOG DISCARDABLE 34, 22, 221, 128
STYLE DS_SYSMODAL | WS_POPUP | WS_VISIBLE | WS_CLIPSIBLINGS |
WS_CAPTION | WS_SYSMENU
CAPTION "Acerca del Simulador del MCU6805"
FONT 8, "MS Sans Serif"
BEGIN
  ICON      6805,IDC_STATIC,11,17,18,20
  LTEXT     "Simulador de los MCU 6805 Versi\363n 1.0",IDC_STATIC,40,10,130,8
  LTEXT     "Facultad de Ingenier\355a, UNAM",IDC_STATIC,40,25,119,8
  DEFPUSHBUTTON     "OK",IDOK,180,5,32,14,WS_GROUP
  CTEXT     "Tesis: ""Diseño y construcción de una tarjeta programadora, un ensamblador
            y un simulador para los MCU 68705P3, R3 y U3"".",
            IDC_STATIC,40,40,170,25
  LTEXT     "Director: M.C. Juan Carlos Roa Beiza",IDC_STATIC,40,70,154,7
  LTEXT     "Diseñaron: Luz María Castillo Jiménez          Graciela Santillán Alcalá",
            IDC_STATIC,40,85,130,15
  LTEXT     "Enero de 1997",IDC_STATIC,85,110,50,7
```

```
END
```

Declaración del cuadro de diálogo que maneja el simulador (ver figura 3.4.2.1)

```
ID_DIALOG1 DIALOG DISCARDABLE  0, 0, 447, 342
STYLE WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU | WS_THICKFRAME
CAPTION "Simuador de los MCU's 68705P3, R3 y U3"
MENU ID_MAINMENU
FONT 8, "MS Sans Serif"
BEGIN
  GROUPBOX      "Memoria",IDC_STATIC,5,250,290,80,WS_TABSTOP
  LTEXT         "0",IDC_STATIC,35,260,8,7
  LTEXT         "1",IDC_STATIC,50,260,8,7
  LTEXT         "2",IDC_STATIC,65,260,8,7
  LTEXT         "3",IDC_STATIC,80,260,8,7
  LTEXT         "4",IDC_STATIC,95,260,8,7
  LTEXT         "5",IDC_STATIC,110,260,8,7
  LTEXT         "6",IDC_STATIC,125,260,8,7
  LTEXT         "7",IDC_STATIC,140,260,8,7
  LTEXT         "8",IDC_STATIC,155,260,8,7
  LTEXT         "9",IDC_STATIC,170,260,8,7
  LTEXT         "A",IDC_STATIC,185,260,8,7
  LTEXT         "B",IDC_STATIC,200,260,8,7
  LTEXT         "C",IDC_STATIC,215,260,8,7
```

```
LTEXT       "D",IDC_STATIC,230,260,8,7
LTEXT       "E",IDC_STATIC,245,260,8,7
LTEXT       "F",IDC_STATIC,260,260,8,7
GROUPBOX    "CPU",IDC_STATIC,5,140,290,105
EDITTEXT    IDC_ACUAB,75,160,40,12,ES_UPPERCASE|ES_AUTOHSCROLL
EDITTEXT    IDC_INDXB,75,180,40,12,ES_UPPERCASE | ES_AUTOHSCROLL
EDITTEXT    IDC_TDRB,76,200,40,12,ES_UPPERCASE | ES_AUTOHSCROLL
            | ES_READONLY
EDITTEXT    IDC_TCRB,76,220,41,12,ES_UPPERCASE | ES_AUTOHSCROLL|
            ES_READONLY
LTEXT       "Acumulador   A",IDC_STATIC,15,160,60,7
LTEXT       "Indice              X",IDC_STATIC,14,181,60,7
EDITTEXT    IDC_ACUAH,120,160,15,12,ES_UPPERCASE |
            ES_AUTOHSCROLL
LTEXT       "Timer Data   TDR",IDC_STATIC,15,200,60,7
LTEXT       "Timer Control TCR",IDC_STATIC,15,220,60,7
CTEXT       "Binario",IDC_STATIC,80,145,30,9
EDITTEXT    IDC_INDXH,120,180,15,12,ES_UPPERCASE|ES_AUTOHSCROLL
EDITTEXT    IDC_TDRH,120,200,15,12,ES_UPPERCASE | ES_AUTOHSCROLL
            |ES_READONLY
EDITTEXT    IDC_TCRH,120,220,15,12,ES_UPPERCASE | ES_AUTOHSCROLL
            | ES_READONLY
CTEXT       "Hex",IDC_STATIC,120,145,15,7,NOT WS_GROUP
LTEXT       "Condicion de C\363digos",IDC_STATIC,145,160,75,10
```

```
LTEXT       "Contador del Programa",IDC_STATIC,145,175,75,10
LTEXT       "Stack Pointer",IDC_STATIC,145,190,75,7
LTEXT       "CPU Ciclos",IDC_STATIC,145,205,75,7
LTEXT       "Tiempo Transcurrido (seg)",IDC_STATIC,145,220,85,10
EDITTEXT    IDC_CP,230,175,51,12,ES_UPPERCASE | ES_AUTOHSCROLL
EDITTEXT    IDC_SP,230,190,51,12,ES_UPPERCASE | ES_AUTOHSCROLL |
            ES_READONLY
EDITTEXT    IDC_CICLOS,230,205,51,12,ES_AUTOHSCROLL |
            ES_READONLY
EDITTEXT    IDC_SEG,230,220,60,12,ES_AUTOHSCROLL | ES_READONLY
EDITTEXT    IDC_H,230,155,10,12,ES_AUTOHSCROLL
EDITTEXT    IDC_I,240,155,10,12,ES_AUTOHSCROLL
EDITTEXT    IDC_N,250,155,10,12,ES_AUTOHSCROLL
EDITTEXT    IDC_Z,260,155,10,12,ES_AUTOHSCROLL
EDITTEXT    IDC_C,270,155,10,12,ES_AUTOHSCROLL
CTEXT        "H",IDC_STATIC,230,145,8,7
CTEXT        "I",IDC_STATIC,240,145,8,7
CTEXT        "N",IDC_STATIC,250,145,8,7
CTEXT        "Z",IDC_STATIC,260,145,8,7
CTEXT        "C",IDC_STATIC,270,145,8,7
GROUPBOX    "Entrada/Salida (I/0)",IDC_STATIC,320,120,110,210
LTEXT        "DDR A",IDC_STATIC,325,150,25,7
LTEXT        "DDR B",IDC_STATIC,325,190,25,7
LTEXT        "Port  A",IDC_STATIC,325,170,25,7
```

```
LTEXT          "Port B",IDC_STATIC,325,210,24,7
CTEXT          "Binario",IDC_STATIC,355,135,30,9
EDITTEXT       IDC_DDRAB,355,150,40,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_PORTAB,355,170,40,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_DDRBB,355,190,40,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_PORTBB,355,210,40,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
CTEXT          "Hex",IDC_STATIC,405,135,15,7,NOT WS_GROUP
EDITTEXT       IDC_DDRAH,405,150,20,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_PORTAH,405,170,20,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_DDRBH,405,190,20,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
EDITTEXT       IDC_PORTBH,405,210,20,12,ES_UPPERCASE |
               ES_AUTOHSCROLL
LTEXT          "DDR C",IDC_STATIC,325,230,25,8
LTEXT          "Port C",IDC_STATIC,325,250,25,7
LTEXT          "Port D",IDC_STATIC,325,290,25,7
LTEXT          "DDR D",IDC_STATIC,325,270,25,8
EDITTEXT       IDC_DDRCB,355,230,40,12,ES_UPPERCASE |
```

```
                ES_AUTOHSCROLL
EDITTEXT        IDC_PORTDB,355,290,40,12,ES_UPPERCASE |
                ES_AUTOHSCROLL
EDITTEXT        IDC_PORTCB,355,250,40,12,ES_UPPERCASE |
                ES_AUTOHSCROLL
LTEXT           "Pins del Procesador",IDC_STATIC,150,75,65,7
LTEXT           "Timer =",IDC_STATIC,150,90,30,7
LTEXT           "-(INT) =",IDC_STATIC,150,105,30,7
EDITTEXT        IDC_TIMER,225,90,15,12,ES_AUTOHSCROLL
EDITTEXT        IDC_INT,225,105,15,12,ES_AUTOHSCROLL
EDITTEXT        IDC_PINS,225,75,15,12,ES_AUTOHSCROLL | ES_READONLY
EDITTEXT        IDC_DDRCH,405,230,20,12,ES_UPPERCASE |
                ES_AUTOHSCROLL
EDITTEXT        IDC_PORTDH,405,290,20,12,ES_UPPERCASE |
                ES_AUTOHSCROLL
EDITTEXT        IDC_PORTCH,405,250,20,12,ES_UPPERCASE |
                ES_AUTOHSCROLL
DEFPUSHBUTTON   "Paso a Paso",IDC_PASOAPASO,320,25,50,14
PUSHBUTTON      "Simula",IDC_SIMULA,274,55,51,14
PUSHBUTTON      "Reset",IDC_RESET,275,85,50,14
```

```
GROUPBOX    "Controles de Simulaci\363n",IDC_STATIC,245,5,190,105,
            WS_TABSTOP
EDITTEXT    IDC_ARCHIVO,150,15,90,12,ES_LOWERCASE |
            ES_AUTOHSCROLL | ES_READONLY
EDITTEXT    IDC_DIRECTORIO,150,40,90,12,ES_LOWERCASE |
            ES_AUTOHSCROLL | ES_READONLY
LTEXT       "Nombre del archivo",IDC_STATIC,150,5,65,7
LTEXT       "Directorio",IDC_STATIC,150,30,35,7
LTEXT       "Procesador simulado",IDC_STATIC,150,60,70,7
EDITTEXT    IDC_PROC,225,60,15,12,ES_UPPERCASE | ES_AUTOHSCROLL
            | ES_READONLY
EDITTEXT    IDC_PROGRAMA,10,15,130,90,ES_MULTILINE | ES_UPPERCASE
            | ES_AUTOVSCROLL | ES_AUTOHSCROLL | ES_READONLY |
                  WS_VSCROLL | WS_HSCROLL
EDITTEXT    IDC_LINEAPROG,10,120,130,14,ES_MULTILINE |
            ES_UPPERCASE | ES_READONLY
LTEXT       "Linea a ejecutarse",IDC_STATIC,10,110,105,7
LTEXT       "Programa en ejecuci\363n",IDC_STATIC,10,5,105,10
PUSHBUTTON  "Salir",IDC_SALIR,360,85,50,14
EDITTEXT    IDC_MEM,10,270,275,54,ES_MULTILINE | ES_UPPERCASE |
            ES_AUTOVSCROLL | ES_AUTOHSCROLL | WS_VSCROLL |
            WS_HSCROLL
LTEXT       "dir",IDC_STATIC,15,260,14,7
LTEXT       "Frecuencia=",IDC_STATIC,150,125,50,7
```

```
  EDITTEXT        IDC_FREC,210,125,40,12,ES_AUTOHSCROLL
  PUSHBUTTON    "Paro",IDC_PARA,362,55,50,14
END
```

Declaración del menú (ver figuras 3.4.2.2a, 3.4.2.2b)

```
ID_MAINMENU MENU DISCARDABLE
BEGIN
  POPUP "&Archivo"
  BEGIN
      MENUITEM "&Abrir\tCtrl+A",     CM_DIALOG
      MENUITEM SEPARATOR
      MENUITEM "&Salir\tAlt+X",      CM_EXIT
  END
  POPUP "Ay&uda"
  BEGIN
      MENUITEM "&Acerca del SIM6805...",    CM_ABOUT
  END
END
```

Declaración del los mensajes de usuario (ver figuras 3.4.2.3b y 3.4.2.3d)

Es la función para crear mensajes pequeños personalizados, en este caso de pregunta para cerrar la aplicación o una ventana.

```
MessageBox("¿Quieres cerrar la aplicación?","Pregunta",MB_YESNO|
MB_ICONQUESTION)

MessageBox("¿Quieres cerrar la ventana?", "Pregunta", MB_YESNO|
MB_ICONQUESTION)
```

3.5 Integración del sistema de software y evaluación de éste.

Se realizó un programa de prueba que es el siguiente:

Programa de prueba: PROY0

```
;************************************************
;* ESTE PROGRAMA ES UN EJEMPLO PARA DEMOSTRAR EL *
;* USO DE LOS PUERTOS A y B ; A TRAVES DEL ENVIO *
;* DE UNA SECUENCIA DE BITS CON UN RETARDO DE UN *
;* SEGUNDO PARA QUE EL OPERADOR PUEDA OBSERVARLOS *
;* ADECUADAMENTE, EN UN CORRIMIENTO DE IZQUIERDA A *
;* DERECHA.      " PRUEBA0 "          *
;************************************************
DDRA      EQU $004    ;DIRECCION DDRA
DDRB      EQU $005    ;DIRECCION DDRB
PORTA     EQU $000    ;DIRECCION PORTA
PORTB     EQU $001    ;DIRECCION PORTB
```

```
TEMP    EQU $015    ;REGISTRO TEMPORAL
        ORG $80     ;ORIGEN DEL PROGRMA
        LDA #$FF    ;PROGRAMACION DE LOS
        STA DDRA    ;PUERTOS COMO SALIDAS
        STA DDRB    ;CON UN $ FF
GO      LDA #$01    ;DATO DE INICIO
ZAPO    STA PORTA   ;ENVIO A PUERTO A
        STA PORTB   ;ENVIO A PUERTO B
        JSR DELAY   ;RETARDO DE 1 SEG
        ROLA        ;RECORRE BIT DE DATO
        BCC ZAPO    ;REINICIA SECUENCIA
        CLC         ;SI EL C=0
        BRA GO      ;VUELVE A INICIAR TODO
DELAY   STA TEMP    ;GUARDA EL CONTENIDO DE A
        LDA #$00    ;INICIA CONTADOR EN CERO
OTRO    LDX #$10    ;CARGA X CON $ FF
RANA    DECX        ;DECREMENTA X
        NOP         ;PIERDE TIEMPO
        NOP         ;PIERDE TIEMPO
        CPX #$00    ;COMPARA (X) CON CERO
        BNE RANA    ;SI NO SON IGUALES VE A RANA
        INCA        ;SI SON IGUALES (A)=(A)+1
        CMP #$FF    ;(A)=$FF
        BNE OTRO    ;SI NO SON IGUALES VE A OTRO
        LDA TEMP    ;RESCATA EL VALOR DEL A
        RTS         ;REGRESA DE LA SUBRUTINA
        ORG $784    ;DIRECCION DEL  MOR
```

```
DB  $08      ;DATOS PARA EL MOR
ORG $7FE     ;DIRECCION DEL RESET
DW  $0080    ;DIRECCION DE INICIO DEL PROGRAMA
END          ;FIN
```

Cuya salida del ensamblador es:

PROY0. PRN

```
;************************************************
;* ESTE PROGRAMA ES UN EJEMPLO PARA DEMOSTRAR EL *
;* USO DE LOS PUERTOS  A y B : A TRAVES DEL ENVIO *
;* DE UNA SECUENCIA DE BIT'S CON UN RETARDO DE UN *
;* SEGUNDO PARA QUE EL OPERADOR PUEDA OBSERVARLOS *
;* ADECUADAMENTE, EN UN CORRMIENTO DE IZQUIERDA A *
;* DERECHA.      " PRUEBA0 "              *
;************************************************
= 0004    DDRA    EQU $004    ;DIRECCION DDRA
= 0005    DDRB    EQU $005    ;DIRECCION DDRB
= 0000    PORTA   EQU $000    ;DIRECCION PORTA
= 0001    PORTB   EQU $001    ;DIRECCION PORTB
= 0015    TEMP    EQU $015    ;REGISTRO TEMPORAL
0080              ORG $80     ;ORIGEN DEL PROGRMA
0080 A6FF         LDA #$FF    ;PROGRAMACION DE LOS
0082 B704         STA DDRA    ;PUERTOS COMO SALIDAS
0084 B705         STA DDRB    ;CON UN $ FF
0086 A601   GO    LDA #$01    ;DATO DE INICIO
```

```
0088 B700     ZAPO    STA PORTA  ;ENVIO A PUERTO A
008A B701             STA PORTB  ;ENVIO A PUERTO B
008C CD0095           JSR DELAY  ;RETARDO DE 1 SEG.
008F 49               ROLA ;     RECORRE BIT DE DATO
0090 24F6             BCC ZAPO   ;REINICIA SECUENCIA
0092 98               CLC        ;SI EL C=0
0093 20F1             BRA GO     ;VUELVE A INICIAR TODO
0095 B715     DELAY   STA TEMP   ;GUARDA EL CONTENIDO DE A
0097 A600             LDA #$00   ;INICIA CONTADOR EN CERO
0099 AE10     OTRO    LDX #$10   ;CARGA X CON $ FF
009B 5A       RANA    DECX       ;DECREMENTA X
009C 9D               NOP        ;PIERDE TIEMPO
009D 9D               NOP        ;PIERDE TIEMPO
009E A300             CPX #$00   ;COMPARA (X) CON CERO
00A0 26F9             BNE RANA   ;SI NO SON IGUALES VE A RANA
00A2 4C               INCA  ;SI SON IGUALES (A)=(A)+1
00A3 A1FF             CMP #$FF   ;(A)=$FF
00A5 26F2             BNE OTRO   ;SI NO SON IGUALES VE A OTRO
00A7 B615             LDA TEMP   ;RESCATA EL VALOR DEL A
00A9 81               RTS        ;REGRESA DE LA SUBRUTINA
0784                  ORG $784   ;DIRECCION DEL MOR
0784 08               DB  $08    ;DATOS PARA EL MOR
07FE                  ORG $7FE   ;DIRECCION DEL RESET
07FE 0080             DW  $0080  ;DIRECCION DE INICIO DEL PROGRAMA
0000                  END        ;FIN
```

---Tabla de Simbolos---

DDRA	0004
DDRB	0005
DELAY	0095
GO	0086
OTRO	0099
PORTA	0000
PORTB	0001
RANA	009B
TEMP	0015
ZAPO	0088

PROY0.MIK

0080A6FFB704B705A601B700B701CD009549
009024F69820F1B715A600AE105A9D9DA300
00A026F94CA1FF26F2B61581
078408
07FE0080
0000

Un ejemplo de la corrida del simulador es la que se muestra en la figura 3.5

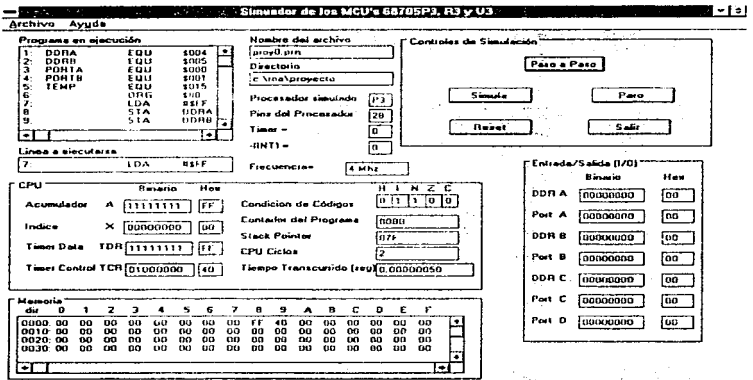
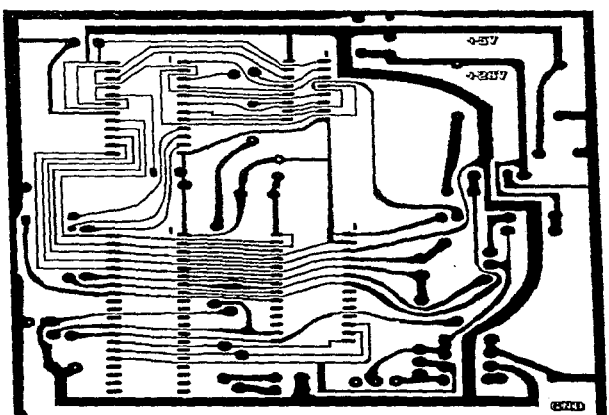
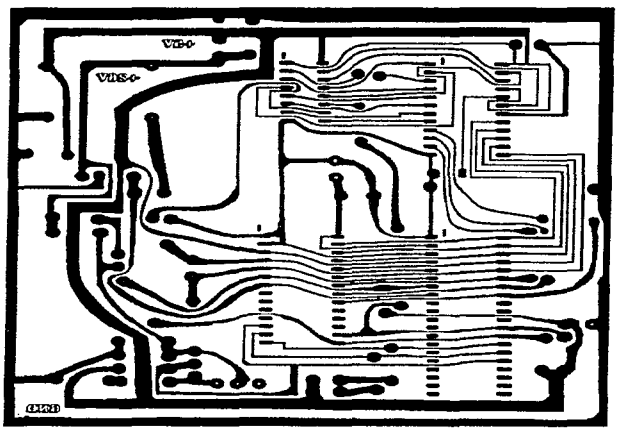


Figura 3.5 Ejemplo simulando el programa PROY0

3.6 Diseño y ensamblado de la tarjeta programadora y evaluación de ésta.

Figuras 3.6.1 y 3.6.2 Frente y vuelta del impreso de la tarjeta programadora





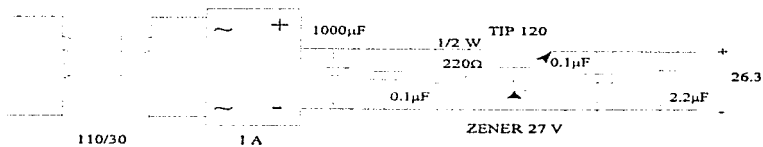
Arme la tarjeta programadora con los componentes mencionados en el capítulo anterior.

NOTA:

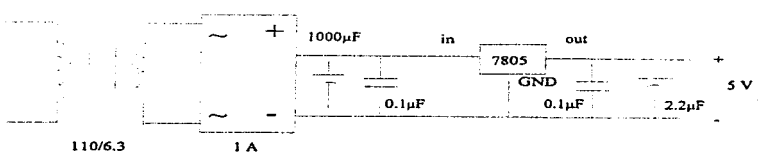
Cuando programe los MCU inserte uno a la vez para evitar daños a la tarjeta programadora y la pérdida de ésta.

En caso de no encontrar el convertidor 26A05 de Astec, arme un circuito de

voltaje equivalente o el proporcionado a continuación.



Todos los capacitores a 50V
Todas las resistencias a 1/2 W
Puente rectificador 1 A, 250 V
Transformador 110/30 V



Conclusiones

Conclusiones

Sabemos la utilidad que tienen los MCU en la industria, como se comentó en el capitulo 1 en historia de los microcontroladores, es por eso que en la Universidad Nacional Autónoma de México, y más explícitamente en la Facultad de Ingeniería se imparte la materia de Microprocesadores, para que los alumnos aprendan como utilizarlos, es por eso que quisimos desarrollar este tema de tesis, para proporcionar una herramienta más en el aprendizaje de los MCU, y por lo mismo escogimos los 68705P3, R3 y U3 por pequeños y económicos.

Sinceramente no sabíamos las dificultades que se presentarían al desarrollar el simulador y el ensamblador en el paquete Visual C++ ya que tuvimos que aprender a programar en la técnica de la POO y sobre el manejo de rutinas para programar en Windows, fue un gran esfuerzo ya que leímos 4 libros para el aprendizaje de la POO, además de la lectura para aprender como funcionaban internamente los MCU, y por ello estamos seguras de lo que aplicamos, pero sabemos que es una área muy grande y que no la abarcamos en toda su extensión, pero los elementos básicos podemos decir que los dominamos. Esto nos ha proporcionado una gran preparación tanto a nivel

escolar como profesional, ya que en nuestros trabajos, actualmente nos ha abierto una gran cantidad de expectativas y oportunidades de desarrollo, porque nos hemos dado cuenta de que en la industria las aplicaciones basadas en Windows tienen un gran peso, por ejemplo hay simuladores de generadores de electricidad (termoeléctricas, nucleoeléctricas) que funcionan en dicho ambiente, y otras en aplicaciones de Windows en tiempo real, que es otra área de aplicación, ya que la simulación es efectuada con un procesamiento real y con diferenciales de tiempo de milisegundos; una de nosotros está desarrollando una interfaz en tiempo real, para simuladores de reguladores de tensión automática, en ambiente QNX Windows (Windows trabajando en tiempo real) y en el sistema operativo QNX, también de tiempo real.

Otros puntos importantes son:

- Visual C++ proporciona herramientas muy amplias y poderosas, es por ello que permite realizar programas de aplicación más completos y complejos.
- Los formatos de arrastrar y pegar permite la realización de interfaces gráficas de usuario fáciles de realizar y de trabajar.
- El manejo de memoria sale fuera de lo convencional causando un poco de problemas por el manejo de la misma, debido a que solamente tenemos un restringido número

de líneas tanto para el ensamblador como para el simulador, se convirtieron en dos aplicaciones de gran tamaño y creemos que por ello y por la versión que manejamos de Visual C++ v.1 es por lo que se presentaron estos inconvenientes. Observamos que para aumentar el número de las líneas de instrucciones a ensamblar y a simular había que modificar el valor del parámetro "stack" (en la opción de "projec options" de "linker" y "memory image") a su mínimo valor que es 2024.

- La biblioteca de clases de Microsoft Fundation Class contiene clases ya definidas que permiten la creación de nuevas clases para el mejor manejo de los recursos.
- La forma de flujo de la información es muy diferente a la programación estructurada ya que en este caso el programador no introduce un main como programa principal como tal, sino más bien sólamente crea un objeto de tipo aplicación que es el que ejecuta todo.
- Curva de aprendizaje. Comprobamos la dificultad de aprender esta técnica de programación pero vemos los resultados de programación son más óptimos que con la programación tradicional y con las buenas bases que se tengan de la programación estructurada se ayuda a mejorar el aprendizaje de la programación orientada a objetos.
- El código de la P.O.O puede reutilizarse más fácilmente ya que se pueden crear

librerías de clases e ir agregando las características de los objetos, de herencia y polimorfismo, los objetos resultantes pueden con el límite de la imaginación.

- El aumento de las funciones en C aprobadas por el comite de estándares ANSI y las que se pueden expander a otros tipos de sistemas operativos aumenta la productividad del programador y la portabilidad de las aplicaciones que se desarrollen.
- La programación orientada a objetos permite una programación más estructurada por ser esta más formal en su manejo de clases y funciones.
- Si el programación no posee los conocimientos básicos de la POO será más difícil su aprendizaje de Visual C++ y el tiempo que le tome en aprenderlo se incrementará sustancialmente.
- El mantenimiento de los programas escritos en la POO es más fácil y así si pasa a otras manos es de fácil entendimiento.
- Varias empresas están cambiando a la POO, aunque como sabemos el miedo al cambio hace que muchas de ellas ni siquiera consideren esta opción, debido a que los programadores se enfocan a resolver el problema y no analizarlo, siendo una opción muy importante la POO.

Apéndice A

Manual Técnico de los MCU MC68705P3, R3 y U3

MC68705P3

Microcontrolador de 8 bits con EPROM[1]

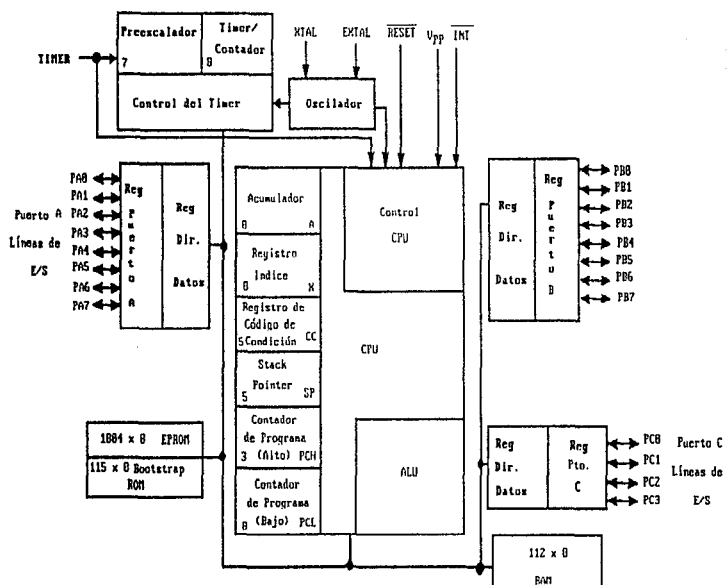
La unidad microcomputadora (MCU) MC68705P3 (de alta densidad NMOS) es una EPROM miembro de la familia de microcomputadoras MC6805. La EPROM programable permite cambiar programas y realizar aplicaciones de bajo volumen. Esta MCU es de bajo costo, tiene capacidad de E/S paralela con pines programables como entrada o salida.

A continuación se muestra un diagrama de bloques de las características de hardware y una lista de características adicionales del MCU.

- Timer interno de 8 bits con 7 bits preescalables programables.
- Oscilador on-chip
- Mapa de memoria de E/S
- Manejador de interrupciones versátil
- Manipulación de bit
- 20 líneas TTL/CMOS de E/S bidireccionales compatibles
- Verificador de bit e instrucciones de bifurcación
- Vectores de interrupción
- Programa bootstrap (arranque) en ROM
- 1804 bytes de EPROM
- 112 bytes en RAM

[1]Traducción y resumen realizado por Luz María Castillo Jiménez y Graciela Santillán Alcalá del manual "Motorola Semiconductor Technical Data, Technical Summary"

DIAGRAMA DE BLOQUES DEL MCU MC68705P3



Descripción de señales

V_{CC} y V_{SS}

Utilizando estos dos pines se suministra potencia al microcomputador. V_{CC} es +5.25 volts (±0.5 %) y V_{SS} es tierra.

V_{PP}

Este pin se utiliza cuando se programa la EPROM. En operación normal este pin se conecta a V_{CC}.

$\overline{INT}$

Este pin permite realizar una interrupción asíncrona externa al MCU.

EXTAL, XTAL

Estos pines proveen el control de entrada para el circuito oscilador de reloj del chip. Una combinación de un cristal y una resistencia/capacitor, o una señal externa se conecta a estos pines para proveer un sistema de reloj. La selección se hace

mediante el bit CLK en el registro de opción enmascarable.

Oscilador RC Con esta opción, se conecta una resistencia al pin del oscilador como se muestra el la figura 1. La relación entre R y f_{osc} se muestra en la figura 2.

Cristal El circuito mostrado en la figura se recomienda cuando se usa un cristal. El uso de un oscilador CMOS externo se recomienda cuando los cristales dan como salidas rangos específicos. El cristal y los componentes deben montarse en un espacio tan pequeño y cercano como sea posible a los pines de entrada para evitar la distorsión de salida y comenzar la estabilización del tiempo.

Reloj Externo Puede aplicarse un reloj externo a la entrada **EXTAL** con la entrada **XTAL** conectada a tierra como se muestra en la figura 1. Esta opción sólo puede utilizarse con la opción de oscilador de cristal seleccionada en el registro de opción enmascarada.

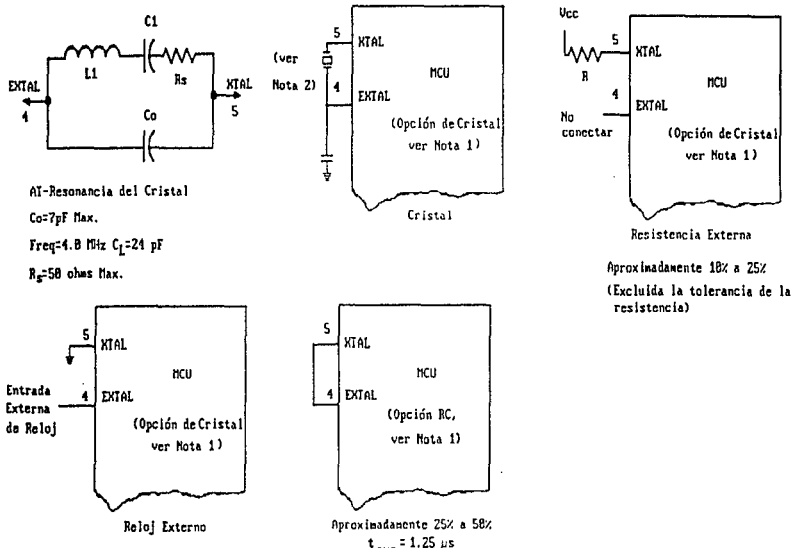


Figura 1. Conexiones del Oscilador

Notas:

1. Cuando el pin de entrada del TIMER está en el rango V_{HTP} (en el modo de programación bootstrap de la EPROM), se forza la opción del cristal.
2. El valor que se recomienda para C_L con un cristal de 4.0 Mhz es de 27 pF como máximo incluyendo la capacitancia del sistema de distribución. Esta es una capacitancia interna en el pin de XTAL de 25 pF aproximadamente. Para cristales de frecuencias diferentes a 4MHz, la capacitancia total en cada pin debe escalarse como el inverso del radio de frecuencia. Por ejemplo, con un cristal de 2 Mhz, se utilizan aproximadamente 50 pF en EXTAL y aproximadamente 25 pF en XTAL. El valor exacto depende de los parámetros del cristal utilizado.

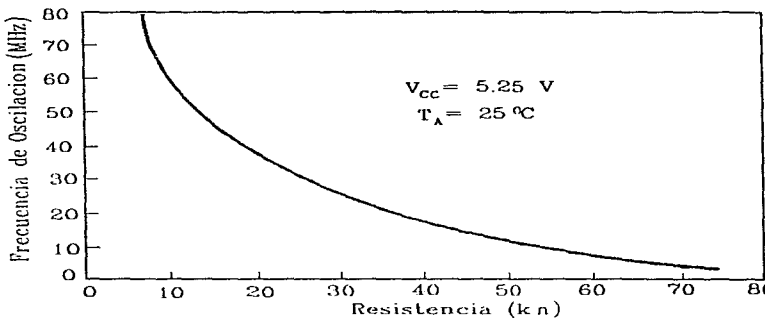


Figura 2. Frecuencia Típica vs Resistencia solamente para la opción del oscilador RC.

TIMER

Este pin se usa como señal externa de entrada para controlar la circuitería interna timer/contador. Este pin también detecta un nivel alto de voltaje que se utiliza al iniciar el programa bootstrap.

$\overline{\text{RESET}}$

Este pin tiene una entrada de disparador Schmitt y una resistencia de carga. El

MCU puede reinicializarse mediante un RESET de bajo nivel.

Líneas de entrada/salida (PA0-PA7, PB0-PB7, PC0-PC3)

Estas 20 líneas están colocadas en dos puertos de 8 bits (A y B) y un puerto C de 4 bits. Todas las líneas son programables como entradas o salidas bajo un software de control de los registros de dirección de datos.

Programación

Programación de Entrada/Salida

Cualquier pin de un puerto es programable como entrada o salida bajo un software de control del correspondiente registro de dirección de datos de escritura solamente (DDR); **DDR** siempre lee "1". La programación del puerto E/S se complementa mediante la escritura del correspondiente bit en el puerto **DDR**, con 1 lógico para salida y 0 lógico para entrada. En reset, todos los **DDR** se inicializan con un estado 0 lógico para poner los puertos en modo de entrada. Los registros de salida del puerto no se inicializan en reset y pueden escribirse antes de colocar los bits **DDR**.

Cuando se programan como salidas el dato de salida se lee como dato de entrada haciendo caso omiso de los niveles lógicos en los pines de salida, ya que sólo se toma en cuenta lo que haya a la salida. El dato de salida siempre puede escribirse. Por lo tanto cualquier escritura a un puerto escribe todos sus bits de datos, así el puerto **DDR** se dispone para entrada . Este puerto de escritura puede usarse para inicializar los registros de datos y evitar salidas indefinidas. Debe tener cuidado al utilizar instrucciones de lectura-modificación-escritura y respetar el que los datos corresponden al pin de nivel si el **DDR** está como entrada o de salida cuando el **DDR** es una salida. Ver la tabla 1 para funciones E/S y la figura 3 que muestra un circuito típico de un puerto.

Tabla1. Funciones de E/S

Bit del Registro de Dirección de Datos	Dato de Salida Bit	Estado de la Salida	Entrada al MCU
1	0	0	0
1	1	1	1
0	X	Hi-Z**	Pin

****Puertos A (con el manejador CMOS deshabilitado), B y C son puertos de tres estados. El puerto A tiene un dispositivo opcional de pullup que permite la capacidad de manejar CMOS.**

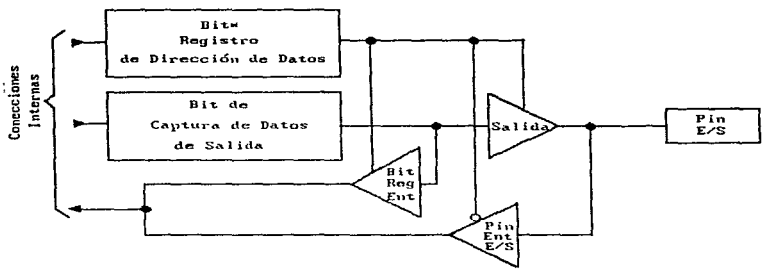
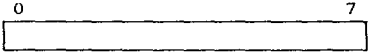


Figura 3. Circuito típico del circuito de E/S y configuración del registro

Registro de Dirección de Datos

Puerto A Dirección = $000
Puerto B Dirección = $001
Puerto C Dirección = $002 (bits 0 - 3)

Registro de Dirección del Puerto del Datos (DDR)



(1) Sólo de escritura: todo se lee como 1's
(2) 1= Salida; 0=Entrada. Se limpia a cero mediante el reset.
(3) Puerto A Dirección=$004
Puerto B Direcciòn = $005
Puerto C Direcciòn = $006 (bits 0 → 3)

Memoria

El MCU es capaz de direccionar **2048 bytes** de memoria y registros de E/S. El mapa de memoria se muestra en la figura 4. La EPROM consta de las siguientes localidades: ROM bootstrap, RAM, un registro de opción enmascarable (MOR), un registro de control de programa, y E/S.

Los vectores de interrupción se localizan desde **S7F8** hasta **S7FF**. El bootstrap es un programa en la ROM que permite al MCU programar su EPROM. El área de la pila se usa durante el procesamiento de una interrupción o llamada a subrutina para salvar el estado del CPU. El stack pointer se decrementa durante un push y se incrementa durante un pull. Ver interrupciones para más información.

NOTA

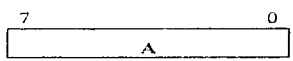
Al utilizar el área de la pila para almacenar datos o localidades de trabajo temporales se requiere tener cuidado para prevenir una sobreescritura.

Registros

El MCU contiene los registros que se describen a continuación.

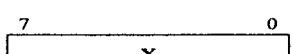
Acumulador(A)

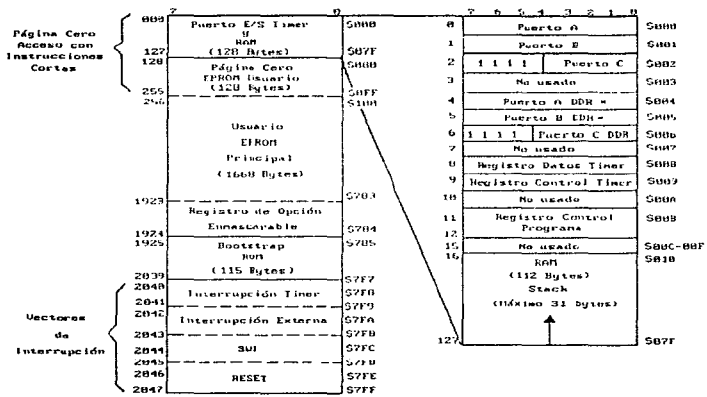
El acumulador es un registro de propósito general de 8 bits que se utiliza para mantener operandos y, resultados y cálculos aritméticos o de manipulación de datos.



Registro índice (X)

El registro índice es un registro de 8 bits que se usa en el modo de direccionamiento indexado. Contiene un valor de 8 bits que pueden sumarse a un valor inmediato de 8 o 16 bits para crear una dirección efectiva. El registro índice también puede utilizarse como un área de almacenamiento temporal.



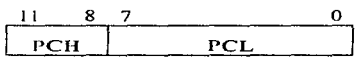


* Precaución: Los registros de dirección de datos (DDRs) únicamente son de escritura, y se leen como $FF.

Figura 4. Mapa de Memoria

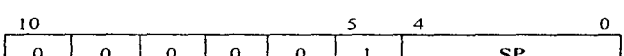
Contador de Programa (PC)

El contador de programa es un registro de 11 bits que contiene la dirección del siguiente byte a ejecutar.



Apuntador de pila (Stack Pointer (SP))

El apuntador a pila es un registro de 11 bits que contiene la dirección de la siguiente localidad libre en la pila. Durante una instrucción de reset al MCU o un reset del apuntador (RSP) el SP se coloca en la localidad **S07F**. Entonces el **SP** se decrementa y el dato se coloca dentro de la pila y se incrementa cuando el dato se saca de la pila.



Registro de códigos de condición

El registro de códigos de condición es un registro de 5 bits en el cual cuatro bits

se utilizan para indicar los resultados de la instrucción ejecutada. Estos bits pueden probarse individualmente con un programa, y se pueden tomar acciones específicas como resultado de su estado. Cada bit se explica en los siguientes párrafos.

4	3	2	1	0
H	I	N	Z	C

Half Carry (H) (Medio Acarreo)

Este bit se coloca durante las operaciones **ADD** y **ADC** para indicar que una carga ocurre entre los **bits 3 y 4**.

Interrupción (I)

Cuando este bit se coloca el timer y el interruptor externo se enmascara (deseable). Si ocurre una interrupción externa cuando el bit se coloca, la interrupción se captura y se procesa tan pronto como el bit de interrupción se limpia.

Negativo (N)

Cuando se coloca, este bit indica que el resultado del último dato aritmético,

lógico, o de manipulación fue negativo (el bit 7 en los resultados es un 1 lógico).

Cero (Z)

Cuando se coloca, este bit indica que el resultado del último dato aritmético, lógico, o de manipulación fue cero.

Acarreo/Borrow (C)

Cuando se carga este bit indica que un acarreo o préstamo salió de la unidad aritmética y lógica (ALU) y que ocurrió en la última operación aritmética. Sin embargo, este bit se ve afectado durante la prueba de bit e instrucciones de bifurcación, y durante desplazamientos y rotaciones.

Resets (Inicializadores)

El MCU puede reinicializarse de dos formas por la inicialización de encendido y por una entrada externa de reset ($\overline{\text{RESET}}$). La entrada de $\overline{\text{RESET}}$ consta principalmente de un disparador Schmitt que sensa un nivel lógico bajo de la línea.

Reset de Encendido

Un reset interno se genera mediante un encendido que permite que el generador de reloj interno se estabilice. El reset de encendido se utiliza estrictamente para las condiciones de encendido y no deberá utilizarse para detectar cualquier caída en la alimentación de voltaje. Un retardo de t_{RHL} de milisegundos se requiere antes de que la entrada del $\overline{\text{RESET}}$ sea un nivel alto. Conectando un capacitor a la entrada del $\overline{\text{RESET}}$ (figura 5) se provee un retardo suficiente.

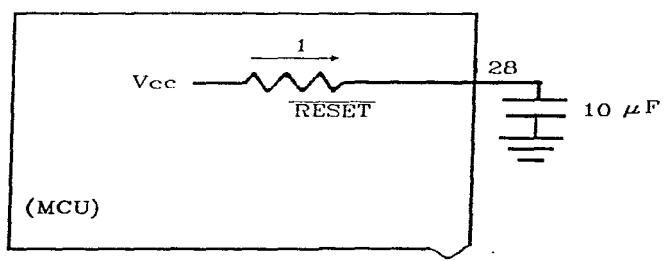


Figura 5. Circuito de retardo del encendido del reset

Entrada de $\overline{\text{RESET}}$ externa

El MCU se reinicializa con un 0 lógico que se aplica en la entrada del $\overline{\text{RESET}}$ por un largo periodo (t_{CYC}). Bajo este tipo de reset los interruptores de los disparadores Schmitt se apagan para un V_{IRES} para proveer un voltaje de reset interno.

Interrupciones

El MCU puede interrumpirse por cuatro caminos diferentes: (1) a través de una interrupción externa $\overline{\text{INT}}$ mediante un pin de entrada, (2) con un requerimiento de interrupción de un timer interno, (3) usando instrucciones de interrupción mediante software (SWI), o (4) mediante el puerto externo D ($\overline{\text{INT2}}$) que es un pin de entrada.

Las interrupciones causan que los registros de proceso se salven en una pila y en una interrupción enmascarada (un bit) previniendo a interrupciones adicionales. Las instrucciones RTI causan que los registros del CPU se recuperen de la pila cuando se reasuma su proceso normal. El orden de almacenamiento se muestra en la figura. 6.

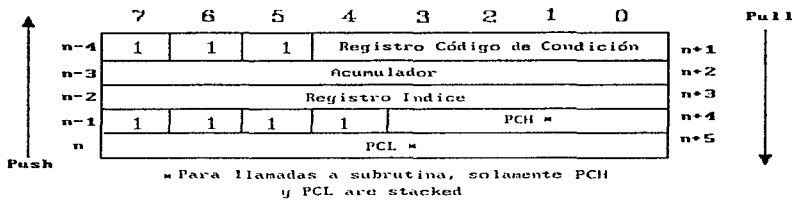


Figura 6. Orden de almacenamiento de interrupciones

Cuando no se requiera el $\overline{\text{RESET}}$, las interrupciones del hardware harán que no se ejecuten instrucciones pero se consideran como pendientes hasta que la instrucción actual se complete.

NOTA

La instrucción que se ejecuta es considerada como la única operando.

Cuando la instrucción se completa el procesador verifica todas las interrupciones pendientes del hardware y, si se desenmascara (un bit limpio) se procede con el proceso de interrupción; en otro caso, la siguiente instrucción se llama y se ejecuta.

Las interrupciones enmascaradas son llamadas mediante un servicio de interrupción posterior. El estado de interrupción del timer se limpia después de desenmascarar la interrupción cuando la interrupción no se carga.

Si ambos, una interrupción externa y una interrupción de timer, están pendientes al final de la ejecución de alguna instrucción, primero se da servicio a la interrupción externa. El SWI se ejecuta como cualquier otra instrucción sin reparar en la activación del bit I. Ver figura 7 para la secuencia de procesamiento de las instrucciones de reset y de interrupción.

Interrupciones de timer

Si el bit de enmascaramiento (TCR6) se limpia, entonces, cada vez que el timer se decrementa hacia cero (transiciones desde $01 a $00), una interrupción se genera. El proceso de la interrupción solo se genera si el bit de enmascaramiento de la interrupción del registro de códigos de condición (CCR) está limpio. Cuando se reconoce la interrupción, el estado actual de la máquina se coloca dentro de la pila y el bit I se coloca en el CCR, mientras que las otras interrupciones se dejan pendientes

mientras se hace el servicio actual. El contenido del vector de interrupción del timer, contiene la localidad de la rutina de servicio de interrupción del timer, entonces esto se carga dentro del contador del programa. Al final de la rutina de servicio de interrupción del timer, normalmente el software ejecuta una instrucción RTI la cuál restaura el estado de la máquina y comienza a ejecutar el programa interrumpido. El bit de estado de interrupción del timer solo puede limpiarse mediante software.

Interrupciones externas

La interrupción externa se sincroniza internamente y entonces es capturada en el flanco de bajada de $\overline{\text{INT}}$ e $\overline{\text{INT2}}$. Limpiando el **bit I** se habilita la interrupción externa. La interrupción $\overline{\text{INT2}}$ tiene un bit de requerimiento de interrupción (**bit 7**) y un bit de enmascaramiento (**bit 6**) en el registro misceláneo (**MR**). La interrupción $\overline{\text{INT2}}$ se inhibe cuando el bit de enmascaramiento se coloca. La $\overline{\text{INT2}}$ siempre se lee como una entrada digital en el puerto D. Los bits de requerimiento de interrupción de timer y la $\overline{\text{INT2}}$, si se colocan, causan que el MCU procese una interrupción cuando el bit I del registro de código de condición se limpia. Los siguientes párrafos describen dos circuitos típicos de interrupción externa.

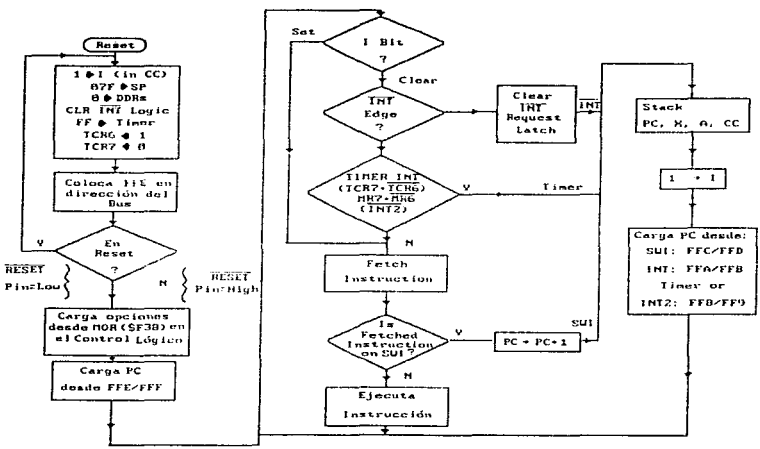


Figura 7. Diagrama de flujo del procesamiento de interrupción y reinicialización

Interrupción de Cruce por Cero

Una señal de entrada senoidal (f_{INT} máxima) puede usarse para generar una interrupción externa (ver figura 8a) para usarlo como un detector de cruce por cero (para transiciones negativas de la sinusoidal en "ac"). Este tipo de circuito permite aplicaciones tales como rutinas de servicio de tiempo real y de atracción/repulsión de dispositivos de control de potencia de "ac". Una rectificación de onda completa fuera del chip provee una interrupción en cada cruce por cero de la señal de "ac" y con lo cual permite un reloj de 2f.

Interrupción por Señal Digital

Con este tipo de circuito (figura 8b), el pin $\overline{\text{INT}}$ puede manipularse mediante una señal digital. La frecuencia máxima de una señal puede reconocerse mediante el **TIMER** o el pin $\overline{\text{INT}}$ lógicamente esto depende del parámetro etiquetado t_{WL}, t_{WH}. Ver **TIMER** para más información.

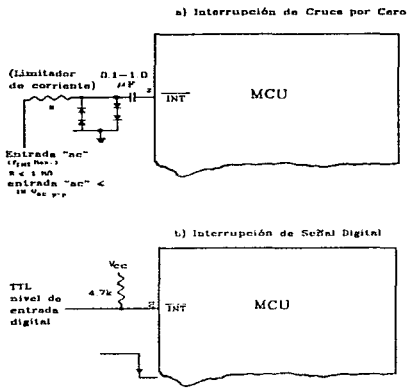


Figura 8. Circuitos típicos de interrupción

Interrupción de Software (SWI)

La SWI es una instrucción ejecutable que se ejecuta no obstante el estado del **bit I** en el CCR. Si el **bit I** es cero, SWI se ejecuta después de otras interrupciones. La ejecución de SWI es similar a las interrupciones por hardware.

Modos de Operación

El MCU tiene dos modos de operación: normal y bootstrap.

Modo Normal

Este modo es del chip sencillo y se activa si se reúnen las siguientes condiciones:

(1) la línea $\overline{\text{RESET}}$ es baja.

(2) el pin PC0 está dentro de su rango normal de operación y

(3) el pin V_{PP} es conectado al pin V_{CC}.

El siguiente flanco de subida en el pin de $\overline{\text{RESET}}$ causa entonces que este entre al modo normal.

Modo Bootstrap

Al modo bootstrap se entra si se le conectan +12V al pin TIMER.

TIMER

El MCU consta de un contador programable por software de 8 bits controlado por un preescalador de 7 bits programado por software. Varias fuentes de reloj pueden seleccionarse mediante el registro de control de tiempo (TCR). El contador de 8 bits puede cargarse bajo el control de un programa y se decrementa hacia el cero. Cuando el timer llega a cero, la interrupción de timer requiere un bit (el **bit 7**) que se coloca en el registro de control timer (TCR). Ver figura 9 (diagrama de bloques del timer).

La interrupción timer puede enmascararse (deshabilitarse) colocando el bit de la interrupción timer enmascarada (**bit 6**) en el **TCR**. Cuando el **bit I** en el registro de código de condición se limpia, y el **bit 6** del TCR se limpia, el procesador recibe la interrupción. El MCU responde a esta interrupción: 1) salvando el estado presente del CPU en la pila, 2) localizando el vector de interrupción del temporizador, y 3) ejecutando la rutina de interrupción. La interrupción del temporizador requiere un bit que debe limpiarse mediante el software. Ver resets e interrupciones para más información. El preescalador es un divisor de 7 bits el cual se utiliza para extender a

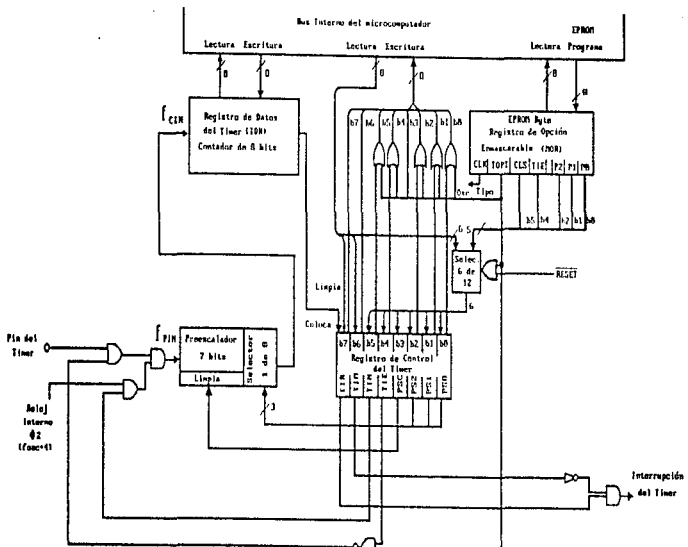


Figura 9. Diagrama de bloques del temporizador

su máxima longitud el temporizador. Para evitar errores de truncamiento, el preescalador se limpia cuando el **bit 3** del TCR se coloca en un 1 lógico, sin embargo, el **bit 3** TCR siempre se lee como un 0 lógico de lectura-modificación-escritura.

El temporizador continua el conteo pasando por cero, decreciendo desde $00 hasta $FF, y continua el conteo descendente. El contador puede leerse en cualquier momento mediante la lectura del registro de datos del temporizador **(TDR)**. Esto permite un programa para determinar la longitud del tiempo cuando una interrupción del temporizador ocurre, sin alterar el proceso de conteo. **TDR** no se afecta con el reset.

Modo controlador de software

La entrada del preescalador del temporizador puede configurarse para 3 modos de operación diferentes más un modo deshabilitado, dependiendo del valor escrito en los bits de control del **TCR (bit 4 y 5) (TIE y TIN)**. Los siguientes párrafos describen los diferentes modos:

Modo 1 de Entrada al Timer

Cuando **TIE=0** y **TIN=0**, la entrada al timer es el reloj interno (fase 2) y el pin de entrada al timer se deshabilita. El modo de reloj interno, puede utilizarse para generar interrupciones periódicas, tan bien como una referencia para frecuencia y medición de eventos.

Modo 2 de Entrada al Timer

Cuando **TIE=1** y **TIN=0**, el reloj interno y las señales de entrada al temporizador son colocadas en una compuerta lógica "AND" para formar la entrada a este. Este modo puede utilizarse para medir el ancho de pulsos externos. El flanco alto del pulso externo dispara al reloj interno por la duración del pulso externo. La precisión del contador es ±1.

Modo 3 de Entrada al Timer

Cuando TIE=0 y TIN=1, se aplica una entrada no preescalada en frecuencia al preescalador y se deshabilita el timer.

Modo 4 de Entrada al Timer

Cuando **TIE** y **TIN** sean iguales a uno, la entrada del timer es desde el reloj externo. El reloj externo puede usarse para contar eventos externos, si este es provisto de una frecuencia externa para generar interrupciones periódicas.

Modo controlado por MOR

Este modo se selecciona cuando **TOPT (bit 6)** en el **MOR** se programa a 1 lógico. Los circuitos timer son los mismos como se describieron en el modo controlado por software MOR. Los niveles lógicos de **TCR bits 0, 1, 2** y **5** se determinan durante la programación de la EPROM por los mismos bits en el **MOR**. Aún más, los **bits 0, 1, 2** y **5** en el **MOR** controlan la división del preescalador y la selección del reloj del timer. Los bits **TIE (bit 4)** y **PSC (bit 3)** en el **TCR** se colocan a 1 lógico en el modo controlado por **MOR**.

TIM (bit 6) y **TIR (bit 7)** se controlan mediante el contador y software.

Registro controlador del timer (TRC) $009

Este es un registro de 8 bits que controla varias funciones tales como configurar

el modo de operación, establecer la razón del preescalador y generar la señal requerida para la interrupción del timer. Todos los bits son de lectura/escritura excepto el **bit 3**. Cuando en el **MOR TOPT=1**, entonces los **bits 5, 2, 1 y 0** en el **TCR** toman los bits correspondientes en el **MOR** durante el reset.

7	6	5	4	3	2	1	0
TIR	TIM	1	1	1	1	1	1
RESET							
0	1		U	U			

TCR con MOR TOPT=1 (Emulación del MC6805P2/P6)

7	6	5	4	3	2	1	0
TIR	TIM	TIN	TIE	PSC	PS2	PS1	PS0
RESET							
0	1						

TCR con MOR TOPT = 0 (Timer programable por software)

TIR Petición de interrupción por Timer

Se usa para indicar la interrupción del timer cuando este es un uno lógico.

1 Se coloca cuando el registro de datos del timer cambia todo a ceros.

0 Se limpia mediante el reset externo, power on reset o bajo el control de un programa.

TIM Interrupción enmascarada del timer.

Se usa para inhibir la interrupción del timer

1 Inhibe interrupción

0 Habilita interrupción

TIN Externo o interno

Selecciona la fuente de entrada del reloj

1 Selecciona reloj externo

0 Selecciona reloj interno ($f_{osc}/4$)

TIE Habilita el timer externo

Se usa para habilitar el pin que corresponde al timer externo

1 Habilita el pin del timer externo

1 Deshabilita el pin del timer externo

PSC Limpia el preescalador

Sólo escribe un bit. Escribiendo un 1 este bit reinicializa el preescalador a cero. Una lectura a esta localidad siempre indica un cero.

PS2, PS1, PS0 Limpiar el Preescalador

Decodifica la selección de una de las 8 salidas del preescalador

PS2	PS1	PS0	División Preescalada
0	0	0	1 (Preescalador en Bypass)
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Registro de opción enmascarable (MOR)

El MOR se implementa en la EPROM y contiene todos sus bits en ceros previos a la programación. Este registro no se afecta con el reset. Los bits MOR se describen en los siguientes párrafos.

7	6	5	4	3	2	1	0
CLK	TOPT	CLS	TIE		P2	P1	P0

CLK Reloj (de tipo oscilador)

1 Resistor Capacitor (RC)

0 Cristal

TOPT Opción timer

1 Tipo MC6805 P2/P6 timer/preescalador. Todos los bits excepto el **6 y el 7** del TCR son invisibles al usuario.

Los bits 5, 2, 1 y 0 del MOR, determinan el equivalente a las opciones enmascaradas del MC6805 P2/P6.

0 Todos los bits del TCR se implementan como un timer programable de software. El estado de los bits 5, 4, 2, 1 y 0 del MOR colocan los valores iniciales de sus respectivos bits TCR.

CLS Fuente del reloj del Timer/Preescalador

1 Pin de timer externo

0 Reloj interno

TIE Habilita el timer externo

No se use si TOPT=1. Coloca el valor inicial de TIE en el TCR si TOPT=0.

1 No utilizado

0 Coloca el valor inicial en el TCR.

P2, P1, P0

Los niveles lógicos de estos bits se decodifican cuando se selecciona una de las 8 salidas en el preescalador del timer.

Registro del control de programa (PCR)

El PCR es un registro de 8 bits el cual provee el control necesario de bits para programar la EPROM. El programa bootstrap manipula el PCR cuando este esta programando, al usuario no le concierne saber más aplicaciones del PCR.

7	6	5	4	3	2	1	0
1	1	1	1	1	$\overline{\text{VPON}}$	$\overline{\text{PGE}}$	$\overline{\text{PLE}}$
RESET							
U	U	U	U	U	U	1	1

$\overline{\text{PLE}}$ Habilita la captura del programa

Controla las direcciones y el comienzo de datos en la EPROM. Durante el reset se coloca, pero puede limpiarse en cualquier momento.

1 Lee EPROM

0 Captura la dirección y los datos sobre la EPROM.

$\overline{\text{PGE}}$ Habilita la programación

Habilita la programación de la EPROM. Puede colocarse cuando cambia la

dirección y el dato.

Se coloca durante el reset.

1 Inhibe la programación de la EPROM.

0 Habilita la programación de la EPROM (si $\overline{\text{PLE}}$ es baja).

$\overline{\text{VPON}}$ V_{PP} encendido

Un sólo bit de lectura indica un alto voltaje en el pin V_{PP}. Cuando se coloca un "1", se desconecta $\overline{\text{PGE}}$ y $\overline{\text{PLE}}$ del chip.

1 No hay alto voltaje en el pin V_{PP}.

0 Alto voltaje.

NOTA

$\overline{\text{VPON}}$ en 0 no indica que el nivel V_{PP} es correcto para programar. Se usa como una sincronización segura para el usuario en el modo de operación normal.

$\overline{VPON}$	$\overline{PGE}$	$\overline{PLE}$	Condiciones de Programación
0	0	0	Modo de Programación (Byte para programar la EPROM)
1	0	0	$\overline{PGE}$ y $\overline{PLE}$ desabilitados desde el sistema
0	1	0	Deshabilitador de la programación (Latch Address and Data in EPROM)
1	1	0	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados desde el sistema
0	0	1	Estado inválido $\overline{PGE}$=0 si $\overline{PLE}$=0
1	0	1	Estado inválido $\overline{PGE}$=0 si $\overline{PLE}$=0
0	1	1	Alto Voltaje en V_{pp}
1	1	1	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados desde el sistema (modo de operación)

Conjunto de Instrucciones

El MCU tiene un conjunto de 59 instrucciones básicas las cuales pueden dividirse en 5 tipos diferentes: registro/memoria, lectura-modificación-escritura, bifurcación, manipulación de bits, y control. Los siguientes párrafos explican cada

tipo:

Instrucciones Registro / Memoria

Muchas de estas instrucciones usan dos operandos. Un operando puede resindir en el acumulador o el registro índice. El otro operando se obtiene de memoria usando uno de los modos de direccionamiento. Las instrucciones de salto incondicional (JMP) y salto a subrutina (JSR) no tienen operando de registro. Veamos la siguiente lista de instrucciones.

Función	Mnemónico
Carga A desde la memoria	LDA
Carga X desde la memoria	LDX
Almacena A en la memoria	STA
Almacena X en la memoria	STX
Suma la memoria al contenido de A	ADD
Suma la memoria y el acarreo al contenido de A	ADC
Resta a la memoria	SUB
Resta a la memoria desde A con préstamo	SBC
Realiza una AND de la memoria con A	AND
Realiza una OR de la memoria con A	ORA

Función	Mnemónico
Realiza una OR Exclusiva de la memoria con A	EOR
Comparación aritmética de A con la memoria	CMP
Comparación aritmética de X con la memoria	CPX
Bit de verificación de la memoria con A (comparación lógica)	BIT
Salto incondicional	JMP
Salto a subrutina	JSR

Instrucciones lectura-modificación-escritura

Estas instrucciones leen una localidad de memoria o un registro, modifican o verifican estos contenidos y escriben el valor modificado en memoria o en un registro. Las instrucciones de verificación para no negativa o cero (TST) son una excepción de la lectura-modificación-escritura ya que no modifican el valor. Veamos la siguiente lista de instrucciones.

Función	Mnemónico
Incremento	INC
Decremento	DEC
Limpia	CLR

Función	Mnemónico
Complemento	COM
Negado (Segundo complemento)	NEG
Rotación a la izquierda con acarreo	ROL
Rotación a la derecha con acarreo	ROR
Corrimiento lógico a la izquierda	LSL
Corrimiento lógico a la derecha	LSR
Corrimiento aritmético a la derecha	ASR
Verifica para negativo o cero	TST

Instrucciones de bifurcación

Este conjunto de instrucciones bifurca si se da una condición en particular; en otro caso la operación no se realiza. Las instrucciones de bifurcación son de 2 bytes. Veamos la siguiente lista de instrucciones:

Función	Mnemónico
Bifurca siempre	BRA
Nunca bifurca	BRN
Bifurca si es alto	BHI
Bifurca si es bajo o igual	BLS

Función	Mnemónico
Bifurca si el acarreo está limpio	BCC
(Bifurca si es alto o igual)	(BHS)
Bifurca si se coloca el acarreo	BCS
Bifurca si no es igual	BNE
Bifurca si es igual	BEQ
Bifurca si el acarreo medio está limpio	BHCC
Bifurca si el acarreo medio se coloca	BHCS
Bifurca si suma	BPL
Bifurca si disminuye	BMI
Bifurca si el bit de interrupción enmascarable se limpia	BMC
Bifurca si el bit de interrupción enmascarable se coloca	BMS
Bifurca si la línea de interrupción es baja	BIL
Bifurca si la línea de interrupción es alta	BIH
Bifurca a subrutina	BSR

Instrucciones de control

Estas instrucciones registran las instrucciones de referencia y se usan para

controlar la operación del procesador durante la ejecución de un programa. Veamos la siguiente lista.

Función	Mnemónico
Transfiere A a X	TAX
Transfiere X a A	TXA
Coloca el bit de acarreo	SEC
Limpia el bit de acarreo	CLC
Coloca el bit de interrupción enmascarada	SEI
Limpia el bit de interrupción enmascarada	CLI
Interrupción por Software	SWI
Retorno de subrutina	RTS
Retorno de interrupción	TRI
Reinicializa el Stack Pointer	RSP
No Opera	NOP

Instrucciones de manipulación de bits

El MCU es capaz de colocar o limpiar cualquier bit que resida en los primeros 256 bytes del espacio de memoria, donde todos los registros de puerto, puertos DDR, timer, control del timer y de la RAM. Una característica adicional permite la

verificación de software y bifurcación de un estado de. cualquier bit en estas 256 localidades. Las funciones de limpiar bit y verificar bit y de bifurcación se implementan con una simple instrucción. Para instrucciones de verificación y bifurcación, el valor del bit verificado también se coloca en el bit de acarreo del registro de código de condición. Veamos la siguiente lista.

Función	Mnemónico
Bifurca si el bit n se coloca	BRSET n (n = 0...7)
Bifurca si el bit n se limpia	BRCLR n (n = 0...7)
Coloca el bit n	BSET n (n = 0...7)
Limpia el bit n	BCLR n (n = 0...7)

Modos de direccionamiento

El MCU usa 10 modos de direccionamiento diferentes para darle al programador la oportunidad de optimizar el código para cualquier situación. Los diversos modos de direccionamiento indexado hacen esto posible al localizar tablas de datos, tablas de

conversión de código y tablas de escalamiento en cualquier lugar del espacio de memoria. Los accesos cortos indexados simplemente son instrucciones de un byte, mientras que las instrucciones largas son de 3 bytes y permiten accesar tablas de memoria. También se incluyen direccionamientos absolutos cortos y largos. Las instrucciones de direccionamiento directo de 2 bytes accesan todos los bytes de datos en más aplicaciones. El direccionamiento extendido permite instrucciones de salto que alcanzan toda la memoria.

El término dirección efectiva (EA) se usa para describir varios modos de direccionamiento. La EA se define como la dirección desde la cual el argumento de una instrucción se carga o almacena.

Inmediato

En el modo de direccionamiento inmediato, el operando está contenido en el byte inmediato siguiente al código de operación. El modo de direccionamiento inmediato se usa para accesar constantes que no cambian durante la ejecución del programa (por ejemplo una constante que se usa para inicializar un ciclo de conteo).

Directo

En el modo de direccionamiento directo, la dirección efectiva del argumento está contenida en un byte que le sigue al byte del código de operación. El direccionamiento directo permite el uso de la dirección directamente en los 256 bytes bajos en memoria con una sola instrucción de 2 bytes.

Extendido

En el modo de direccionamiento extendido, la dirección efectiva del argumento está contenida en los 2 bytes siguientes al byte del código de operación. Las instrucciones con modo de direccionamiento extendido son capaces de referenciar argumentos en cualquier lugar de la memoria con una instrucción de 3 bytes.

Relativo

El modo de direccionamiento relativo sólo se usa en instrucciones de bifurcación. En el direccionamiento relativo, el contenido de los 8 bits llamados byte (el offset), siguientes al código de operación, se añaden al PC si y sólo si, las condiciones de bifurcación son ciertas.

En otro caso, el control se pasa a la siguiente instrucción. El largo del modo de direccionamiento relativo va desde -126 a +129 de la dirección del código de operación.

Indexado sin offset

En el modo de direccionamiento indexado, sin offset, la dirección efectiva del argumento está contenida en los 8 bits del registro índice. Este modo de direccionamiento puede accesar las primeras 256 localidades de memoria. Estas instrucciones son de un byte de longitud. Este modo es utilizado para mover un apuntador a una tabla o para mantener la dirección de una referencia de RAM frecuente o una locación de E/S.

Indexado con 8 bits de offset

En el modo de direccionamiento indexado con 8 bits de offset, la dirección efectiva es la suma de los contenidos de los registros índice de 8 bits sin signo y del byte sin signo que precede al código de operación. Este modo de direccionamiento se utiliza para seleccionar el elemento K-ésimo en un n elemento de una tabla. Con

instrucciones de 2 bytes, k típicamente puede estar en X con la dirección del comienzo de la tabla en la instrucción. Así, las tablas pueden comenzar en cualquier lugar dentro de las 256 localidades de memoria y pueden extenderse tan lejos como la localidad 510 ($1FE es la última localidad en la cual la instrucción puede comenzar).

Indexado con 16 bits de offset

En el modo de direccionamiento indexado con 16 bits de offset, la dirección efectiva es la suma de los contenidos del registro índice sin signo de 8 bits y de los dos bytes sin signo siguientes al código de operación. Este modo de direccionamiento puede utilizarse de forma similar al modo indexado con 8 bits de offset, excepto que este utiliza instrucciones de 3 bytes que permiten el uso de tablas en cualquier parte de la memoria.

Bit Set/Clear

En el modo de direccionamiento bit set/clear, el bit que se coloca o limpia es parte del código de operación. El byte siguiente al código de operación especifica la dirección directa del byte en el cual el bit especificado se colocará o se limpiará.

Cualquier bit de lectura/escritura en las primeras 256 localidades de memoria, incluyendo E/S, puede colocarse o limpiarse con una instrucción de 2 bytes.

Precaución

Los correspondientes DDRs para puertos A, B y C sólo son registros de escritura (registros $004 a $005, y $006). Una operación de lectura a estos registros siempre retorna un "1". Dado que BSET y BCLR son funciones de lectura-modificación-escritura no pueden utilizarse para colocar o limpiar un bit DDR. Es recomendable que todos los bits de puertos DDR se escriban utilizando una instrucción de almacenamiento.

Bit prueba y bifurcación

El modo de direccionamiento bit test and branch es una combinación de los modos de direccionamiento directo y relativo. El bit que será verificado y su condición (colocar o limpiar) se incluyen en el código de operación. La dirección del byte a verificar está en el byte inmediatamente después del byte del código de operación. El offset de 8 bits relativo y signado en el tercer byte se suma al PC si el bit especificado

se coloca o se limpia en la localidad de memoria correspondiente. Esta instrucción de 3 bytes permite que el programa se bifurque basándose en la condición de cualquier bit leíble en las primeras 256 localidades de memoria. El largo de la bifurcación está desde -125 a +130 desde la dirección del código de operación. El estado del bit verificado se transfiere al bit de acarreo del registro de código de condición.

Inherente

En el modo de direccionamiento inherente, toda la información necesaria para ejecutar la instrucción está contenida en el código de operación. Las operaciones especifican sólo el registro índice o el acumulador tan bien como el control de instrucciones con otros argumentos, se incluyen en este modo. Estas instrucciones son de un byte de longitud.

Especificaciones Eléctricas

Rangos Máximos

Rangos	Símbolo	Valor	Unidad
Voltaje de alimentación	V_{CC}	-0.3 a +7.0	V
Voltajes de entrada EPROM Voltaje de programación (Pin VPP) Pin de TIMER (modo normal) Pin TIMER (modo de programación bootstrap) Todos los demás	V_{PP} V_{in} V_{in} V_{in}	-0.3 a +22.0 -0.3 a +7.0 -0.3 a +15.0 -0.3 a +7.0	V V V V
Rango de operación de la temperatura	T_A	T_L a T_H 0 a +70	°C
Rango de temperatura de almacenamiento	T_{stg}	-55 a +150	°C
Temperatura de la unión del chip	T_J	150	°C/W

Características Térmicas

Características	Símbolo	Valor	Unidad
Resistencia Térmica del chip	θ_{JA}	60	°C/W

Consideraciones de Potencia

El promedio de la temperatura de la unión del chip , T_J, en °C puede obtenerse de la siguiente manera:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad \ldots\ldots (1)$$

donde:

T_A = Temperatura ambiente, °C.

θ_{JA} = Resistencia térmica del encapsulado, Unión a temperatura ambiente, °C/W.

$P_D = P_{INT} + P_{I/O}$

$P_{INT} = I_{CC} \times V_{CC}$, Watts - Potencia interna del chip.

$P_{I/O}$ = Disipación de la potencia en los pines de entrada y salida - Determinado por el usuario.

Para más aplicaciones $P_{I/O} < P_{INT}$ y puede ser ignorado. La siguiente es una relación es una aproximación entre P_D y T_J (Si se ignora $P_{I/O}$):

$$P_D = K / (T_J + 273\ °C) \quad \ldots\ldots (2)$$

Resolviendo las ecuaciones (1) y (2) para K, se obtiene:

$$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P^2_D \quad \ldots\ldots (3)$$

donde K es una constante perteneciente a la parte particular. K puede determinarse mediante la ecuación (3) midiendo P_D (en equilibrio) para conocer T_A. Utilizando este valor de K, los valores de P_D y T_J pueden obtenerse resolviendo iterativamente las ecuaciones (1) y (2) para cualquier valor de T_A.

Características Eléctricas

(Vcc=5.25 +-0.5 V_{DC}, Vss=0 V_{DC}, Vss=0 V_{DC}, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de entrada alto	V_{IH}				V
$\overline{\text{RESET}}$(4.75 <= V_{CC} <=5.75)		4.0	-	Vcc	
$\overline{\text{INT}}$ (4.75 <= V_{CC} <=5.75)		Vcc-0.5	-	Vcc	
(V_{CC} <4.75))		4.0	*	Vcc	
Todos los demás		Vcc-0.5	*	Vcc	
		2.0	-	Vcc	
Voltaje de entrada alto (Pin TIMER)	V_{IH}				V
Modo Timer		2.0	-	Vcc	
Modo de programación bootstrap		9.0	12.0	15.0	

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de entrada bajo $\overline{\text{RESET}}$ $\overline{\text{INT}}$ Todos los demás	V_{IH}	 -0.3 -0.3 -0.3	 -- ** --	 0.8 1.5 0.8	V
Disipación interna de energía (No carga al puerto, V_{CC}= 5.25 V, T_A= °C)	P_{INT}	---	450	TBD	mW
Capacitancia de entrada XTAL Todas las demás	C_{in}	 --- ---	 25 10	 -- --	pF
$\overline{\text{INT}}$ voltaje de cruce por cero, a través de un capacitor	V_{INT}	2.0	--	4.0	Vacp-p
$\overline{\text{RESET}}$ voltaje de histéresis Fuera del voltaje de reinicialización Dentro del voltaje de reinicialización	 V_{IRES}+ V_{IRES}-	 2.1 0.8	 -- --	 4.0 2.0	V
Voltaje de programación (Pin V_{PP}) Programación de la EPROM Modo de operación	V_{PP}*	 20.0 4.0	 21.0 V_{CC}	 22.0 5.75	V

Características	Símbolo	Min	Tipo	Max	Unidad
Entrada actual TIMER (V_{in} = 0.4V) $\overline{INT}$ (V_{in} = 0.4V) EXTAL (V_{in}=2.4 V a V_{CC} opción del cristal) (V_{in}=0.4 V opción del cristal) $\overline{RESET}$ (V_{in}=0.8 V) (Capacitor externo de cambio de corriente)	I_{in}	 --- --- --- --- -4.0	 -- 20 -- -- --	 20 50 10 -1600 -40	µA

*V_{PP} es el pin 6 en el MC68705P3 y está conectado a V_{CC} en el modo de operación normal. En el MC6805P2, el pin 6 es NUM y está conectado a V_{SS} en el modo de operación normal.

**Debido a la oblicuidad interna, esta entrada (cuando no es utilizada) varía aproximadamente entre 2.0 V.

Puerto DC características eléctricas

(V_{CC}=+5.25 +-0.5 Vdc, V_{SS}=0 Vdc, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de Salida bajo, I_{LOAD}=1.6mA	V_{OL}	---	---	0.4	V
Voltaje de salida alto I_{LOAD}= -100µA	V_{OH}	2.4	---	---	V

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de salida alto I_{LOAD}= -10μA	V_{OH}	V_{CC}-10	---	---	V
Voltaje de entrada alto I_{LOAD}= -300μA (Max)	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo I_{LOAD}= -500μA (Max)	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada (V_{in} = 20 V a V_{cc})	I_{IH}	---	---	-300	μA
Hi-Z estado actual de la entrada (V_{in} = 0.4 V)	I_{IL}	---	---	-500	μA
Puerto B					
Voltaje de Salida bajo, I_{Load} = 3.2 mA	V_{OL}	---	---	0.4	V
Voltaje de Salida bajo, I_{Load} = 10 mA (Sink)	V_{OL}	---	---	1.0	V
Voltaje de salida alto I_{Load} = -200 μA	V_{OH}	2.4	---	---	V
Darlington Current Drive (Source) V_O = 1.5 V	I_{OH}	-1.0	---	-10	mA
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V

Características	Símbolo	Min	Tipo	Max	Unidad
Hi-Z estado actual de la entrada	I_{TSI}	---	2	20	μA
Puerto C					
Voltaje de Salida bajo, I_{Load} = 1.6 mA	V_{OL}	---	---	0.4	V
Voltaje de Salida alto, I_{Load} = 100 μA	V_{OH}	2.4	---	---	V
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada	I_{TSI}	---	---	20	μA

Características de encendido

(V_{CC}= +5.25 +/- 0.5 Vdc, V_{SS}=0 Vdc, T_A= 0° C a 70° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Frecuencia del Oscilador Normal	f_{osc}	0.4	---	4.2	MHz

Características	Símbolo	Min.	Tipo	Max.	Unidad
Tiempo del Ciclo de intrucción	t_{cyc}	0.950	---	10	μs
Ancho del pulso de INT o Timer	t_{WL}, t_{WH}	t_{cyc}+250	---	---	ns
Ancho del pulso de RESET	t_{RWL}	t_{cyc}+250	---	---	ns
Tiempo de retardo del RESET (Capacitancia externa = 1.0 μF)	t_{RHL}	100	---	---	ms
Detección de la frecuencia de entrada de cruce por cero INT	f_{INT}	0.03	---	1.0	kHZ
Ciclo duty del reloj externo (EXTAL)	---	40	50	60	%

Características de la operación de programación eléctrica

(V_{CC}= +5.25 +/-0.5 Vdc, V_{SS}=0 Vdc, T_A= 20° C a 30° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Voltaje de programación (Pin V_{PP})	V_{PP}	20.0	21.0	22.0	V
Current Supply V_{PP} V_{PP} = 5.25 V V_{PP} = 21.0 V	I_{PP}	 --- ---	 --- ---	 8 30	mA

Características	Símbolo	Min.	Tipo	Max.	Unidad
Frecuencia de oscilación de programación	f_{oscp}	0.9	1.0	1.1	MHz
Voltaje del modo de programación bootstrap (Pin del TIMER) I_{in} = 100	V_{HTP}	9.0	12.0	15.0	V

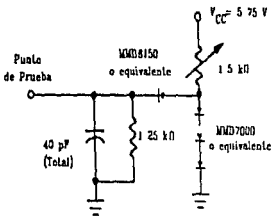


Figura 10. TTL Equivalent Test Load (Port B)

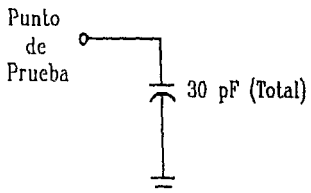


Figura 11. CMOS Equivalent Test Load (Port A)

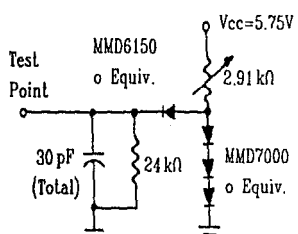


Figura 12. TTL Equivalent Test Load (Ports A and C)

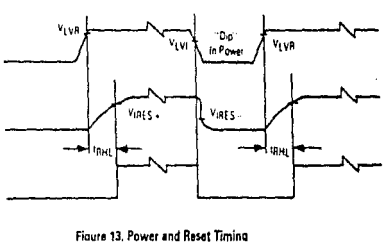


Figure 13. Power and Reset Timing

Figura 13. Open-Drain Equivalent Test Load (Port C)

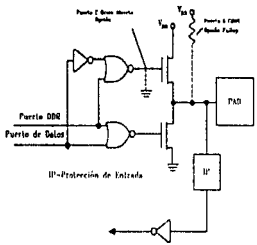


Figura 14. Diagrama Lógico de los Puertos A y C

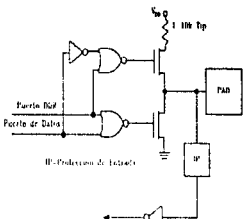


Figura 15. Diagrama Lógico del Puerto B

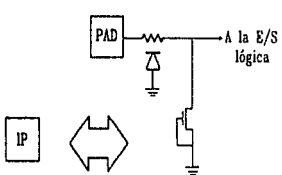


Figura 16. Típica Protección de Entrada

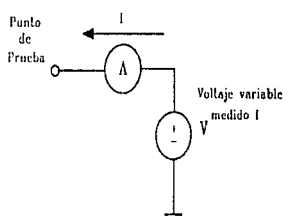


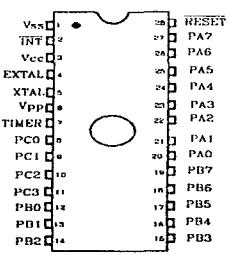
Figura 17. I/O Characteristic Meassurement Circuit

Información Genérica

La siguiente tabla provee información genérica perteneciente al tipo de encapsulado, y a los número del MC para el MC68705P3.

Tabla 3. Información Genérica

Tipo de encapsulado	Frecuencia del reloj interno (MHz)	Temperatura	Número de orden
Encapsulado (Sufijo S)	1.0	0 °C a 70 °C	MC68705P3S
Encapsulado (Sufijo S)	1.0	-40 °C a +85 °C	MC68705P3CS



ASIGNACION DE PINES

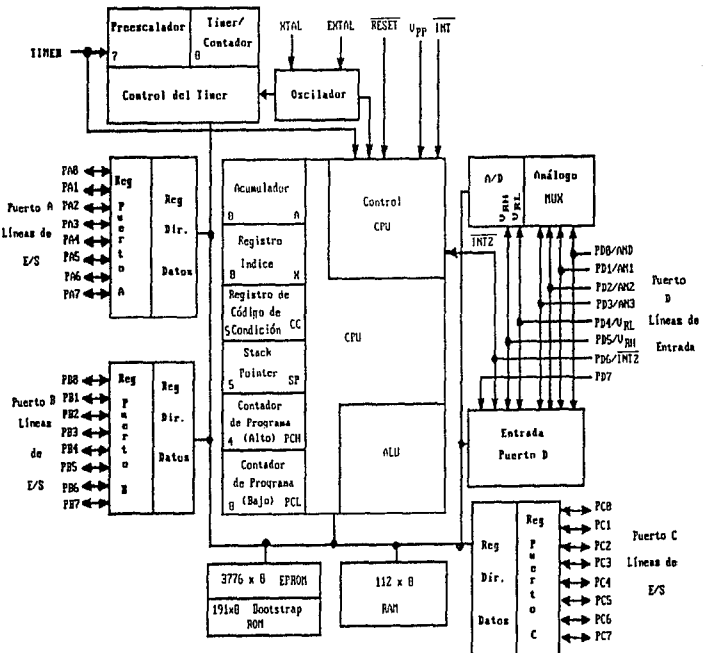
MC68705R3
Microcontrolador de 8 bits con EPROM

La unidad microcontroladora (MCU) MC68705R3 (HMOS) es miembro de la familia de microcomputadoras MC6805 con EPROM. La EPROM programable permite cambiar programas y realizar aplicaciones de bajo volumen. Esta MCU es de bajo costo, tiene capacidad de E/S paralela con pines programables como entrada o salida.

A continuación se muestra un diagrama de bloques de las características de hardware y una lista de características adicionales del MCU.

- Timer interno de 8 bits con un preescalador programable de 7 bits.
- Oscilador on-chip
- Mapa de memoria de E/S
- Manejador de interrupciones versátil
- Manipulación de bit
- 24 pines de E/S
- Verificador de bit e instrucciones de bifurcación
- Vectores de interrupción
- Programa bootstrap (arranque) en ROM
- 112 bytes en RAM
- 3776 bytes de EPROM
- 4 canales para conversión de analógico a digital

DIAGRAMA DE BLOQUES DEL MCU 68705R3



Descripción de señales

V_{CC} y V_{SS}

Utilizando estos dos pines se suministra potencia al microcomputador. V_{CC} es +5.00 volts (±0.5 %) y V_{SS} es tierra.

V_{PP}

Este pin se utiliza cuando se programa la EPROM. En operación normal este pin se conecta a V_{CC}.

$\overline{INT}$

Este pin permite realizar una interrupción externa al MCU.

EXTAL, XTAL

Estos pines proveen el control de entrada para el circuito oscilador de reloj del chip. Una combinación de un cristal y una resistencia/capacitor, o una señal externa se conecta a estos pines para proveer un sistema de reloj. (dependiendo de la

colocación del registro de opción enmascarable).

Oscilador RC Con esta opción, se conecta una resistencia al pin del oscilador como se muestra el la figura 1. La relación entre **R** y $\mathbf{f_{osc}}$ se muestra en la figura 2.

Cristal El circuito mostrado en la figura se recomienda cuando se usa un cristal. El cristal y los componentes deben montarse en un espacio tan pequeño y cercano como sea posible a los pines de entrada para evitar la distorsión de salida y comenzar la estabilización del tiempo.

Reloj Externo Debe aplicarse un reloj externo a la entrada **EXTAL** con la entrada **XTAL** conectada a $\mathbf{V_{SS}}$ como se muestra en la figura 1. Esta opción sólo puede utilizarse con la opción de oscilador de cristal seleccionada en el registro de opción enmascarada.

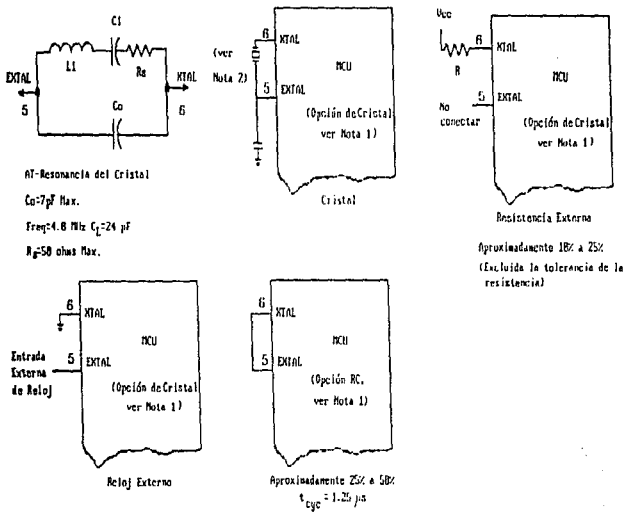


Figura 1. Conexiones del oscilador

Notas:

1. Para el MC68705R3 MOR b7=0 para la opción de cristal y MOR b7=1 para la opción RC. Cuando el pin de entrada del TIMER está en el rango V_{HTP} (en el modo de programación bootstrap de la EPROM), se forza la opción de cristal. Cuando la entrada del TIMER está en o más arriba del valor de V_{cc} la opción generadora de reloj está determinada por el bit 7 del registro de opción enmascarable (CLK).
2. El valor que se recomienda para C_L con un cristal de 4.0 Mhz es de 27 pF como máximo incluyendo la capacitancia del sistema de distribución. Esta es una capacitancia interna en el pin de XTAL de 25 pF aproximadamente. Para cristales de frecuencias diferentes a 4MHz, la capacitancia total en cada pin debe escalarse como el inverso del radio de frecuencia. Por ejemplo, con un cristal de 2 Mhz, se utilizan aproximadamente 50 pF en EXTAL y aproximadamente 25 pF en XTAL. El valor exacto depende de los parámetros del cristal utilizado.

TIMER

Este pin se usa como señal externa de entrada para controlar la circuitería interna timer/contador. Este pin también detecta un nivel alto de voltaje que se utiliza al iniciar el programa bootstrap.

$\overline{\text{RESET}}$

Este pin tiene una entrada de disparador Schmitt y una resistencia de carga. El MCU puede reinicializarse mediante un $\overline{\text{RESET}}$ de bajo nivel.

Líneas de entrada/salida (PA0-PA7, PB0-PB7, PC0-PC7, PD0-PD7)

Estas 32 líneas están colocadas en cuatro puertos de 8 bits (A, B, C y D). Los puertos A, B y C son programables, cada uno, como entradas o como salidas bajo un software de control de los registros de dirección de datos. El puerto D es una combinación de puertos de entrada. Tiene 4 entradas analógicas, más dos entradas de referencia de voltaje cuando se utiliza el convertidor de A/D (PD5/V_{RH}, PD4/V_{RL}), y una entrada $\overline{\text{INT2}}$. Las líneas del puerto D pueden leerse y utilizarse directamente

directamente como entradas binarias. Si una entrada analógica es aplicada al convertidor A/D, los pines de voltaje de referencia deben utilizarse en el modo analógico. Ver programación para más información.

Programación

Programación de Entrada/Salida

Los puertos A, B y C son programables como entrada o salida bajo un software de control del correspondiente registro de dirección de datos de escritura solamente (**DDR**). Las líneas del puerto D sólo son entradas. La programación del puerto E/S se complementa mediante la escritura del correspondiente bit en el puerto **DDR**, con 1 lógico para salida y 0 lógico para entrada. En reset, todos los **DDR** se inicializan con un estado 0 lógico para poner los puertos en modo de entrada. Los registros de salida del puerto no se inicializan en reset y deben escribirse antes de colocar los bits **DDR**.

Cuando se programan como salidas el dato de salida se lee como dato de entrada haciendo caso omiso de los niveles lógicos en los pines de salida, ya que sólo se toma

en cuenta el dato que se encuentre en la salida. El bit de dato de salida siempre puede escribirse. Por lo tanto cualquier escritura a un puerto escribe todos sus bits de datos, así el puerto **DDR** se dispone para entrada . Este puerto de escritura puede usarse para inicializar los registros de datos y evitar salidas indefinidas. Debe tener cuidado al utilizar instrucciones de lectura-modificación-escritura y respetar el que los datos corresponden al pin de nivel si el **DDR** esta como entrada o de salida cuando el **DDR** es una salida. Ver la tabla 1 para funciones E/S y la figura 3 que muestra un circuito típico de un puerto.

Tabla 1. Funciones de E/S

Bit del Registro de Dirección de Datos	Dato de Salida Bit	Salida Estado	Entrada al MCU
1	0	0	0
1	1	1	1
0	X	Hi-Z**	Pin

**Puertos B y C son puertos de tres estados. El puerto A tiene un dispositivo interno de pullup que permite la capacidad de manejar CMOS.

El puerto D provee referencias de voltaje y el multiplexado de entradas analógicas. Las lineas V_{RL} y V_{RH} están conectadas internamente al convertidor A/D. El puerto D siempre se utiliza como entradas digitales, pero para entradas analógicas,

el V_{RH} y el V_{RL} deben conectarse a los voltajes de referencia apropiados.

NOTA

Las instrucciones de lectura-modificación-escritura no deberán usarse cuando se escriba en el DDR siempre que se lea como unos.

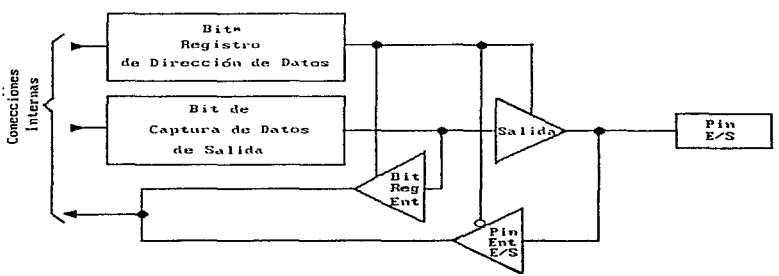


Figura 3. Circuito típico del circuito de E/S y configuración del registro

Memoria

El MCU es capaz de direccionar 4096 bytes de memoria y registros de E/S. El mapa de memoria se muestra en la figura 4. La EPROM consta de las siguientes localidades : ROM bootstrap, RAM, un registro de opción enmascarable (MOR), un registro de control de programa, un registro misceláneo, registros de control A/D, y E/S.

Los vectores de interrupción se localizan desde $FF8 hasta $FFF. El bootstrap es un programa en la ROM que permite alojar programas del MCU en su propia EPROM. El área de la pila se usa durante el procesamiento de una interrupción o llamada a subrutina para salvar el estado del CPU. El apuntador a pila se decrementa durante un push y se incrementa durante un pull. Ver interrupciones para más información.

NOTA

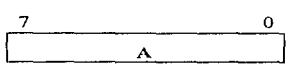
Al utilizar el área de la pila para almacenar datos o localidades de trabajo temporales se requiere tener cuidado para prevenir una sobreescritura.

Registros

El MCU contiene los registros que se describen a continuación.

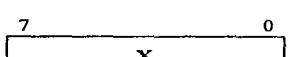
Acumulador(A)

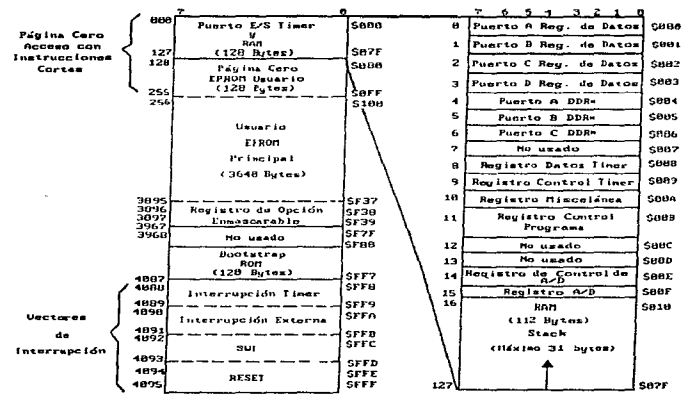
El acumulador es un registro de propósito general de 8 bits que se utiliza para mantener operandos, resultados y cálculos aritméticos o de manipulación de datos.



Registro índice (X)

El registro índice es un registro de 8 bits que se usa en el modo de direccionamiento indexado. Contiene un valor de 8 bits que pueden sumarse a un valor inmediato de 8 o 16 bits para crear una dirección efectiva. El registro índice también puede utilizarse como un área de almacenamiento temporal.



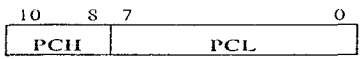


* Precaución: Los registros de dirección de datos (DDRs) únicamente son de escritura, y se leen como $FF.

Figura 4. Mapa de memoria

Contador de Programa (PC)

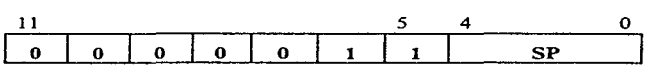
El contador de programa es un registro de 12 bits que contiene la dirección del siguiente byte a ejecutar.



Apuntador de pila (Stack Pointer (SP))

El apuntador a pila es un registro de 12 bits que contiene la dirección de la siguiente localidad libre en la pila. Durante una instrucción de reset al MCU o un reset del apuntador (RSP) el SP se coloca en la localidad $07F. Entonces el SP se decrementa y el dato se coloca dentro de la pila y se incrementa cuando el dato se saca de la pila.

El séptimo bit más significativo se encuentra permanentemente en 0000011. Las subrutinas e interrupciones necesitarán estar debajo de la localidad $061 (31 bytes máximo), como el programa lo permita para utilizar 15 los niveles de llamado a subrutinas (menos si se permiten interrupciones).



Registro de códigos de condición

El registro de códigos de condición es un registro de 5 bits en el cual cuatro bits se utilizan para indicar los resultados de la instrucción ejecutada. Estos bits pueden probarse individualmente con un programa, y se pueden tomar acciones específicas como resultado de su estado. Cada bit se explica en los siguientes párrafos.

4	3	2	1	0
H	I	N	Z	C

Half Carry (H)

Este bit se coloca durante las operaciones ADD y ADC para indicar que una carga ocurre entre los bits 3 y 4.

Interrupción (I)

Cuando este bit se coloca el timer y el interruptor externo se enmascara (deshabilitado). Si ocurre una interrupción externa cuando el bit se coloca, la interrupción se captura y se procesa tan pronto como el bit de interrupción se limpia.

Negativo (N)

Cuando se coloca, este bit indica que el resultado del último dato aritmético, lógico, o de manipulación fue negativo (el bit 7 en los resultados es un 1 lógico).

Cero (Z)

Cuando se coloca, este bit indica que el resultado del último dato aritmético, lógico, o de manipulación fue cero.

Acarreo/Borrow (C)

Cuando se carga este bit indica que un acarreo o préstamo salió de la unidad aritmética y lógica (ALU) y que ocurrió en la última operación aritmética. Sin embargo, este bit se ve afectado durante la prueba de bit e instrucciones de bifurcación, y durante desplazamientos y rotaciones.

RESETS

El MCU puede reinicializarse de dos formas por la inicialización de encendido y por una entrada externa de reset ($\overline{\text{RESET}}$). La entrada de $\overline{\text{RESET}}$ consta

principalmente de un disparador Schmitt que sensa un nivel lógico bajo de la línea.

Reset de encendido (Power On Reset)

Un reset interno se genera mediante un encendido que permite que el generador de reloj interno se estabilice. El reset de encendido se utiliza estrictamente para las condiciones de encendido y no deberá utilizarse para detectar cualquier caída en la alimentación de voltaje. Un retardo de t_{RHL} de milisegundos se requiere antes de que la entrada del $\overline{\text{RESET}}$ sea un nivel alto. Conectando un capacitor a la entrada del $\overline{\text{RESET}}$ (figura 5) se provee un retardo suficiente.

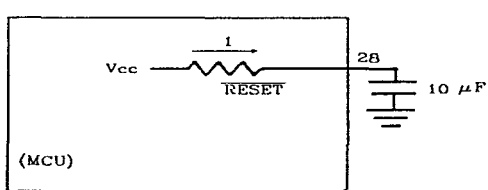


Figura 5. Circuito de retardo del encendido del reset

Entrada de $\overline{\text{RESET}}$ externa

El MCU se reinicializa con un 0 lógico que se aplica en la entrada del $\overline{\text{RESET}}$ por un largo período (t_{CYC}). Bajo este tipo de reset los interruptores de los disparadores Schmitt se apagan para un V_{IRES} para proveer un voltaje de reset interno.

Interrupciones

El MCU puede interrumpirse por cuatro caminos diferentes:

(1) A través de una interrupción externa $\overline{\text{INT}}$ mediante un pin de entrada,

(2) con un requerimiento de interrupción de un timer interno,

(3) usando instrucciones de interrupción mediante software (SWI), o

(4) mediante el puerto externo D ($\overline{\text{INT2}}$) que es un pin de entrada.

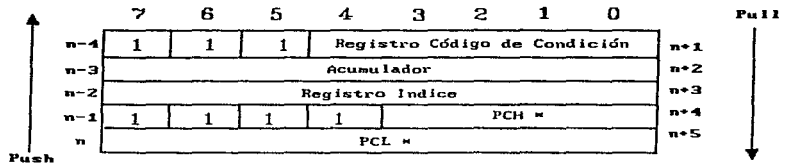


Figura 6. Orden de la pila de interrupción
* Para llamadas a subrutina, sólo PCH y PCL se almacenan

Las interrupciones causan que los registros de proceso se salven en una pila y en una interrupción enmascarada (bit I) previniendo a interrupciones adicionales. Las instrucciones RTI causan que los registros del CPU se recuperen de la pila cuando se resuma su proceso normal. La orden de almacenamiento se muestra en la figura. 6.

Cuando no se requiera el $\overline{\text{RESET}}$, las interrupciones del hardware harán que no se ejecuten instrucciones pero se consideran como pendientes hasta que la instrucción actual se complete.

NOTA

La instrucción que se ejecuta es considerada como la única operando.

Cuando la instrucción se completa el procesador verifica todas las interrupciones pendientes del hardware y, si se desenmascara (el **bit I** es limpiado) se procede con el proceso de interrupción: en otro caso, la siguiente instrucción se llama y se ejecuta. Las interrupciones enmascaradas son llamadas mediante un servicio de interrupción posterior. El estado de interrupción del timer se limpia después de desenmascarar la interrupción cuando la interrupción no se carga.

Si ambos, una interrupción externa y una interrupción de timer, están pendientes al final de la ejecución de alguna instrucción, primero se da servicio a la interrupción externa. El SWI se ejecuta como cualquier otra instrucción sin reparar en la activación del **bit I**. Ver figura 7 para la secuencia de procesamiento de las instrucciones de reset y de interrupción.

Interrupciones de timer

Si el bit de enmascaramiento (TCR6) se limpia, entonces, cada vez que el timer

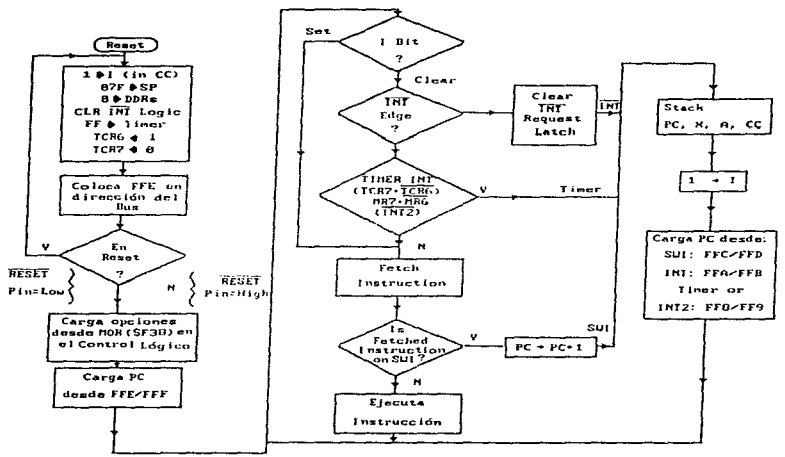


Figura 7. Diagrama de flujo del procesamiento de interrupción y Reset

se decrementa hacia cero (transiciones desde $01 a $00), una interrupción se genera. El proceso de la interrupción solo se genera si el bit de enmascaramiento de la interrupción del registro de códigos de condición (CCR) está limpio. Cuando se reconoce la interrupción, el estado actual de la máquina se coloca dentro de la pila y el **bit I** se coloca en el CCR, mientras que las otras interrupciones se dejan pendientes mientras se hace el servicio actual. El contenido del vector de interrupción del timer, contiene la localidad de la rutina de servicio de interrupción del timer, entonces esto se carga dentro del contador del programa. Al final de la rutina de servicio de interrupción del timer, normalmente el software ejecuta una instrucción RTI la cuál restaura el estado de la máquina y comienza a ejecutar el programa interrumpido. El bit de estado de interrupción del timer solo puede limpiarse mediante software.

Interrupciones externas

La interrupción externa se sincroniza internamente y entonces es capturada en el flanco de bajada de $\overline{\text{INT}}$ e $\overline{\text{INT2}}$. Limpiando el **bit I** se habilita la interrupción externa. La interrupción $\overline{\text{INT2}}$ tiene un bit de requerimiento de interrupción (**bit 7**) y un bit de enmascaramiento (**bit 6**) en el registro misceláneo (**MR**). La interrupción

$\overline{\text{INT2}}$ se inhibe cuando el bit de enmascaramiento se coloca. La $\overline{\text{INT2}}$ siempre se lee como una entrada digital en el puerto D. Los bits de requerimiento de interrupción de timer y la $\overline{\text{INT2}}$, si se colocan, causan que el MCU procese una interrupción cuando el bit del registro de código de condición I se limpia. Los siguientes párrafos describen dos circuitos típicos de interrupción externa.

Interrupción de Cruce por Cero

Una señal de entrada senoidal (f_{INT} máxima) puede usarse para generar una interrupción externa (ver figura 8a) para usarlo como un detector de cruce por cero (para transiciones negativas de la sinusoidal en "ac"). Este tipo de circuito permite aplicaciones tales como rutinas de servicio de tiempo real y de atracción/repulsión de dispositivos de control de potencia de "ac". Una rectificación de onda completa fuera del chip provee una interrupción en cada cruce por cero de la señal de "ac" y con lo cual permite un reloj de 2f.

Interrupción por Señal Digital

Con este tipo de circuito (figura 8b), el pin $\overline{\text{INT}}$ puede manipularse mediante

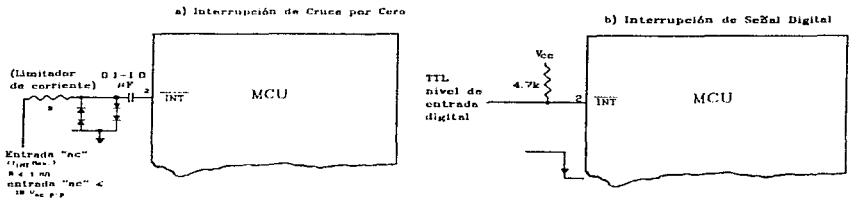


Figura 8. Circuitos típicos de interrupción

una señal digital. La frecuencia máxima de una señal puede reconocerse mediante el **TIMER** o el pin $\overline{INT}$ lógicamente esto depende del parámetro etiquetado t_{WL}, t_{WH}. Ver **TIMER** para más información.

Interrupción de Software (SWI)

La **SWI** es una instrucción ejecutable que se ejecuta no obstante el estado del **bit I** en el **CCR**. Si el **bit I** es cero, **SWI** se ejecuta después de otras interrupciones. La ejecución de **SWI** es similar a las interrupciones por hardware.

Modos de Operación

El MCU puede operar en dos modos distintos. Estos modos son el normal y el bootstrap . Los siguientes párrafos describen los modos.

Modo normal

Este modo es un modo de chip sencillo y se activa cuando las siguientes condiciones se encuentran: (1) la línea $\overline{\text{RESET}}$ es baja, (2) el pin PC0 está dentro de su rango normal de operación y (3) el pin V_{PP} se conecta al pin V_{CC}. El siguiente borde de subida del pin de $\overline{\text{RESET}}$ causa entonces que esta parte entre al modo normal.

Bootstrap

El modo bootstrap se encuentra si el pin del timer es igual a +12 Volts.

TIMER

El MCU consta de un contador programable por software de 8 bits controlado por un preescalador de 7 bits programado por software. Varias fuentes de reloj pueden

seleccionarse mediante el registro de control de tiempo (**TCR**). El contador de 8 bits puede cargarse bajo el control de un programa y se decrementa hacia cero. Cuando el timer llega a cero, la interrupción de timer requiere un bit (el **bit 7**) que se coloca en el registro de control timer (**TCR**). Ver figura 9 (diagrama de bloques del timer).

La interrupción timer puede enmascararse (deshabilitarse) colocando el bit de la interrupción timer enmascarada (**bit 6**) en el **TCR**. Cuando el **bit 1** en el registro de código de condición se limpia, y el **bit 6** del TCR se limpia, el procesador recibe la interrupción. El MCU responde a esta interrupción: 1) salvando el estado presente del CPU en la pila, 2) localizando el vector de interrupción del temporizador, y 3) ejecutando la rutina de interrupción. La interrupción del temporizador requiere un bit que debe limpiarse mediante el software. Ver resets e interrupciones para más información. El preescalador es un divisor de 7 bits el cual se utiliza para extender a su máxima longitud el temporizador. Para evitar errores de truncamiento, el preescalador se limpia cuando el **bit 3** del TCR se coloca en un 1 lógico, sin embargo, el **bit 3** TCR siempre se lee como un 0 lógico de lectura-modificación-escritura.

El temporizador continua el conteo pasando por cero, decreciendo desde $00 hasta $FF, y continua el conteo descendente. El contador puede leerse en cualquier

momento mediante la lectura del registro de datos del temporizador (**TDR**). Esto permite un programa para determinar la longitud del tiempo cuando una interrupción del temporizador ocurre, sin alterar el proceso de conteo. **TDR** no se afecta con el reset.

Modo controlador de software

Este modo se selecciona cuando TOPT (bit 6) en el MOR se programa como cero. La entrada del preescalador puede configurarse para tres modos de operación diferentes más el modo deshabilitado, dependiendo del valor escrito en TCR los bits de control 4 y 5 (TIE y TIN). Los siguientes párrafos describen los diferentes modos.

Modo 1 de Entrada al Timer

Cuando **TIE** y **TIN** se programan a cero, la entrada al timer es el reloj interno (fase 2) y el pin de entrada al timer se deshabilita. El modo de reloj interno, puede utilizarse para generar interrupciones periódicas, tan bien como una referencia para frecuencia y medición de eventos.

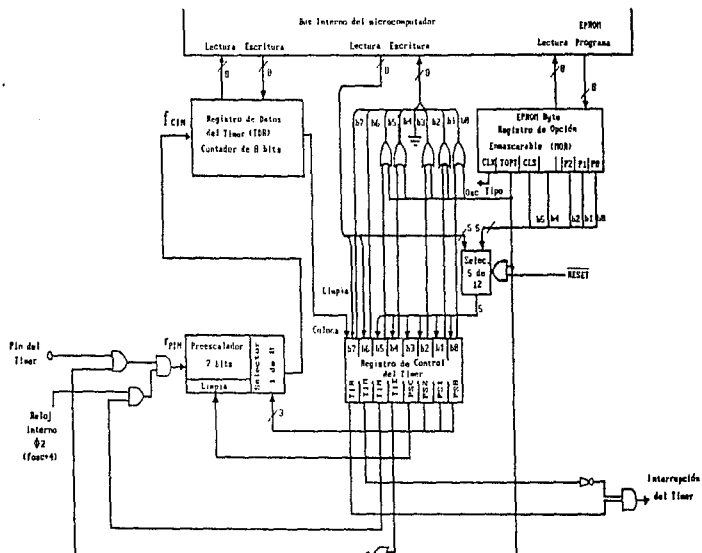


Figura 9. Diagrama de bloques del temporizador

Modo 2 de Entrada al Timer

Cuando **TIE=1** y **TIN=0**, el reloj interno y las señales de entrada al temporizador son colocadas en una compuerta lógica "AND" para formar la entrada a este. Este modo puede utilizarse para medir el ancho de pulsos externos. El flanco alto del pulso externo dispara al reloj interno por la duración del pulso externo. La precisión del contador es ±1.

Modo 3 de Entrada al Timer

Cuando **TIE=0** y **TIN=1**, se aplica una entrada no preescalada en frecuencia al preescalador y se deshabilita el timer.

Modo 4 de Entrada al Timer

Cuando **TIE=TIN=1**, la entrada del timer es desde el reloj externo. El reloj externo puede usarse para contar eventos externos, si este es provisto de una frecuencia externa para generar interrupciones periódicas. La frecuencia externa de entrada debe ser menor o igual que $f_{OSC}/8$.

Modo controlado por MOR

Este modo se selecciona cuando **TOPT (bit 6)** en el **MOR** se programa a 1 lógico. Los circuitos timer son los mismos como se describieron en MODE software MOR. Los niveles lógicos de **TCR bits 0, 1, 2 y 5** se determinan durante la programación de la EPROM por los mismos bits en el **MOR**. Aún más, los **bits 0, 1, 2 y 5** en el **MOR** controlan la división del preescalador y la selección del reloj del timer. Los bits **TIE (bit 4)** y **PSC (bit 3)** en el **TCR** se colocan a 1 lógico en el modo controlado por **MOR**. **TIM (bit 6)** y **TIR (bit 7)** se controlan mediante el contador y software.

Registro controlador del timer (TRC) S009

Este es un registro de 8 bits que controla varias funciones tales como configurar el modo de operación, establecer la razón del preescalador y generar la señal requerida para la interrupción del timer. Todos los bits son de lectura/escritura excepto el **bit 3**; cuando **TOPT=0**, el **TCR** se controla por software.

7	6	5	4	3	2	1	0
TIR	**TIM**	*	1	**PSC**	*	*	*

TCR con MOR TOPT=0

7	6	5	4	3	2	1	0
TIR	TIM	TIN	TIE	PSC	PS2	PS1	PS0
RESET							
0	1	U	U	U	U	U	U

* El valor correspondiente a los bits en MOR se escriben durante el $\overline{\text{RESET}}$. Estos bits siempre se leen como "uno".

U Indeterminado.

TIR - Petición de interrupción por timer

Se usa para indicar la interrupción del timer cuando este es un uno lógico.

1 Se coloca cuando el registro de datos del timer cambia todo a ceros.

0 Se limpia mediante el reset externo, power on reset o bajo el control de un programa.

TIM Interrupción enmascarada del timer

Se usa para inhibir la interrupción del timer

1 Inhibe interrupción

0 Habilita interrupción

TIN Externo o interno.

Selecciona la fuente de entrada del reloj

1 Selecciona reloj externo

0 Selecciona reloj interno ($f_{osc}/4$)

TIE Habilita el timer externo

Se usa para habilitar el pin que corresponde al timer externo. Cuando **TOPT=1,** TIE siempre es un uno lógico.

0 Habilita el pin del timer externo

1 Deshabilita el pin del timer externo

PSC Limpia el preescalador. Sólo escribe un bit. Escribiendo un 1 este bit reinicializa el preescalador a cero. Una lectura a esta localidad siempre indica

un cero cuando TOPT=0. Cuando TOPT=1, este bit puede leerse como un uno lógico y no tiene efecto sobre el preescalador. Esta localidad siempre indica un cero.

PS2, PS1, PS0 - Limpia el preescalador.

Decodifica la selección de una de las 8 salidas del preescalador

PS2	PS1	PS0	División Por
0	0	0	1 (Bypass Prescaler)
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

NOTA

Cuando se cambian los bits PS mediante software, el bit PSC debe escribirse con un uno en el mismo ciclo de escritura para limpiar el preescalador. Cambiar los bits

PS sin limpiar el preescalador puede causar que el preescalador quede truncado.

Registro de opción enmascarable (MOR) SF38

El MOR se implementa en la EPROM y contiene todos sus bits en ceros previos a la programación. Este registro no se afecta con el reset. Los bits MOR se describen en los siguientes párrafos.

7	6	5	4	3	2	1	0
CLK	TOPT	CLS			P2	P1	P0

CLK Reloj (de tipo oscilador)

1 Resistor Capacitor (RC)

0 Cristal

TOPT Opción timer

1 Tipo MC6805 R2 timer/preescalador. Todos los bits excepto el 6 y el 7 del TCR son invisibles al usuario.

Los bits 5, 2, 1 y 0 del MOR, determinan el equivalente a las opciones

enmascaradas del MC6805 R2.

0 Todos los bits del **TCR** se implementan como un timer programable de software. El estado de los bits **5, 4, 2, 1 y 0** del **MOR** colocan los valores iniciales de sus respectivos bits **TCR**.

CLS Fuente de Reloj del Timer/Preescalador

1 Pin de timer externo.

0 Reloj interno.

Bit 4

No se use si **TOPT=1**. Coloca el valor inicial de **TIE** en el **TCR** si **TOPT=0**.

1 No utilizado.

0 Coloca el valor inicial en el **TCR**.

Bit 3

No utilizado.

P2, P1, P0

Los niveles lógicos de estos bits se decodifican cuando se selecciona una de las 8 salidas en el preescalador del timer.

Registro del control de programa (PCR)

El PCR es un registro de 8 bits el cual provee el control necesario de bits para programar la EPROM. El programa bootstrap manipula el PCR cuando este esta programando, al usuario no le concierne saber más aplicaciones del PCR.

7	6	5	4	3	2	1	0
1	1	1	1	1	$\overline{VPON}$	$\overline{PGE}$	$\overline{PLE}$
RESET:							
U	U	U	U	U	U	1	1

$\overline{PLE}$ Habilita la captura del programa

Controla las direcciones y el comienzo de datos se cierra en la EPROM. Durante el reset se coloca, pero puede limpiarse en cualquier momento.

1 Lee EPROM

0 Captura la dirección y los datos sobre la EPROM.

$\overline{\text{PGE}}$ Habilita la programación

Habilita la programación de la EPROM. Puede colocarse cuando cambia la dirección y el dato.

Se coloca durante el reset.

1 Inhibe la programación de la EPROM

0 Habilita la programación de la EPROM (si $\overline{\text{PLE}}$ es baja)

$\overline{\text{VPON}}$ - VPP encendido

Un sólo bit de lectura indica un alto voltaje en el pin V_{PP}. Cuando se coloca un "1", se desconecta $\overline{\text{PGE}}$ y $\overline{\text{PLE}}$ del chip.

1 No hay alto voltaje en el pin V_{PP}

0 Alto voltaje

NOTA

$\overline{VPON}$ en 0 no indica que el nivel V_{PP} es correcto para programar. Se usa como una sincronización segura para el usuario en el modo de operación normal.

$\overline{VPON}$	$\overline{PGE}$	$\overline{PLE}$	Condiciones de Programación
0	0	0	Modo de Programación (Byte para programar la EPROM)
1	0	0	$\overline{PGE}$ y $\overline{PLE}$ desabilitados desde el sistema
0	1	0	Deshabilitador de la programación (Latch Address and Data in EPROM)
1	1	0	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados desde el sistema
0	0	1	Estado inválido $\overline{PGE}$=0 si $\overline{PLE}$=0
1	0	1	Estado inválido $\overline{PGE}$=0 si $\overline{PLE}$=0
0	1	1	Alto Voltaje en V_{PP}
1	1	1	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados desde el sistema (modo de operación)

Convertidor Analógico Digital

En el chip reside un Convertidor Analógico Digital de 8 bit (A/D) el convertidor utiliza una técnica de aproximaciones sucesivas como se muestra en la figura 10.

Cuatro entradas externas analógicas pueden ser conectadas al A/D vía el puerto D.

Cuatro canales internos analógicos (V_{RH}-V_{RL}, V_{RH}-$V_{RL/2}$, V_{RH}-$V_{RL/4}$ y V_{RL}) son seleccionados para calibración. La precisión de estos cambios internos no se encuentren en las especificaciones precisas de canales externos.

La selección del Multiplexor es controlada por el registro de control del A/D (**ACR**) los bits **0**, **1** y **2**. Consulte a la tabla 2 para la selección del multiplexor. El **ACR** es mostrado en la figura 10. El convertidor utiliza 30 ciclos de máquina para completar una conversión de una entrada analógica muestreada. Cuando la conversión es completada, el valor digital es colocado en el registro de resultados del A/D (**ARR**), la bandera de la conversión es activada con la selección de la entrada y se muestrea nuevamente y una nueva conversión comienza. Cuando el **ACR7** es limpiado la conversión en progreso es abortada y la entrada seleccionada, es retenida internamente, es muestreada en cinco ciclos de máquina.

El convertidor emplea V_{RH} y V_{RL} como voltajes de referencia. Una entrada de voltaje igual a más grande que V_{RH} se convierte en **$FF**. Una entrada de voltaje igual o más pequeña que V_{RL} pero más grande que V_{SS} se convierte a **$00**.

Los rangos máximos y mínimos no deben ser excedidos cada fuente analógica

de entrada podría usar V_{RH} como un suministro de voltaje y podría ser referido a V_{RL} para la conversión ratiometrico. Para mantener una precisión total en el A/D necesita de los tres requerimientos siguientes:

(1) V_{RH} será igual o más pequeña que V_{CC}

(2) V_{RL} será igual o más grande que V_{SS} pero más grande que las máximas especificaciones y

(3) V_{RH}-V_{RL} será igual o más grande que 4 volts.

El A/D tiene incorporado 1/2 offset LSB destinado a reducir la magnitud de error a ±1/2 LSB más que +0, -1 LSB sin offset. Esto implica que, ignora errores, la transición puntual de $00 a $01 ocurre en 1/2 LSB por encima de V_{RL}. Similarmente la transición de $FE a $FF ocurre 1-1/2 LSB por abajo de V_{RH} idealmente.

Tabla 2. Selección del Multiplexor de entrada del A/D

Registro de Control A/D			Entrada Seleccionada	Salida A/D (Hex)		
ACR2	ACR1	ACR0		Min	Tip	Max
000			AN0			
001			AN1			
010			AN2			

Registro de Control A/D			Entrada Seleccionada	Salida A/D (Hex)		
ACR2	ACR1	ACR0		Min	Tip	Max
011			AN3			
100			VRH*	FE	FF	FF
101			VRL*	00	00	01
110			VRH/4	3F	40	41
111			VRH/2	7F	80	81

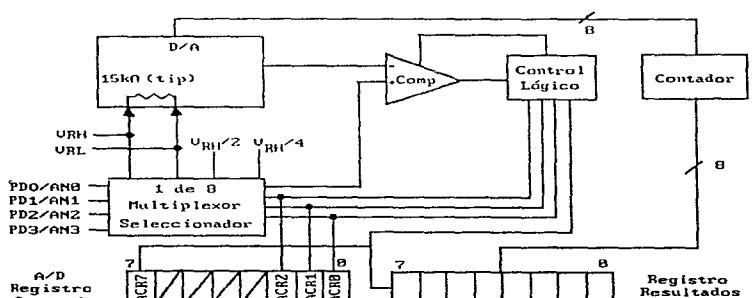


Figura 10. Diagrama de Bloques del Convertidor A/D

Conjunto de Instrucciones

El MCU tiene un conjunto de 59 instrucciones básicas las cuales pueden dividirse en 5 tipos diferentes: registro/memoria, lectura-modificación-escritura, bifurcación, manipulación de bits, y control. Los siguientes párrafos explican cada tipo:

Instrucciones Registro / Memoria

Muchas de estas instrucciones usan dos operandos. Un operando es el acumulador o el registro índice. El otro operando se obtiene de memoria usando uno de los modos de direccionamiento. Las instrucciones de salto incondicional (JMP) y salto a subrutina (JSR) no tienen operando de registro. Veamos la siguiente lista de instrucciones.

Función	Mnemónico
Carga A desde la memoria	LDA
Carga X desde la memoria	LDX
Almacena A en la memoria	STA
Almacena X en la memoria	STX

Función	Mnemónico
Suma la memoria al contenido de A	ADD
Suma la memoria y el acarreo al contenido de A	ADC
Resta a la memoria	SUB
Resta a la memoria desde A con préstamo	SBC
Realiza una AND de la memoria con A	AND
Realiza una OR de la memoria con A	ORA
Realiza una OR Exclusiva de la memoria con A	EOR
Comparación aritmética de A con la memoria	CMP
Comparación aritmética de X con la memoria	CPX
Bit de verificación de la memoria con A (comparación lógica)	BIT
Salto incondicional	JMP
Salto a subrutina	JSR

Instrucciones lectura-modificación-escritura

Estas instrucciones leen una localidad de memoria o un registro, modifican o verifican estos contenidos y escriben el valor modificado en memoria o en un registro. Las instrucciones de verificación para no negativa o cero (TST) son una excepción de la lectura-modificación-escritura ya que no modifican el valor. Veamos la siguiente lista de instrucciones.

Función	Mnemónico
Incremento	INC
Decremento	DEC
Limpia	CLR
Complemento	COM
Negado (Segundo complemento)	NEG
Rotación a la izquierda con acarreo	ROL
Rotación a la derecha con acarreo	ROR
Corrimiento lógico a la izquierda	LSL
Corrimiento lógico a la derecha	LSR
Corrimiento aritmético a la derecha	ASR
Verifica para negativo o cero	TST

Instrucciones de bifurcación

Este conjunto de instrucciones bifurca si se da una condición en particular; en otro caso la operación no se realiza. Las instrucciones de bifurcación son de 2 bytes. Veamos la siguiente lista de instrucciones:

Función	Mnemónico
Bifurca siempre	BRA
Nunca bifurca	BRN

Función	Mnemónico
Bifurca si es alto	BHI
Bifurca si es bajo o igual	BLS
Bifurca si el acarreo está limpio	BCC
(Bifurca si es alto o igual)	(BHS)
Bifurca si se coloca el acarreo	BCS
Bifurca si es bajo	(BLO)
Bifurca si no es igual	BNE
Bifurca si es igual	BEQ
Bifurca si el acarreo medio está limpio	BHCC
Bifurca si el acarreo medio se coloca	BHCS
Bifurca si suma	BPL
Bifurca si disminuye	BMI
Bifurca si el bit de interrupción enmascarable se limpia	BMC
Bifurca si el bit de interrupción enmascarable se coloca	BMS
Bifurca si la línea de interrupción es baja	BIL
Bifurca si la línea de interrupción es alta	BIH
Bifurca a subrutina	BSR

Instrucciones de control

Estas instrucciones registran las instrucciones de referencia y se usan para controlar la operación del procesador durante la ejecución de un programa. Veamos la siguiente lista.

Función	Mnemónico
Transfiere A a X	TAX
Transfiere X a A	TXA
Coloca el bit de acarreo	SEC
Limpia el bit de acarreo	CLC
Coloca el bit de interrupción enmascarada	SEI
Limpia el bit de interrupción enmascarada	CLI
Interrupción por Software	SWI
Retorno de subrutina	RTS
Retorno de interrupción	RTI
Reinicializa el Stack Pointer	RSP
No Opera	NOP

Instrucciones de manipulación de bits

El MCU es capaz de colocar o limpiar cualquier bit que resida en los primeros 256 bytes del espacio de memoria, donde todos los registros de puerto, puertos DDR,

timer, control del timer y de la RAM. Una característica adicional permite la verificación de software y bifurcación de un estado de. cualquier bit en estas 256 localidades. Las funciones de limpiar bit y verificar bit y de bifurcación se implementan con una simple instrucción. Para instrucciones de verificación y bifurcación, el valor del bit verificado también se coloca en el bit de acarreo del registro de código de condición. Veamos la siguiente lista.

Función	Mnemónico
Bifurca si el bit n se coloca	BRSET n (n = 0...7)
Bifurca si el bit n se limpia	BRCLR n (n = 0...7)
Coloca el bit n	BSET n (n = 0...7)
Limpia el bit n	BCLR n (n = 0...7)

Modos de direccionamiento

El MCU usa 10 modos de direccionamiento diferentes para darle al programador la oportunidad de optimizar el código para cualquier situación. Los diversos modos de direccionamiento indexado hacen esto posible al localizar tablas de datos, tablas de conversión de código y tablas de escalamiento en cualquier lugar del espacio de

memoria. Los accesos cortos indexados simplemente son instrucciones de un byte, mientras que las instrucciones largas son de 3 bytes y permiten accesar tablas de memoria. También se incluyen direccionamientos absolutos cortos y largos. Las instrucciones de direccionamiento directo de 2 bytes accesan todos los bytes de datos en más aplicaciones. El direccionamiento extendido permite instrucciones de salto que alcanzan toda la memoria.

El término dirección efectiva (EA) se usa para describir varios modos de direccionamiento. La EA se define como la dirección desde la cual el argumento de una instrucción se carga o almacena.

Inmediato

En el modo de direccionamiento inmediato, el operando está contenido en el byte inmediato siguiente al código de operación. El modo de direccionamiento inmediato se usa para accesar constantes que no cambian durante la ejecución del programa (por ejemplo una constante que se usa para inicializar un ciclo de conteo).

Directo

En el modo de direccionamiento directo, la dirección efectiva del argumento está contenida en un byte que le sigue al byte del código de operación. El direccionamiento directo permite el uso de la dirección directamente en los 256 bytes bajos en memoria con una sola instrucción de 2 bytes.

Extendido

En el modo de direccionamiento extendido, la dirección efectiva del argumento está contenida en los 2 bytes siguientes al byte del código de operación. Las instrucciones con modo de direccionamiento extendido son capaces de referenciar argumentos en cualquier lugar de la memoria con una instrucción de 3 bytes.

Relativo

El modo de direccionamiento relativo sólo se usa en instrucciones de bifurcación. En el direccionamiento relativo, el contenido de los 8 bits llamados byte (el offset), siguientes al código de operación, se añaden al PC si y sólo si, las condiciones de bifurcación son ciertas.

En otro caso, el control se pasa a la siguiente instrucción. El largo del modo de

direccionamiento relativo va desde -126 a +129 de la dirección del código de operación.

Indexado sin offset

En el modo de direccionamiento indexado, sin offset, la dirección efectiva del argumento está contenida en los 8 bits del registro índice. Este modo de direccionamiento puede accesar las primeras 256 localidades de memoria. Estas instrucciones son de un byte de longitud. Este modo es utilizado para mover un apuntador a una tabla o para mantener la dirección de una referencia de RAM frecuente o una localidad de E/S.

Indexado con 8 bits de offset

En el modo de direccionamiento indexado con 8 bits de offset, la dirección efectiva es la suma de los contenidos de los registros índice de 8 bits sin signo y del byte sin signo que precede al código de operación. Este modo de direccionamiento se utiliza para seleccionar el elemento K-ésimo en un n elemento de una tabla. Con instrucciones de 2 bytes, k típicamente puede estar en X con la dirección del comienzo

de la tabla en la instrucción. Así, las tablas pueden comenzar en cualquier lugar dentro de las 256 localidades de memoria y pueden extenderse tan lejos como la localidad 510 ($1FE es la última localidad en la cual la instrucción puede comenzar).

Indexado con 16 bits de offset

En el modo de direccionamiento indexado con 16 bits de offset, la dirección efectiva es la suma de los contenidos del registro índice sin signo de 8 bits y de los dos bytes sin signo siguientes al código de operación. Este modo de direccionamiento puede utilizarse de forma similar al modo indexado con 8 bits de offset, excepto que este utiliza instrucciones de 3 bytes que permiten el uso de tablas en cualquier parte de la memoria.

Bit Set/Clear

En el modo de direccionamiento bit set/clear, el bit que se coloca o limpia es parte del código de operación. El byte siguiente al código de operación especifica la dirección directa del byte en el cual el bit especificado se colocará o se limpiará. Cualquier bit de lectura/escritura en las primeras 256 localidades de memoria,

incluyendo E/S, puede colocarse o limpiarse con una instrucción de 2 bytes.

Precaución

Los correspondientes DDRs para puertos A, B y C sólo son registros de escritura (registros $004 a $005, y $006). Una operación de lectura a estos registros siempre retorna un "1". Dado que BSET y BCLR son funciones de lectura-modificación-escritura no pueden utilizarse para colocar o limpiar un bit DDR. Es recomendable que todos los bits de puertos DDR se escriban utilizando una instrucción de almacenamiento.

Bit prueba y bifurcación

El modo de direccionamiento bit test and branch es una combinación de los modos de direccionamiento directo y relativo. El bit que será verificado y su condición (colocar o limpiar) se incluyen en el código de operación. La dirección del byte a verificar está en el byte inmediatamente después del byte del código de operación. El offset de 8 bits relativo y signado en el tercer byte se suma al PC si el bit especificado se coloca o se limpia en la localidad de memoria correspondiente. Esta instrucción de

3 bytes permite que el programa se bifurque basándose en la condición de cualquier bit leíble en las primeras 256 localidades de memoria. El largo de la bifurcación está desde -125 a +130 desde la dirección del código de operación. El estado del bit verificado se transfiere al bit de acarreo del registro de código de condición.

Inherente

En el modo de direccionamiento inherente, toda la información necesaria para ejecutar la instrucción está contenida en el código de operación. Las operaciones especifican sólo el registro índice o el acumulador tan bien como el control de instrucciones con otros argumentos, se incluyen en este modo. Estas instrucciones son de un byte de longitud.

Especificaciones Eléctricas

Rangos Máximos

Rangos	Símbolo	Valor	Unidad
Voltaje de alimentación	V_{CC}	-0.3 a +7.0	V

Rangos	Símbolo	Valor	Unidad
Voltajes de entrada EPROM Voltaje de programación (Pin VPP) Pin de TIMER (modo normal) Pin TIMER (modo de programación bootstrap) Todos los demás	V_{PP} V_{in} V_{in} V_{in}	-0.3 a +22.0 -0.3 a +7.0 -0.3 a +15.0 -0.3 a +7.0	V V V V
Rango de operación de la temperatura	T_A	T_L a T_H 0 a +70	°C
Rango de temperatura de almacenamiento	T_{stg}	-55 a +150	°C
Temperatura de la unión del chip	T_J	150	°C/W

Características Térmicas

Características	Símbolo	Valor	Unidad
Resistencia Térmica del chip Plastico (Sufijo P) Plastico (Sufijo N) Cerdip (Sufijo S)	θ_{JA}	 50 100 60	°C/W

Consideraciones de Potencia

El promedio de la temperatura de la unión del chip , T_J, en °C puede obtenerse

de la siguiente manera:

$T_J = T_A + (P_D \cdot \theta_{JA})$ ..... (1)

donde:

T_A = Temperatura ambiente, °C.

θ_{JA} = Resistencia térmica del encapsulado,
Unión a temperatura ambiente, °C/W.

P_D = $P_{INT} + P_{I/O}$

P_{INT} = $I_{CC} \times V_{CC}$, Watts - Potencia interna del chip.

$P_{I/O}$ = Disipación de la potencia en los pines de entrada y salida - Determinado por el usuario.

Para más aplicaciones $P_{I/O} < P_{INT}$ y puede ser ignorado. La siguiente es una relación es una aproximación entre P_D y T_J (Si se ignora $P_{I/O}$):

$P_D = K / (T_J + 273\ °C)$ ..... (2)

Resolviendo las ecuaciones (1) y (2) para K, se obtiene:

$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P^2_D$ ..... (3)

donde K es una constante perteneciente a la parte particular. K puede determinarse mediante la ecuación (3) midiendo P_D (en equilibrio) para conocer T_A. Utilizando este

valor de K, los valores de P_D y T_J pueden obtenerse resolviendo iterativamente las ecuaciones (1) y (2) para cualquier valor de T_A.

Características Eléctricas

(Vcc=5.25 +-0.5 VDC, Vss=0 VDC, Vss=0 VDC, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de entrada alto $\overline{\text{RESET}}$(4.75 <= V_{CC} <=5.75) $\overline{\text{INT}}$ (4.75 <= V_{CC} <=5.75) (V_{CC} <4.75)) Todos los demás	V_{IH}	 4.0 Vcc-0.5 4.0 Vcc-0.5 2.0	 - - * * -	 Vcc Vcc Vcc Vcc Vcc	V
Voltaje de entrada alto (Pin TIMER) Modo Timer Modo de programación bootstrap	V_{IH}	 2.0 9.0	 - 12.0	 Vcc 15.0	V
Voltaje de entrada bajo $\overline{\text{RESET}}$ $\overline{\text{INT}}$ Todos los demás	V_{IH}	 -0.3 -0.3 -0.3	 -- ** --	 0.8 1.5 0.8	V

Características	Símbolo	Min	Tipo	Max	Unidad
Disipación interna de energía (No carga al puerto, V_{CC}= 5.25 V, T_A= °C)	P_{INT}	---	450	TBD	mW
Capacitancia de entrada XTAL Todas las demás	C_{in}	 --- ---	 25 10	 -- --	pF
$\overline{INT}$ voltaje de cruce por cero, a través de un capacitor	V_{INT}	2.0	--	4.0	Vacp-p
$\overline{RESET}$ voltaje de histéresis Fuera del voltaje de reinicialización Dentro del voltaje de reinicialización	 V_{IRES}+ V_{IRES}-	 2.1 0.8	 -- --	 4.0 2.0	V
Voltaje de programación (Pin V_{PP}) Programación de la EPROM Modo de operación	V_{PP}*	 20.0 4.0	 21.0 V_{CC}	 22.0 5.75	V
Entrada actual TIMER (V_{in} = 0.4V) $\overline{INT}$ (V_{in} = 0.4V) EXTAL (V_{in}=2.4 V a V_{CC} opción del cristal) (V_{in}=0.4 V opción del cristal) $\overline{RESET}$ (V_{in}=0.8 V) (Capacitor externo de cambio de corriente)	I_{in}	 --- --- --- --- -4.0	 -- 20 -- -- --	 20 50 10 -1600 -40	μA

*V_{PP} es el pin 6 en el MC68705P3 y está conectado a V_{CC} en el modo de operación

normal. En el MC6805P2, el pin 6 es NUM y está conectado a V_{SS} en el modo de operación normal.
**Debido a la oblicuidad interna, esta entrada (cuando no es utilizada) varía aproximadamente entre 2.0 V.

Puerto DC características eléctricas

(V_{CC}=+5.25 +-0.5 Vdc, V_{SS}=0 Vdc, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de Salida bajo, I_{LOAD}=1.6mA	V_{OL}	---	---	0.4	V
Voltaje de salida alto I_{LOAD}= -100µA	V_{OH}	2.4	---	---	V
Voltaje de salida alto I_{LOAD}= -10µA	V_{OH}	V_{CC}-10	---	---	V
Voltaje de entrada alto I_{LOAD}= -300µA (Max)	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo I_{LOAD}= -500µA (Max)	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada (V_{in} = 20 V a V_{cc})	I_{IH}	---	---	-300	µA
Hi-Z estado actual de la entrada (V_{in} = 0.4 V)	I_{IL}	---	---	-500	µA
Puerto B					

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de Salida bajo, $I_{Load} = 3.2\,mA$	V_{OL}	---	---	0.4	V
Voltaje de Salida bajo, $I_{Load} = 10\,mA$ (Sink)	V_{OL}	---	---	1.0	V
Voltaje de salida alto $I_{Load} = -200\,\mu A$	V_{OH}	2.4	---	---	V
Darlington Current Drive (Source) $V_O = 1.5\,V$	I_{OH}	-1.0	---	-10	mA
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada	I_{TSI}	---	2	20	μA
Puerto C					
Voltaje de Salida bajo, $I_{Load} = -1.6\,mA$	V_{OL}	---	---	0.4	V
Voltaje de Salida alto, $I_{Load} = -100\,\mu A$	V_{OH}	2.4	---	---	V
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada	I_{TSI}	---	---	20	μA

Características de encendido (V_{CC}= +5.25 +/- 0.5 Vdc, V_{SS}=0 Vdc, T_A= 0° C a 70° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Frecuencia del Oscilador Normal	f_{OSC}	0.4	---	4.2	MHz
Tiempo del Ciclo de intrucción	t_{cyc}	0.950	---	10	μs
Ancho del pulso de INT o Timer	t_{WL}, t_{WH}	t_{cyc}+250	---	---	ns
Ancho del pulso de RESET	t_{RWI}	t_{cyc}+250	---	---	ns
Tiempo de retardo del RESET (Capacitancia externa = 1.0 μF)	t_{RHL}	100	---	---	ms
Detección de la frecuencia de entrada de cruce por cero INT	f_{INT}	0.03	---	1.0	kHZ
Ciclo duty del reloj externo (EXTAL)	---	40	50	60	%

Características de la operación de programación eléctrica (V_{CC}= +5.25 +/-0.5 Vdc, V_{SS}=0 Vdc, T_A= 20° C a 30° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Voltaje de programación (Pin V_{PP})	V_{PP}	20.0	21.0	22.0	V
Current Supply V_{PP} V_{PP} = 5.25 V V_{PP} = 21.0 V	I_{PP}	 --- ---	 --- ---	 8 30	mA
Frecuencia de oscilación de programación	f_{oscp}	0.9	1.0	1.1	MHz
Voltaje del modo de programación bootstrap (Pin del TIMER) I_{in} = 100	V_{HTP}	9.0	12.0	15.0	V

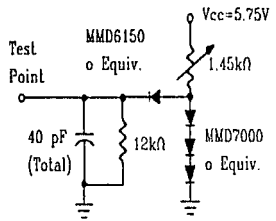


Figura 11. TTL Equivalent Test Load (Port B)

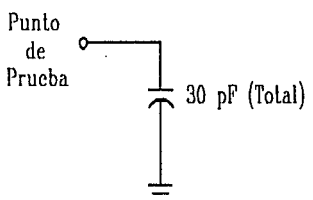


Figura 12. CMOS Equivalent Test Load (Port A)

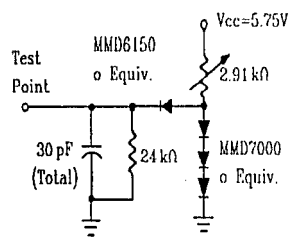


Figura 13. TTL Equivalent Test Load (Ports A and C)

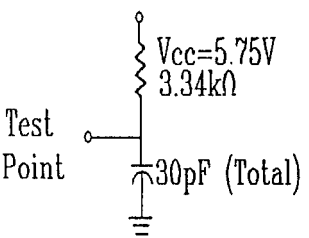


Figura 14. Open-Drain Equivalent Test Load (Port C)

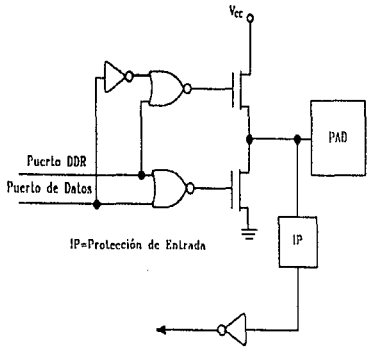


Figura 15. Diagrama Lógico de los Puertos A y C

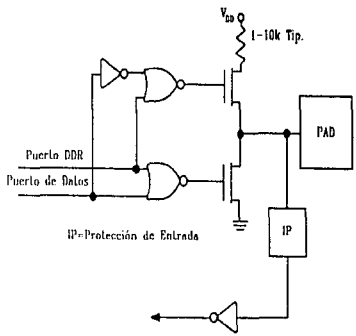


Figura 16. Diagrama Lógico del Puerto B

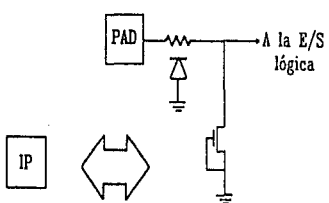


Figura 17. Típica Protección de Entrada

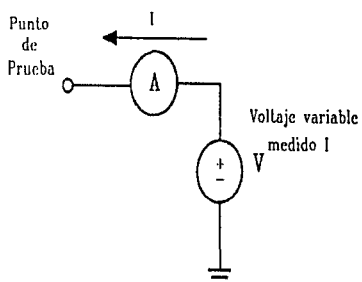


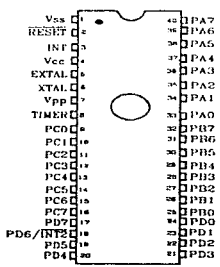
Figura 18. I/O Characteristic Meassurement Circuit

Información Genérica

La siguiente tabla provee información genérica perteneciente al tipo de encapsulado, y a los número del MC para el MC68705P3.

Tabla 3. Información Genérica

Tipo de encapsulado	Frecuencia del reloj interno (MHz)	Temperatura	Número de orden
Encapsulado (Sufijo S)	1.0	0 °C a 70 °C	MC68705R3S
Encapsulado (Sufijo S)	1.0	-40 °C a +85 °C	MC68705R3CS



ASIGNACION DE PINES

MC68705U3

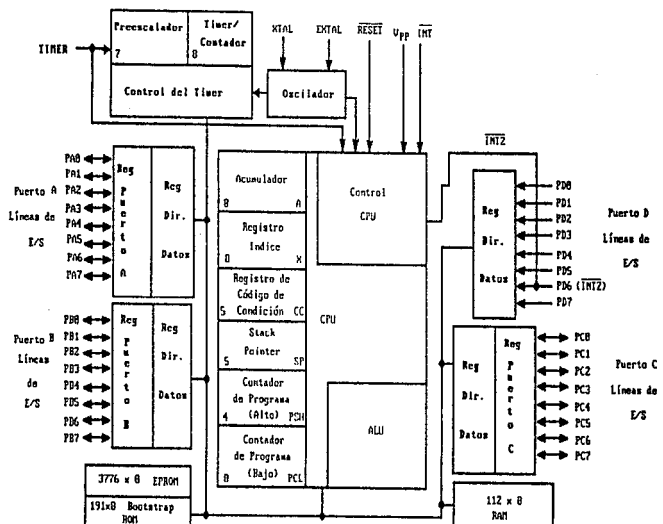
Microcontrolador de 8 bits con EPROM

El MCU MC68705U3 (HMOS) es un miembro de la familia de microcontroladores MC6805 EPROM. La EPROM permite cambiar programas y reducir el volumen de las aplicaciones. El MCU es de bajo costo y tiene una capacidad paralela de E/S con pines programables como entrada o salida.

El diagrama de bloques muestra las características de hardware y se listan enseguida características adicionales disponibles en el MCU.

- Timer interno de 8 bits con un Prescalador programable de 7 bits
- Un circuito oscilador interno.
- Mapa de memoria de E/S.
- Manejo de interrupciones versátil.
- Manipulación de bits.
- 24 pins E/S.
- Instrucciones Bit Test y Branch.
- Vectores de interrupción.
- Programa Bootstrap (arranque) en la ROM.
- 112 Bytes de RAM.
- 3776 Bytes de EPROM.

DIAGRAMA DE BLOQUES DEL MCU MC68705U3



Descripción de las señales

V_{CC} y V_{SS}

La energía es suministrada al microcontrolador usando estas dos terminales. Se alimenta V_{CC} con +5.25 voltios ±0.5% y V_{SS} se conecta a tierra.

V_{PP}

Esta terminal es utilizada al programar la EPROM. En la operación normal esta es conectada a V_{CC}.

$\overline{INT}$

Esta terminal provee la capacidad de interrupciones asíncronas externas al MCU.

EXTAL, XTAL

Estas entradas controlan el reloj interno del circuito oscilador. Un cristal, una

resistencia/capacitor o una señal externa (dependiendo de la opción enmascarada en una colección de registros) son conectados a estos pines para proveer el sistema de reloj.

Circuito oscilador RC Con esta opción, una resistencia es conectada al pin de entrada del oscilador como se muestra en la figura 1. La relación entre R y f_{osc} es mostrada en la figura 2.

Cristal El circuito mostrado en la figura 1 es recomendado cuando usamos un cristal. El cristal y los demás componentes deben ser montados tan cerca como sea posible a las entradas de los pines para minimizar la distorsión en la salida y comenzar el tiempo de estabilización.

Reloj externo Un reloj externo puede ser aplicado a la entrada **EXTAL** y la entrada **XTAL** conectada a V_{SS} como muestra la figura 1. Esta opción selecciona la opción de registro enmascarado.

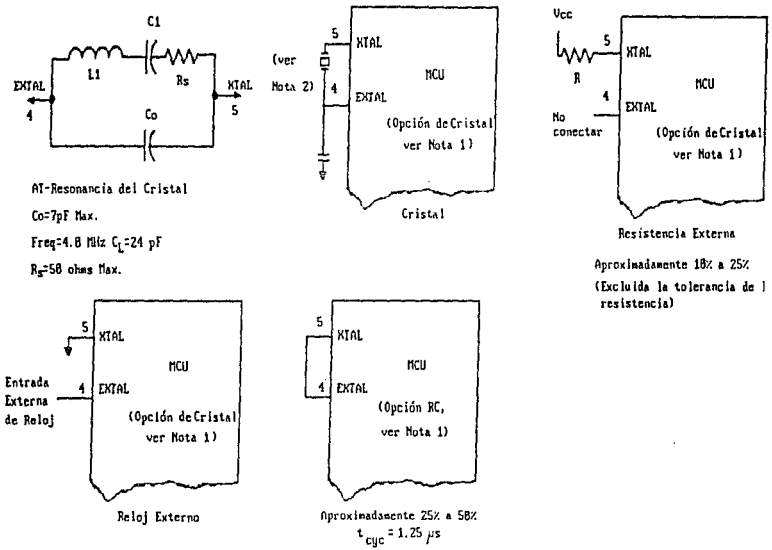


Figura 1. Conexiones del Oscilador

Notas:

1. Cuando el pin de entrada del TIMER está en el rango V_{HTP} (en el modo de programación bootstrap de la EPROM), se forza la opción del cristal.
2. El valor que se recomienda para C_L, con un cristal de 4.0 Mhz es de 27 pF como máximo incluyendo la capacitancia del sistema de distribución. Esta es una capacitancia interna en el pin de XTAL de 25 pF aproximadamente. Para cristales de frecuencias diferentes a 4MHz, la capacitancia total en cada pin debe escalarse como el inverso del radio de frecuencia. Por ejemplo, con un cristal de 2 Mhz, se utilizan aproximadamente 50 pF en EXTAL y aproximadamente 25 pF en XTAL. El valor exacto depende de los parámetros del cristal utilizado.

TIMER

El pin es usado como una entrada externa para controlar el PC (Program Counter contador de programa) interno del circuito. El pin también detecta un nivel alto de voltaje que se utiliza para iniciar el programa "bootstrap".

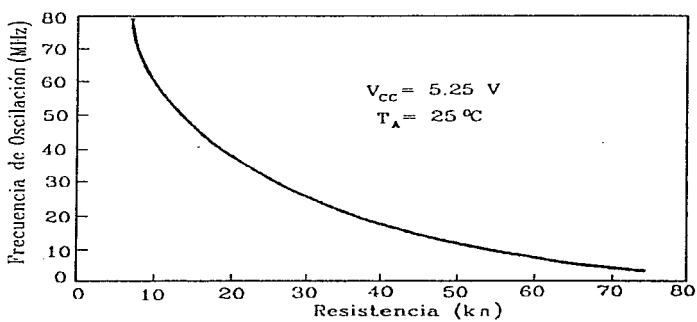


Figura 2. Relación típica entre Frecuencia y Resistencia para la opción del Oscilador RC

$\overline{\text{RESET}}$

Este pin tiene un Schmit trigger de entrada y una carga interna. El MCU puede ser inicializado al poner $\overline{\text{RESET}}$ en bajo.

Líneas de Entrada/Salida (PA0-PA7, PB0-PB7, PC0-PC7, PD0-PD7)

Estas 32 líneas son organizadas en 4 puertos de 8 bits (A, B, C y D). Puertos A, B, y C son programables como entradas o salidas bajo el software de control del registro de dirección de datos (**DDR**). El puerto D es únicamente un puerto de entrada. El puerto D es de 6 bits y también es usado como un segundo interruptor $\overline{\text{INT2}}$.

Programación

Programación de Entrada/Salida

Los pines del puerto A, B, y C son programables como entrada o salida por medio del software de control correspondiente al registro de dirección de datos (**DDR**). El puerto D es solamente de entrada. La programación del puerto de E/S se lleva a cabo escribiendo el bit correspondiente en el puerto **DDR** un uno lógico para salida y

un cero lógico para entrada. En reset, todos los **DDR's** son inicializados en cero lógico poniendo los puertos en el modo de entrada. Los registros de los puertos de salida no son inicializados durante el reset.

Cuando se programan como salidas el dato de salida se lee como dato de entrada haciendo caso omiso de los niveles lógicos en los pines de salida, ya que sólo se toma en cuenta lo que haya a la salida. El dato de salida siempre puede escribirse. Por lo tanto cualquier escritura a un puerto escribe todos sus bits de datos, así el puerto **DDR** se dispone para entrada . Este puerto de escritura puede usarse para inicializar los registros de datos y evitar salidas indefinidas. Debe tener cuidado al utilizar instrucciones de lectura-modificación-escritura y respetar el que los datos corresponden al pin de nivel si el **DDR** está como entrada o de salida cuando el **DDR** es una salida. Ver la tabla 1 para funciones E/S y la figura 3 que muestra un circuito típico de un puerto.

NOTA

Las instrucciones para lectura-modificación-escritura no deberán ser usadas cuando escribimos al DDR, porque el DDR siempre se leen como "uno".

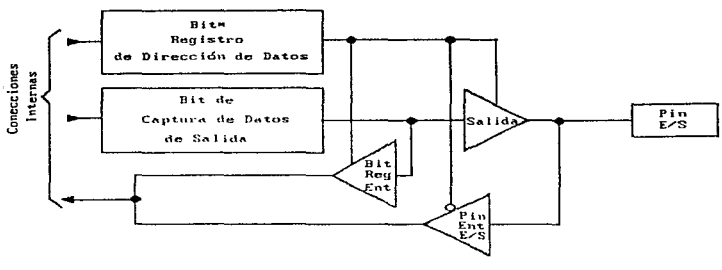


Figura 3. Típica Circuitería de Puertos E/S y Configuración de Registros

Tabla 1. Pines de función de E/S

Bit del Registro de Dirección de Datos	Bit Captura Datos Salida	Estado de la Salida	Entrada al MCU
1	0	0	0
1	1	1	1
0	X	Hi-Z**	Pin

**Los Puertos B y C son puertos de tres estados. El puerto A tiene una carga interna para proveer de energía a los datos en CMOS.

Memoria

El MCU es capaz de direccionar 4096 bytes de memoria y registros de E/S. El mapa de memoria se muestra en la figura 4. Las localidades son utilizadas para usar la EPROM, el bootstrap ROM, la RAM, la opción de registro enmascarado (MOR), un registro de control de programación y E/S. Los vectores de interrupción están localizados de la SFF8 a la SFFF. El bootstrap es un programa enmascarado en la ROM que permite al MCU programar su EPROM.

El área del stack es usado durante el procesamiento de una interrupción o una llamada a subrutina para salvar el estado del MCU. El apuntador al stack se decrementa durante la salida y se incrementa durante la entrada.

NOTA

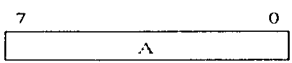
Usando el área del stack para el almacenamientos de datos o trabajos temporales se requiere tener cuidado para prevenir que se haga una sobreescritura debido a una interrupción o llamada a subrutina.

Registros

El MCU contiene los registros que se describen a continuación.

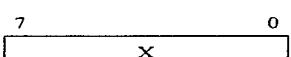
Acumulador (A)

El acumulador es un registro de propósito general de 8 bits usado para retener operandos y resultados de cálculos aritméticos o manipular datos.



Registro Indice X

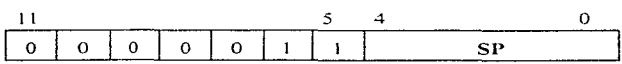
El registro índice es un registro de 8 bits usado para el modo de direccionamiento indexado. Este contiene un valor de 8 bits que puede ser añadido a un valor inmediato de 8 a 16 bits para crear un direccionamiento efectivo. El registro índice puede también ser usado como un almacenamiento temporal.



Stack Pointer SP (Apuntador a la pila)

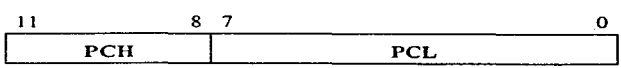
El apuntador al stack es un registro de 12 bits que contiene la dirección de la siguiente localidad libre del stack. Durante una inicialización del MCU o la inicialización de la instrucción del apuntador al stack (**RSP**), el apuntador al stack es colocado en la localidad **S07F**. El apuntador al stack es entonces decrementado cuando un dato es introducido al stack e incrementado cuando un dato es sacado del stack.

Los siete bits más significativos del apuntador al stack son permanentemente puestos como **0000011**. Las subrutinas e interrupciones pueden ser anidadas bajo la localidad **S061** (31 bits como máximo), permite al programador usar arriba de 15 niveles de llamadas a subrutina (menos si las interrupciones son admitidas).



Contador del programa (PC)

El contador del programa es un registro de 12 bits que contiene la dirección del siguiente byte a ser ejecutado.



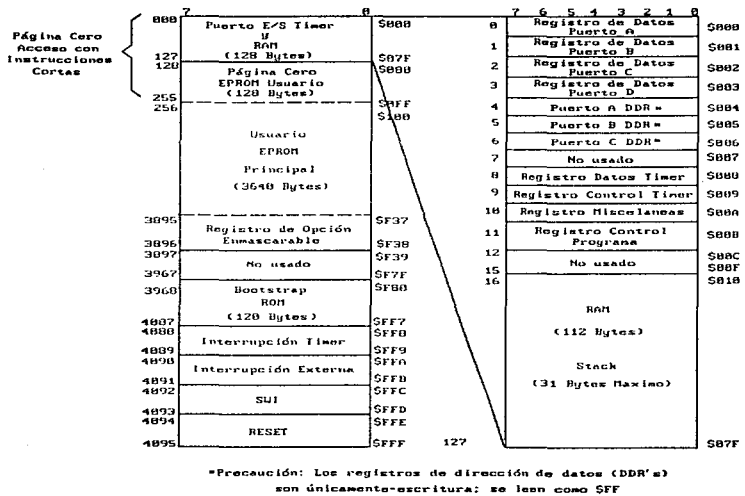


Figura 4. Mapa de Memoria

Registro de Código Condición (CCR)

El Registro de Código de Condición es de 5 bits y 4 bits son usados para indicar el resultado de la instrucción recién ejecutada. Estos bits pueden ser examinados individualmente como un estado del resultado. Cada bit es explicado en los siguientes párrafos.

4	3	2	1	0
H	I	N	Z	C

Half Carry (H) (Medio Acarreo)

Este bit es activado durante las operaciones **ADD** y **ADC** para indicar que un acarreo ocurrió entre los **bits** 3 y 4.

Interrupción (I)

Cuando este bit es activado, el timer y las interrupciones externas son enmascaradas (deshabilitadas). Si una interrupción externa ocurre mientras este bit es activado, la interrupción es capturada y procesada tan pronto como el bit de interrupción es limpiado.

Negativo (N)

Se activa este bit indicando que el resultado de la última operación aritmética, lógica o datos fue negativa (el bit 7 en el resultado es un 1 lógico).

Cero (Z)

Cuando se coloca, este bit indica que el resultado del último dato aritmético, lógico o de manipulación fue cero.

Carry/Borrow (Acarreo/Préstamo)

Cuando se carga este bit indica que un acarreo o borrow salió de la unidad aritmética y lógica (ALU) y que ocurrió en la última operación aritmética. Sin embargo, este bit se ve afectado durante la prueba de bit e instrucciones de ramificación y durante desplazamientos y rotaciones.

Resets (Inicializadores)

El MCU puede ser inicializado en dos formas: por un encendido inicial y por una entrada externa de inicialización $\overline{\text{RESET}}$. La entrada de $\overline{\text{RESET}}$ consiste principalmente de un Schmitt trigger que sensa el nivel lógico de la línea.

Reset de encendido

La inicialización es generada al encenderse y permite que el reloj interno generado se estabilice. El reset de encendido es usado estrictamente para poder activar el voltaje. Un retardo de t_{RHL} milisegundos es requerido antes de dejar que la entrada de $\overline{\text{RESET}}$ vaya a alto. Conectando un capacitor a la entrada de $\overline{\text{RESET}}$ (figura 5) normalmente provee el retardo suficiente.

Entrada externa de Reset

El MCU es inicializado cuando un cero lógico es aplicado a la entrada del $\overline{\text{RESET}}$ por un período mas largo que un ciclo de máquina t_{CYC}. Bajo este tipo de reset, los interruptores Schmitt trigger producen un voltaje interno de reset de V_{IRES}.

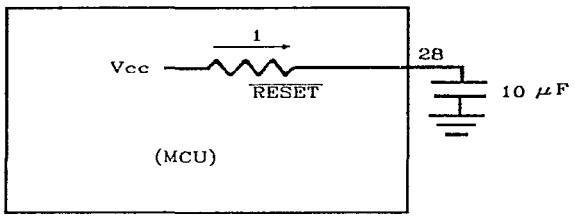


Figura 5. Circuito del Retardo de Encendido del RESET

Interrupciones

El MCU puede ser interrumpido de 4 diferentes formas:

(1) A través de la entrada de interrupción externa $\overline{\text{INT}}$.

(2) Con la petición de interrupción interna del timer.

(3) Usando la instrucción de interrupción por software **(SWI)** o

(4) El pin de entrada externo del Puerto D $\overline{\text{INT2}}$

Las interrupciones causan que los registros del procesador sean salvados en el stack y la interrupción enmascarada **(bit I)** se active para impedir interrupciones adicionales. La instrucción **RTI** hace que los contenidos de los registros sean

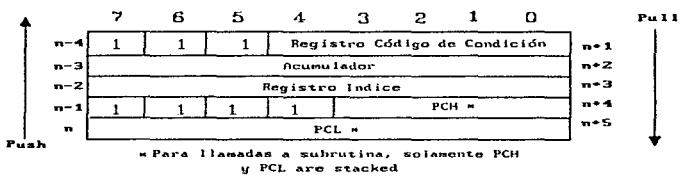


Figura 6. Interrupt Stacking Order

recuperados del stack para después continuar con el procesamiento normal. El orden del apilamiento es mostrado en la figura 6.

Cuando no se requiera el $\overline{\text{RESET}}$, las interrupciones de hardware harán que no se ejecuten instrucciones pero se considerarán como pendientes hasta que la

instrucción actual se complete.

NOTA

La instrucción que se ejecuta es considerada como la única operando.

Cuando la instrucción actual es completada, el procesador revisa todas las interrupciones pendientes por hardware y si no están enmascaradas (el **bit I** limpio) procede con el procesamiento de la interrupción, por otro lado, la siguiente instrucción es buscada y ejecutada. La interrupción enmascarada es capturada al final del servicio de interrupción. Si el bit de estado de la interrupción por timer es limpiado antes de la interrupción no enmascarada, entonces la interrupción no es capturada.

Si una interrupción externa y una interrupción por timer están pendientes al final de la ejecución de una instrucción, la interrupción externa es atendida primero. La instrucción **SWI** es ejecutada igual que otra instrucción sin reparar en la activación del bit I. Ver la figura 7 para la secuencia de procesamiento de las instrucciones de reset y de interrupción.

Interrupción por Timer

Si el bit enmascarado de tiempo (TCR6) es limpiado, cada vez que el tiempo del

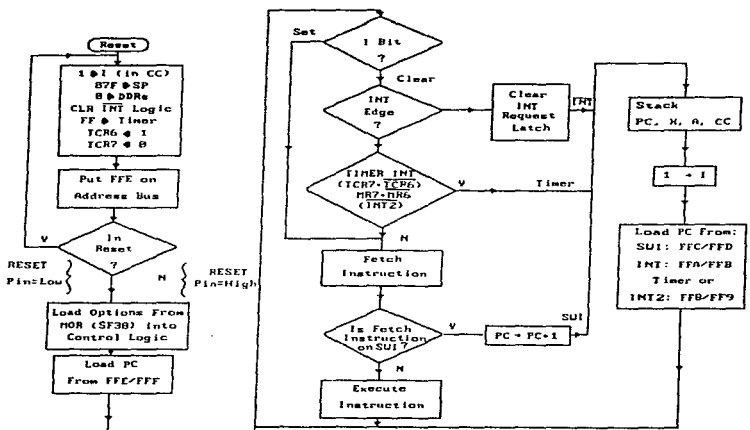


Figura 7. Diagrama de flujo del porcesamiento de Reset e Interrupciones

timer se decremente a cero (transición de $01 a $00) una petición de interrupción será generada. El procesamiento de la actual interrupción se genera solamente si el bit de la interrupción enmascarada del registro de código de condición (CCR) es también limpiado. Cuando la interrupción es reconocida, el estado actual de la maquina es guardado en el stack y el bit I en el CCR es activado, se enmascaran las interrupciones adicionales hasta que sean servidas. El vector de interrupción por timer, contiene la ubicación de la rutina de servicio de la interrupción, entonces es cargada en el contador del programa. Al final de la rutina de servicio de la interrupción por timer, el software ejecuta una instrucción RTI la cual restaura el estado de la máquina y continua la ejecución del programa interrumpido. El bit de estado de la interrupción por timer solamente puede ser limpiado por software.

Interrupciones Externas

Las interrupciones externas son internamente sincronizadas y capturadas sobre el flanco de bajada de $\overline{\text{INT}}$ e $\overline{\text{INT2}}$. Al limpiar el bit I se habilita la interrupción externa. La interrupción $\overline{\text{INT2}}$ tiene un bit de petición de interrupción (el bit 7) y un bit enmascarado (el bit 6) en el registro de misceláneas (MR). La interrupción $\overline{\text{INT2}}$

se inhibe cuando el bit enmascarado es activado. $\overline{INT2}$ siempre será leída como una entrada digital en el puerto D. Los bits de $\overline{INT2}$ y de petición de interrupción por timer, si son activados causan que el proceso del MCU sea interrumpido cuando el bit **I** del **CCR** es limpiado. Los siguientes párrafos describen dos circuitos típicos de interrupciones externas.

Interrupción de Cruce por Cero

Una señal de entrada senoidal (f_{INT} máxima) puede ser usada para generar una interrupción externa (ver la figura 8a) y usarla como un detector de cruce por cero (la transición negativa de la AC senoidal). Este tipo de circuitos permiten aplicaciones tales como rutinas de servicio de tiempo real y de atracción/repulsión de dispositivos de control de potencia de "ac". Una rectificación de onda completa fuera del chip provee una interrupción en cada cruce por cero de la señal de "ac" y con lo cual permite un reloj de 2f.

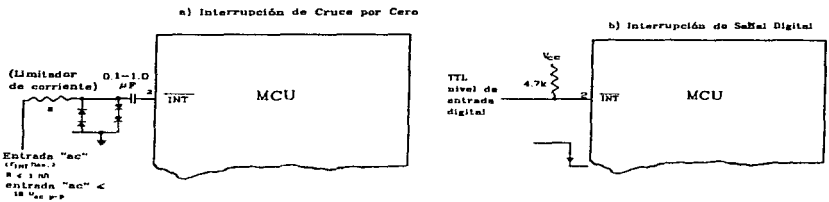


Figura 8. Circuito típico de interrupciones

Interrupción por medio de una Señal Digital

Con este tipo de circuito (Figura 8b) el pin $\overline{INT}$ puede ser impulsado por una señal digital. La máxima frecuencia de una señal que puede ser reconocida por el pin TIMER o $\overline{INT}$ depende de los parámetros t_{WIL} o t_{WIH}.

Interrupción por Software (SWI)

SWI es una instrucción que es procesada a pesar del estado del **bit I** en el **CCR**. La ejecución del **SWI** es similar a las interrupciones por hardware.

Modos de Operación

El MCU tiene dos modos de operación: normal y bootstrap.

Modo Normal

Este modo es del chip sencillo y se activa si se reúnen las siguientes condiciones:

(1) la línea $\overline{\text{RESET}}$ es baja.

(2) el pin PC0 está dentro de su rango normal de operación y

(3) el pin V_{PP} es conectado al pin V_{SS}.

El siguiente flanco de subida en el pin de $\overline{\text{RESET}}$ causa entonces que este entre al modo normal.

Modo Bootstrap

Al modo bootstrap se entra si se le conectan +12V al pin TIMER.

TIMER

El MCU contiene un contador de 8 bits programable, controlado por un prescalador de 7 bits programable por software. La variedad de fuentes de timer son hechas vía el registro de control del timer (TCR) y la opción de registro enmascarable (MOR). El contador de 8 bits puede ser cargado por medio del control del programa y se decrementa hasta ser cero. Cuando el timer alcance el cero, el bit de petición de interrupción del timer (bit 7) se coloca en el registro de control del timer. Referencia en la figura 9 en el diagrama de bloques del timer.

La interrupción por timer puede ser enmascarada (deshabilitada) al colocar el bit de la interrupción de timer enmascarado (bit 6) en el TCR. Cuando el bit I en el CCR y el bit 6 del TCR son limpiados, el procesador recibe la interrupción. El MCU responde a esta interrupción:

(1) salvando el estado presente del CPU en el stack,

(2) buscando el vector de interrupciones del timer y

(3) ejecutando la rutina de interrupción.

La petición de interrupción por timer deberá ser limpiada por software.

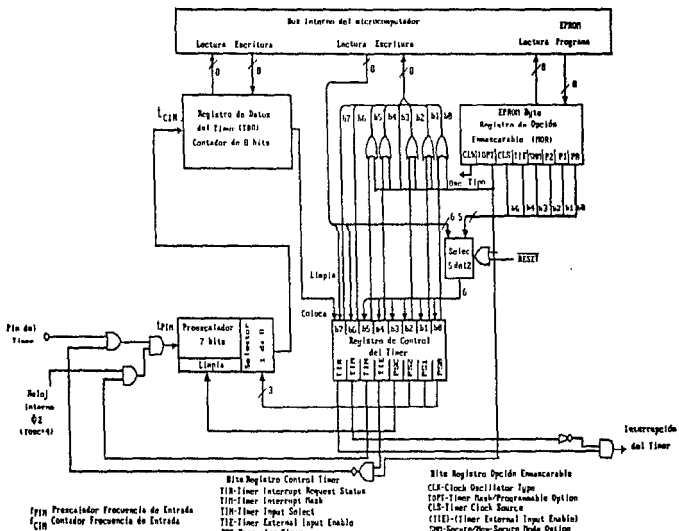


Figura 9. Diagrama de Bloques del Timer

El prescalador es un divisor de 7 bits el cual es usado para extender la longitud del timer al máximo. Para evitar errores de truncamiento, el prescalador es limpiado cuando al bit 3 del TCR se le pone un 1 lógico, comúnmente, el bit 3 del TCR siempre tiene un cero lógico para asegurar las operaciones apropiadas con instrucciones de lectura-modificación-escritura.

El timer al llegar a $00 pasa a $FF y continua su cuenta descendente. El contador puede leerse en cualquier momento, por medio del registro de datos del timer (TDR). Esto ayuda a determinar la longitud del tiempo desde que ocurrió la interrupción, sin distorsionar el tiempo de procesamiento. El TDR no es afectado.

Modo controlado por software

Este modo se selecciona cuando el **TOPT (bit 6)** en el MOR es programado con un cero. La entrada del prescalador del timer puede ser configurada en 3 diferentes modos de operación, se deshabilita el modo dependiendo del valor escrito en los **bits 4 y 5** de control del TCR **(TIE y TIN).**

Modo 1 de entrada al TIMER

Cuando **TIE** y **TIN** son ambos programados con ceros, la entrada del timer es de un reloj interno (φ_2) y la entrada del pin del timer se deshabilita. El modo de reloj interno puede ser usado para la generación de interrupciones periódicas y también como una referencia para medir frecuencias y eventos.

Modo 2 de entrada al TIMER

Cuando el **TIE=1** y **TIN=0** el reloj interno y la entrada de señales del timer son las entradas de una AND y la salida forma la entrada al timer. Este modo puede ser usado para medir pulsos externos. El flanco alto del pulso externo dispara el reloj interno por la duración del pulso externo. La precisión del contador es ± 1.

Modo 3 de entrada al TIMER

Cuando **TIE=0** y **TIN=1**, no hay entrada de frecuencia al prescalador y el timer es deshabilitado.

Modo 4 de entrada al TIMER

Cuando **TIE=1** y **TIN=1**, la entrada del timer es de un reloj externo. El reloj externo puede ser usado para contar los eventos y tan bien provee una frecuencia externa para generar interrupciones periódicas. La frecuencia de entrada deberá ser <= $f_{osc}/8$.

Modo de control MOR

Este modo se selecciona cuando **TOPT (el bit 6)** en el MOR es programado con un uno lógico. El circuito del timer es el mismo que el descrito en el Modo de Control por Software. Los niveles lógicos de los bits del **TCR 0, 1, 2 y 5** son determinados durante la programación de la EPROM por los mismos bits en la MOR. Por lo tanto, los **bits 0, 1, 2 y 5** del MOR controlan la división del prescalador y selecciona el reloj del timer, **TIE (bit 4)** y **PSC (bit 3)** del TCR son activados como unos lógicos cuando está en el modo de control MOR. **TIM (bit 6)** y **TIR (bit 7)** son controlados por el contador y el software.

Registro de control del TIMER (TCR) $009

Es un registro de 8 bits que controla varias funciones que configuran el modo de operación del prescalador, generando señales de interrupción por timer. Todos los bits son de lectura o escritura excepto el bit 3. La configuración del TCR es determinada por el **TOPT (el bit 6)** en el MOR. Cuando **TOPT=1** el TCR emula el MC6805U2, cuando **TOPT=0**, el TCR es controlado por software.

TCR con MOR TOPT=1

7	6	5	4	3	2	1	0
TIR	TIM	*	*	PSC	*	*	*

TCR con MOR TOPT=0

7	6	5	4	3	2	1	0
TIR	TIM	TIN	TIE	PSC	PS2	PS1	PS0
RESET							
0	1	U	U	U	U	U	U

* El valor correspondiente de los bits en el MOR son escritos durante la transición de subida del $\overline{\text{RESET}}$. Esos bits siempre al leerse son "unos".

U Indeterminado

TIR Petición de Interrupción por timer

Usado para indicar una interrupción por timer cuando es un uno lógico.

1 = Colocado cuando el registro de datos del timer cambia a cero.

0 = Limpiado por el reset externo, por el reset de encendido o bajo el control del programa.

TIM Interrupción por timer enmascarada.

Usado para inhibir la interrupción por timer.

1 = Interrupción inhibida

0 = Interrupción habilitada

TIN Externo o Interno

Selecciona la entrada de la fuente de reloj

1 = Reloj externo seleccionado

0 = Reloj interno seleccionado ($f_{osc}/4$)

PSC Limpia el Prescalador

Escribe solamente el bit. Escribe un uno en ese bit para iniciar el prescalador en

ceros. La lectura siempre indica cero cuando **TOPT=0**. Cuando **TOPT=1** ese bit leería un uno lógico y no tiene efecto en el prescalador.

PS2, PS1, PS0 Limpiar el Prescalador

Decodificado para seleccionar uno de ocho salidas del prescalador.

PS2	PS1	PS0	División por
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

NOTA

Cuando se cambia los bits PS en el software, el bit PSC se escribirá un "uno" en

el mismo ciclo de escritura para limpiar el prescalador. Cambiar los bits PS sin limpiar el prescalador podría causar un truncamiento en el prescalador.

Registro de opción enmascarado (MOR) S0F38

El MOR es implementado en la EPROM. Este registro contiene principalmente todos ceros al programar y no es afectado por el reset. Los bits del MOR son descritos en los siguientes párrafos.

7	6	5	4	3	2	1	0
CLK	TOPT	CLS			P2	P1	P0

CLK Reloj (tipo de oscilador)

1 = Resistencia-Capacitor (RC)

0 = Cristal

TOPT Opción del Timer

1= En el tipo de timer/prescalador del MC6805U2. Todos los bits excepto

el **6 y 7** del TCR son invisibles al usuario. Los **bits 5, 2, 1 y 0** del MOR determinan la opción equivalente del MC6805U2 enmascarado.

0= Todos los bits del TCR son controlados por software. El estado de los **bits del MOR 5, 4, 2, 1 y 0** son activados con los valores iniciales de los bits respectivos del TCR.

CLS Fuente del reloj Timer/Prescalador

1= Pin Timer externo.

0= Reloj interno.

Bit 4

No usado si **TOPT=1**. Fija el valor inicial del TIE en el TCR si **TOPT=0.**

1= No usado.

0= Fija los valores iniciales del TIE en el TCR.

Bit 3

No usado.

PS2, PS1, PS0

Los niveles lógicos de estos bits, cuando se decodifican, seleccionan una de las ocho salidas en el prescalador del timer.

Registro de control de programación (PCR) $00B

El PCR es un registro de 8 bits el cual provee los bits necesarios de control para programar la EPROM. El programa de bootstrap manipula el PCR cuando el usuario necesita programar en este modo, no interesa el PCR en otras aplicaciones.

7	6	5	4	3	2	1	0
1	1	1	1	1	$\overline{VPON}$	$\overline{PGE}$	$\overline{PLE}$
RESET							
U	U	U	U	U	U	1	1

$\overline{PLE}$ Habilita la captura del programa

Controla la captura de datos y direcciones en la EPROM. Colocados durante el reset, al ser limpiada.

1 = Lectura de la EPROM

0 = Captura de direcciones y datos en la EPROM

$\overline{\text{PGE}}$ Habilita la programación

Habilita la programación de la EPROM. Deberá colocarse cuando cambian las direcciones y datos, se activa en la inicialización.

1 = Inhibe la programación de la EPROM.

0 = Habilita la programación de la EPROM (si $\overline{\text{PLE}}$ es baja)

$\overline{\text{VPON}}$ V_{PP} encendido

Un bit de solo de lectura que indica un voltaje alto en el pin V_{PP}. Cuando lo coloca en "uno", desconecta $\overline{\text{PGE}}$ y $\overline{\text{PLE}}$ del chip.

1 = No hay un voltaje alto en el pin V_{PP}.

0 = Voltaje alto en el pin V_{PP}.

NOTA

$\overline{\text{VPON}}$ siendo "cero" no indica que el nivel del V_{PP} sea el correcto para la programación. Es usado como un seguro entrelazado por el usuario en el modo normal

de operación.

$\overline{VPON}$	$\overline{PGE}$	$\overline{PLE}$	**Condiciones de programación.**
0	0	0	Modo de programación (programa un byte de la EPROM)
1	0	0	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados del sistema
0	1	0	Programación deshabilitada (captura de datos y direcciones en la EPROM)
1	1	0	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados del sistema
0	0	1	Estado inválido, $\overline{PGE} = 0$ si $\overline{PLE} = 0$
1	0	1	Estado inválido, $\overline{PGE} = 0$ si $\overline{PLE} = 0$
0	1	1	"Alto voltaje en V_{PP}"
1	1	1	$\overline{PGE}$ y $\overline{PLE}$ deshabilitados del sistema (modos de operación)

Conjunto de instrucciones

El MCU tiene un conjunto de 58 instrucciones básicas la cual puede ser dividida en 5 diferentes tipos: registro/memoria, lectura/modificación-escritura, bifurcación, manipulación de bit y control.

Instrucciones de Registro/Memoria

La mayoría de estas instrucciones usan dos operandos. Un operando es cualquiera de los acumuladores o registros indexados. Los otros operandos se obtienen de la memoria usando uno de los modos de direccionamiento. El salto incondicional (JMP) y el salto a instrucciones de subrutina (JSR) no tienen operandos de registros. Referencia en las siguientes listas de instrucciones.

Función	Mnemónico
Carga A desde la memoria	LDA
Carga X desde la memoria	LDX
Almacena A en la memoria	STA
Almacena X en la memoria	STX
Suma la memoria al contenido de A	ADD
Suma la memoria y el acarreo al contenido de A	ADC
Resta a la memoria	SUB
Resta a la memoria desde A con préstamo	SBC
Realiza una AND de la memoria con A	AND
Realiza una OR de la memoria con A	ORA
Realiza una OR Exclusiva de la memoria con A	EOR
Comparación aritmética de A con la memoria	CMP
Comparación aritmética de X con la memoria	CPX

Función	Mnemónico
Bit de verificación de la memoria con A (comparación lógica)	BIT
Salto incondicional	JMP
Salto a subrutina	JSR

Instrucciones de Lectura-Modificación-Escritura

Las instrucciones de lectura de una localidad de memoria o de un registro modifican o revisan su contenido y escriben el valor modificado respaldado en memoria o en el registro. Las instrucciones para verificar valores negativos o cero (TST) es una excepción a la secuencia de lectura-modificación-escritura no modifica el valor.

Función	Mnemónico
Incremento	INC
Decremento	DEC
Limpia	CLR
Complemento	COM
Negado (Segundo complemento)	NEG
Rotación a la izquierda con acarreo	ROL

Función	Mnemónico
Rotación a la derecha con acarreo	ROR
Corrimiento lógico a la izquierda	LSL
Corrimiento lógico a la derecha	LSR
Corrimiento aritmético a la derecha	ASR
Verifica para negativo o cero	TST

Instrucciones de salto

Este conjunto de instrucciones da salto si se encuentra con una condición particular, en otro caso, la operación no es realizada.

Función	Mnemónico
Bifurca siempre	BRA
Nunca bifurca	BRN
Bifurca si es alto	BHI
Bifurca si es bajo o igual	BLS
Bifurca si el acarreo está limpio	BCC
(Bifurca si es alto o igual)	(BHS)
Bifurca si se coloca el acarreo	BCS
(Bifurca si es bajo)	(BLO)
Bifurca si no es igual	BNE

Función	Mnemónico
Bifurca si es igual	BEQ
Bifurca si el acarreo medio está limpio	BHCC
Bifurca si el acarreo medio se coloca	BHCS
Bifurca si suma	BPL
Bifurca si disminuye	BMI
Bifurca si el bit de interrupción enmascarable se limpia	BMC
Bifurca si el bit de interrupción enmascarable se coloca	BMS
Bifurca si la línea de interrupción es baja	BIL
Bifurca si la línea de interrupción es alta	BIH
Bifurca a subrutina	BSR

Instrucciones para la manipulación de bits

El MCU es capaz de fijar o limpiar algunos bits que residen en los primeros 256 bytes del espacio de memoria donde todos los puertos, los DDRs de los puertos, el timer, el control del timer y dentro de la RAM residen. Características adicionales permiten al software verificar y bifurcar el estado de algunos bits más allá de esas 256 localidades. Las funciones de limpiar bit y verificar bit y de bifurcación se implementan con una simple instrucción. Para instrucciones de verificación y

bifurcación, el valor del bit verificado también se coloca en el bit de acarreo del registro de código de condición.

Función	Mnemónico
Bifurca si el bit n se coloca	BRSET n (n = 0...7)
Bifurca si el bit n se limpia	BRCLR n (n = 0...7)
Coloca el bit n	BSET n (n = 0...7)
Limpia el bit n	BCLR n (n = 0...7)

Instrucciones de control

Estas instrucciones son registros de referencia de instrucciones, son usadas para controlar la operación del procesador durante la ejecución de un programa.

Función	Mnemónico
Transfiere A a X	TAX
Transfiere X a A	TXA
Coloca el bit de acarreo	SEC
Limpia el bit de acarreo	CLC
Coloca el bit de interrupción enmascarada	SEI
Limpia el bit de interrupción enmascarada	CLI

Función	Mnemónico
Software Interrupt	SWI
Retorno de subrutina	RTS
Retorno de interrupción	RTI
Reinicializa el Stack Pointer	RSP
No Opera	NOP

Modos de Direccionamiento

El MCU usa 10 diferentes modos de direccionamiento para proveer al programador la oportunidad de optimizar el código para todas las situaciones. La variedad de modos de direccionamiento indexado hacen posible localizar tablas de datos, códigos de conversión de tablas y escalamiento de tablas a alguna parte en el espacio de memoria.

Accesos cortos son instrucciones indexadas de bytes simples mientras que grandes instrucciones (3 bytes) permiten accesar tablas por toda la memoria. Pequeños y grandes direccionamientos absolutos son también incluidos. Para instrucciones de direccionamiento directo con dos bytes se accesa a todos los bytes de datos en la

mayoría de las aplicaciones. El direccionamiento extendido permite a las instrucciones de salto alcanzar toda la memoria.

El término de "direccionamiento efectivo" (EA) es usado en la descripción de varios modos de direccionamiento. El direccionamiento efectivo es definido como la dirección de la cual el argumento para una instrucción se basa o se almacena.

Inmediato

En el modo de direccionamiento inmediato, el operando está contenido en el byte seguido inmediatamente del código. El modo de direccionamiento inmediato es usado para accesar constantes que no son cambiadas durante la ejecución del programa (p.e una constante para inicializar un ciclo en el contador).

Directo

En el modo de direccionamiento directo, la dirección efectiva del argumento está contenida en un byte simple seguida del byte opcode. El direccionamiento directo permite al usuario direccionar directamente los 256 bytes en memoria con una simple instrucción de dos bytes.

Extendido

En el modo de direccionamiento extendido, el direccionamiento efectivo del argumento es contenido en los dos bytes siguientes del byte de código. Las instrucciones con direccionamiento extendido son capaces de referenciar argumentos en cualquier lugar de la memoria con una simple instrucción de tres bytes.

Relativo

El modo de direccionamiento relativo se usa solamente en instrucciones de bifurcación. En el direccionamiento relativo, el contenido de los 8 bits señalados por el byte (el offset) siguiente al código es sumado al PC si y solamente si la condición de bifurcación es verdadera. Por otro lado el control procede a la siguiente instrucción. El espacio del direccionamiento es de -126 a +126 de la dirección del código.

Indexado, sin offset

En el modo de direccionamiento indexado sin offset, la dirección efectiva del argumento está contenida en los 8 bits del registro indexado. De esta forma, el modo de direccionamiento puede ser accesado en las primeras 256 localidades de memoria.

Esas instrucciones están solamente en un byte largo. Este modo es con frecuencia usado para mover un apuntador a través de una tabla o retener la dirección de una localidad de RAM o E/S frecuentemente referenciada.

Indexado con un offset de 8 bits

En el modo de direccionamiento indexado con offset de 8 bits, la dirección efectiva es la suma del contenido del registro indexado de 8 bits sin signo y el byte sin signo siguiente al código. Este modo de direccionamiento es usado para seleccionar el k_ésimo elemento en una tabla de n elementos. Con sus dos bytes la instrucción k podría típicamente estar en x con la dirección de inicio de la tabla en la instrucción. Cuando tales tablas son localidades dentro de cualesquiera de las primeras 256 direcciones y podrían extenderse tan lejos como 510 localidades, (**S1FE** es la última localidad en la cual la instrucción puede comenzar).

Indexado con un offset de 16 bits

En el modo de direccionamiento indexado con offset de 16 bits, la dirección efectiva es la suma del contenido del registro indexado de 8 bytes sin signo y dos bytes

sin signo siguientes al código. Este modo de direccionamiento puede ser usado en una forma similar al indexado con 8 bits de offset excepto que en esta instrucción de 3 bytes la tabla siempre estará en cualquier parte en la memoria.

Bit fijar/limpiar

En el modo de direccionamiento de bit fijado/limpiado, el bit activará o limpiará esa parte del código. El byte siguiente del código especifica el direccionamiento directo del byte en el cual el bit especificado deberá ser fijado o limpiado. De este forma, alguna lectura o escritura de un bit en las primeras 256 localidades de memoria, incluyendo E/S pueden ser selectivamente fijados o limpiados con una simple instrucción de dos bytes.

BIT prueba y bifurcación

El modo de direccionamiento del bit prueba y bifurcación junto con el direccionamiento directo y el direccionamiento relativo. El bit será examinado y su condición (activado o limpiado) es incluida en el código. la dirección del byte para ser examinado está en un byte simple inmediatamente seguido del byte del código.

El offset relativo de 8 bits con signo del tercer byte es sumado al PC si el bit especificado a ser limpiado o colocado en la localidad especificada de memoria. Esta simple instrucción de 3 bytes siempre bifurca el programa y se basa en la condición de algún bit legible en las primeras 256 localidades de memoria. El espacio para bifurcación es de -125 a +130 de la dirección del código. El estado de la examinación del bit es también transferido al acarreo del bit del código de condición.

Inherente

En el modo de direccionamiento inherente toda la información necesaria para ejecutar la instrucción esté contenida en el código. Las operaciones especificadas solamente al registro índice o el acumulador son buenas como instrucciones de control con otros argumentos pero no son incluidos en este modo. Estas instrucciones son de un byte largo.

Especificaciones Eléctricas

Rangos máximos

Rangos	Símbolo	Valor	Unidad
Voltaje de alimentación	V_{CC}	-0.3 a +7.0	V
Voltajes de entrada EPROM Voltaje de programación (Pin VPP) Pin de TIMER (modo normal) Pin TIMER (modo de programación bootstrap) Todos los demás	 V_{PP} V_{in} V_{in} V_{in}	 -0.3 a +22.0 -0.3 a +7.0 -0.3 a +15.0 -0.3 a +7.0	 V V V V
Rango de operación de la temperatura	T_A	T_L a T_H 0 a +70	°C
Rango de temperatura de almacenamiento	T_{stg}	-55 a +150	°C
Temperatura de la unión del chip	T_J	150	°C/W

Características Térmicas

Características	Símbolo	Valor	Unidad
Resistencia Térmica del chip	θ_{JA}	60	°C/W

Consideraciones de Potencia

El promedio de la temperatura de la unión del chip , T_J, en °C puede obtenerse de la siguiente manera:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad \ldots\ldots (1)$$

donde:

T_A = Temperatura ambiente, °C.

θ_{JA} = Resistencia térmica del encapsulado, Unión a temperatura ambiente, °C/W.

$P_D = P_{INT} + P_{I/O}$

$P_{INT} = I_{CC} \times V_{CC}$, Watts - Potencia interna del chip.

$P_{I/O}$ = Disipación de la potencia en los pines de entrada y salida - Determinado por el usuario.

Para más aplicaciones $P_{I/O} < P_{INT}$ y puede ser ignorado. La siguiente es una relación es una aproximación entre P_D y T_J (Si se ignora $P_{I/O}$):

$$P_D = K / (T_J + 273\ °C) \quad \ldots\ldots (2)$$

Resolviendo las ecuaciones (1) y (2) para K, se obtiene:

$$K = P_D \cdot (T_A + 273° C) + \theta_{JA} \cdot P^2_D \quad \ldots\ldots (3)$$

donde K es una constante perteneciente a la parte particular. K puede determinarse

mediante la ecuación (3) midiendo P_D (en equilibrio) para conocer T_A. Utilizando este valor de K, los valores de P_D y T_J pueden obtenerse resolviendo iterativamente las ecuaciones (1) y (2) para cualquier valor de T_A.

Características Eléctricas

(Vcc=5.25 +-0.5 VDC, Vss=0 VDC, Vss=0 VDC, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de entrada alto $\overline{\text{RESET}}$(4.75 <= V_{CC} <=5.75) $\overline{\text{INT}}$ (4.75 <= V_{CC} <=5.75) (V_{CC} <4.75)) Todos los demás	V_{IH}	 4.0 Vcc-0.5 4.0 Vcc-0.5 2.0	 - - * * -	 Vcc Vcc Vcc Vcc Vcc	V
Voltaje de entrada alto (Pin TIMER) Modo Timer Modo de programación bootstrap	V_{IH}	 2.0 9.0	 - 12.0	 Vcc 15.0	V
Voltaje de entrada bajo $\overline{\text{RESET}}$ $\overline{\text{INT}}$ Todos los demás	V_{IH}	 -0.3 -0.3 -0.3	 -- ** --	 0.8 1.5 0.8	V

Características	Símbolo	Min	Tipo	Max	Unidad
Disipación interna de energía (No carga al puerto, V_{CC}= 5.25 V, T_A= °C)	P_{INT}	---	450	TBD	mW
Capacitancia de entrada XTAL Todas las demás	C_{in}	 --- ---	 25 10	 -- --	pF
$\overline{INT}$ voltaje de cruce por cero, a través de un capacitor	V_{INT}	2.0	--	4.0	Vacp-p
$\overline{RESET}$ voltaje de histéresis Fuera del voltaje de reinicialización Dentro del voltaje de reinicialización	 V_{IRES}+ V_{IRES}-	 2.1 0.8	 -- --	 4.0 2.0	V
Voltaje de programacion (Pin V_{PP}) Programación de la EPROM Modo de operación	V_{PP}*	 20.0 4.0	 21.0 V_{CC}	 22.0 5.75	V
Entrada actual TIMER (V_{in} = 0.4V) $\overline{INT}$ (V_{in} = 0.4V) EXTAL (V_{in}=2.4 V a V_{CC} opción del cristal) (V_{in}=0.4 V opción del cristal) $\overline{RESET}$ (V_{in}=0.8 V) (Capacitor externo de cambio de corriente)	I_{in}	 --- --- --- --- -4.0	 -- 20 -- -- --	 20 50 10. -1600 -40	µA

*V_{PP} es el pin 6 en el MC68705P3 y está conectado a V_{CC} en el modo de operación

Características	Símbolo	Min	Tipo	Max	Unidad
Disipación interna de energía (No carga al puerto, V_{CC}= 5.25 V, T_A= °C)	P_{INT}	---	450	TBD	mW
Capacitancia de entrada XTAL Todas las demás	C_{in}	 --- ---	 25 10	 -- --	pF
$\overline{INT}$ voltaje de cruce por cero, a través de un capacitor	V_{INT}	2.0	--	4.0	Vacp-p
$\overline{RESET}$ voltaje de histéresis Fuera del voltaje de reinicialización Dentro del voltaje de reinicialización	 V_{IRES}+ V_{IRES}-	 2.1 0.8	 -- --	 4.0 2.0	V
Voltaje de programación (Pin V_{PP}) Programación de la EPROM Modo de operación	V_{PP}*	 20.0 4.0	 21.0 V_{CC}	 22.0 5.75	V
Entrada actual TIMER (V_{in} = 0.4V) $\overline{INT}$ (V_{in} = 0.4V) EXTAL (V_{in}=2.4 V a V_{CC} opción del cristal) (V_{in}=0.4 V opción del cristal) $\overline{RESET}$ (V_{in}=0.8 V) (Capacitor externo de cambio de corriente)	I_{in}	 --- --- --- --- -4.0	 -- 20 -- -- --	 20 50 10. -1600 -40	µA

*V_{PP} es el pin 6 en el MC68705P3 y está conectado a V_{CC} en el modo de operación

normal. En el MC6805P2, el pin 6 es NUM y está conectado a V_{SS} en el modo de operación normal.

**Debido a la oblicuidad interna, esta entrada (cuando no es utilizada) varía aproximadamente entre 2.0 V.

Puerto DC características eléctricas

(V_{CC}=+5.25 +-0.5 Vdc, V_{SS}=0 Vdc, T_A= 0°C a 70°C, menor en cualquier otro caso)

Características	Símbolo	Min	Tipo	Max	Unidad
Voltaje de Salida bajo, I_{LOAD}=1.6mA	V_{OL}	---	---	0.4	V
Voltaje de salida alto I_{LOAD}= -100µA	V_{OH}	2.4	---	---	V
Voltaje de salida alto I_{LOAD}= -10µA	V_{OH}	V_{CC}-10	---	---	V
Voltaje de entrada alto I_{LOAD}= -300µA (Max)	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo I_{LOAD}= -500µA (Max)	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada (V_{in} = 20 V a V_{cc})	I_{IH}	---	---	-300	µA
Hi-Z estado actual de la entrada (V_{in} = 0.4 V)	I_{IL}	---	---	-500	µA

Características	Símbolo	Min	Tipo	Max	Unidad
Puerto B					
Voltaje de Salida bajo, $I_{Load} = 3.2\,mA$	V_{OL}	---	---	0.4	V
Voltaje de Salida bajo, $I_{Load} = 10\,mA$ (Sink)	V_{OL}	---	---	1.0	V
Voltaje de salida alto $I_{Load} = -200\,\mu A$	V_{OH}	2.4	---	---	V
Darlington Current Drive (Source) $V_O = 1.5\,V$	I_{OH}	-1.0	---	-10	mA
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V
Hi-Z estado actual de la entrada	I_{TSI}	---	2	20	μA
Puerto C					
Voltaje de Salida bajo, $I_{Load} = -1.6\,mA$	V_{OL}	---	---	0.4	V
Voltaje de Salida alto, $I_{Load} = -100\,\mu A$	V_{OH}	2.4	---	---	V
Voltaje de entrada alto	V_{IH}	2.0	---	V_{CC}+0.7	V
Voltaje de entrada bajo	V_{IL}	V_{SS}	---	0.8	V

Características	Símbolo	Min	Tipo	Max	Unidad
Hi-Z estado actual de la entrada	I_{TSI}	---	---	20	μA

Características de encendido

(V_{CC}= +5.25 +/- 0.5 Vdc, V_{SS}=0 Vdc, T_A= 0° C a 70° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Frecuencia del Oscilador Normal	f_{OSC}	0.4	---	4.2	MHz
Tiempo del Ciclo de intrucción	t_{cyc}	0.950	---	10	μs
Ancho del pulso de INT o Timer	t_{WL}, t_{WH}	t_{cyc}+250	---	---	ns
Ancho del pulso de RESET	t_{RWL}	t_{cyc}+250	---	---	ns
Tiempo de retardo del RESET (Capacitancia externa = 1.0 μF)	t_{RHL}	100	---	---	ms
Detección de la frecuencia de entrada de cruce por cero INT	f_{INT}	0.03	---	1.0	kHZ

Características	Símbolo	Min.	Tipo	Max.	Unidad
Ciclo duty del reloj externo (EXTAL)	---	40	50	60	%

Características de la operación de programación eléctrica

(V_{CC}= +5.25 +/-0.5 Vdc, V_{SS}=0 Vdc, T_A= 20° C a 30° C, menor en cualquier otro caso)

Características	Símbolo	Min.	Tipo	Max.	Unidad
Voltaje de programación (Pin V_{PP})	V_{PP}	20.0	21.0	22.0	V
Current Supply V_{PP} V_{PP} = 5.25 V V_{PP} = 21.0 V	I_{PP}	 --- ---	 --- ---	 8 30	mA
Frecuencia de oscilación de programación	f_{OSCP}	0.9	1.0	1.1	MHz
Voltaje del modo de programación bootstrap (Pin del TIMER) I_{in} = 100	V_{HTP}	9.0	12.0	15.0	V

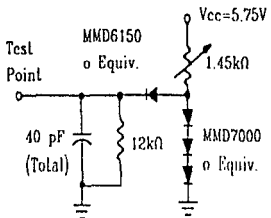


Figura 10. TTL Equivalent Test Load (Port B)

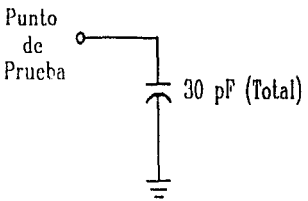


Figura 11. CMOS Equivalent Test Load (Port A)

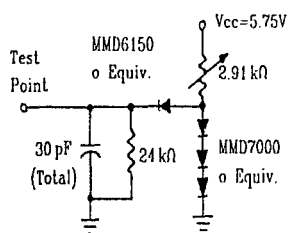


Figura 12. TTL Equivalent Test Load (Ports A and C)

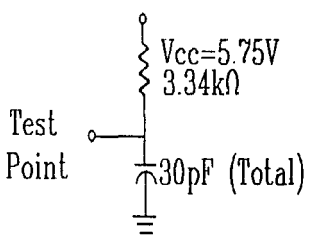


Figura 13. Open-Drain Equivalent Test Load (Port C)

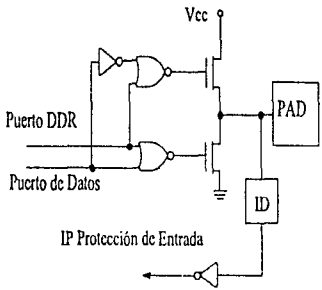


Figura 15. Diagrama Lógico de los Puertos A y C

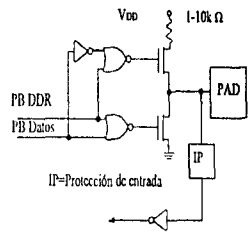


Figura 16. Diagrama Lógico del Puerto B

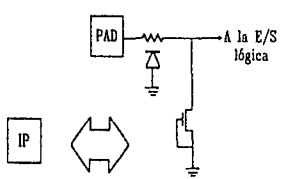


Figura 17. Típica Protección de Entrada

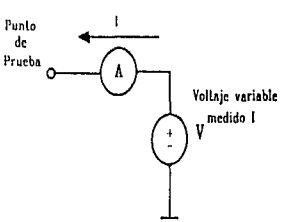


Figura 18. I/O Characteristic Meassurement Circuit

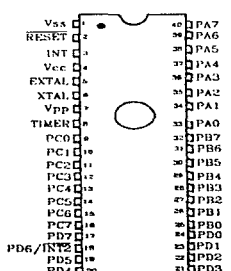
Información de Genérica

La siguiente tabla provee información genérica perteneciente al tipo de empaquetado, temperatura y números de orden del los MC para el MC68705U3.

Tabla 3. Información Genérica

Tipo de empaquetado	Temperatura	Número de Orden
Encapsulado (Sufijo S)	0° a 70°C	MC68705U3S
Encapsulado (Sufijo S)	-40° a +85°C	MC68705U3CS

Datos mecánicos



Asignación de Pines

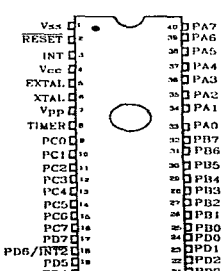
Información de Genérica

La siguiente tabla provee información genérica perteneciente al tipo de empaquetado, temperatura y números de orden del los MC para el MC68705U3.

Tabla 3. Información Genérica

Tipo de empaquetado	Temperatura	Número de Orden
Encapsulado (Sufijo S)	0° a 70°C	MC68705U3S
Encapsulado (Sufijo S)	-40° a +85°C	MC68705U3CS

Datos mecánicos



Asignación de Pines

Apéndice B

Manual de Usuario

Ensamblador de los MCU 68705P3, R3 y U3

Versión 1.0

Manual del usuario

INSTALACIÓN DEL SISTEMA:

Deberá realizar una copia de todos los archivos contenidos en el diskette de instalación, no olvide que este sistema sólo corre en ambiente Windows (ver requerimiento de software y hardware).

INICIO:

Para ejecutar el programa basta con pulsar doble click del botón izquierdo del mouse sobre el archivo **ens6805.exe**. Al ejecutar el programa se desplegará una ventana como la mostrada en la figura 1.

Dicha ventana en su menú principal contiene las siguientes opciones:

Archivo:

Esta opción permite manipular los archivos mediante las siguientes opciones: (ver figuras 1a y 2)

Abrir:

Aparecerá un cuadro de diálogo en el cual se deberá teclear el nombre del archivo y el directorio en el que se encuentra el archivo que se desea abrir.

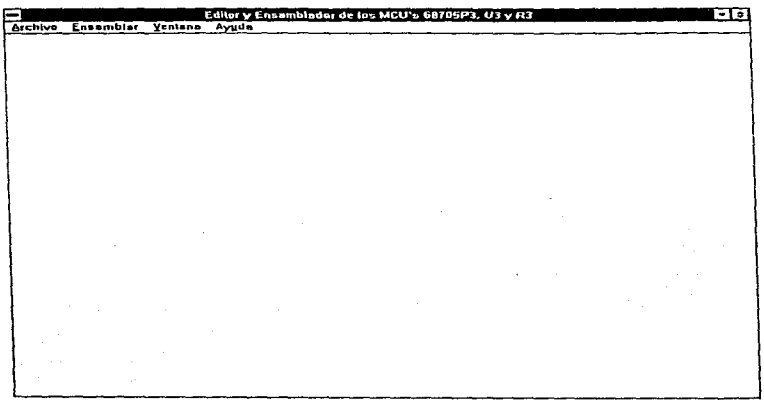


Figura 1 Menú principal

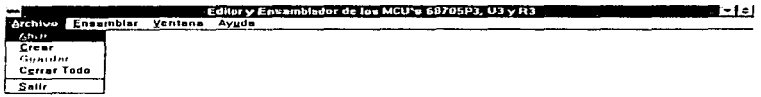


Figura 1a Opciones de archivo

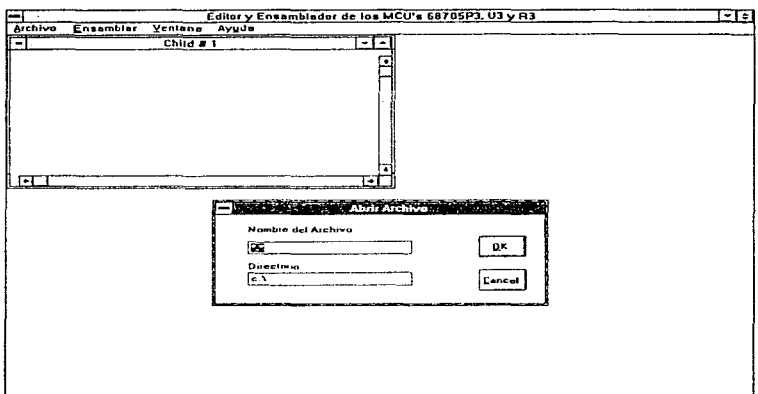


Figura 2 Abrir archivo

En caso de que no encuentre el archivo se desplegará un mensaje indicando el problema.

Crear:

Crea un archivo nuevo como se muestra en la figura 3.

Guardar:

Aparecerá un cuadro de diálogo en el cual se deberá teclear el nombre del archivo a guardar y el directorio en el que se desea almacenar el archivo (figura 4).

En caso de que el directorio especificado no se encuentre se desplegará un mensaje indicando el problema.

Cerrar todo:

Cierra todas las ventanas abiertas. Es importante que salve sus archivos antes de cerrar las ventanas de lo contrario, la información se perderá.

Salir:

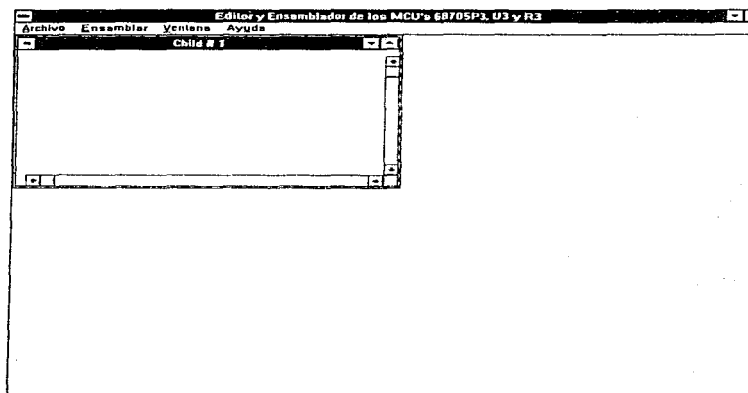


Figura 3 Crear archivo

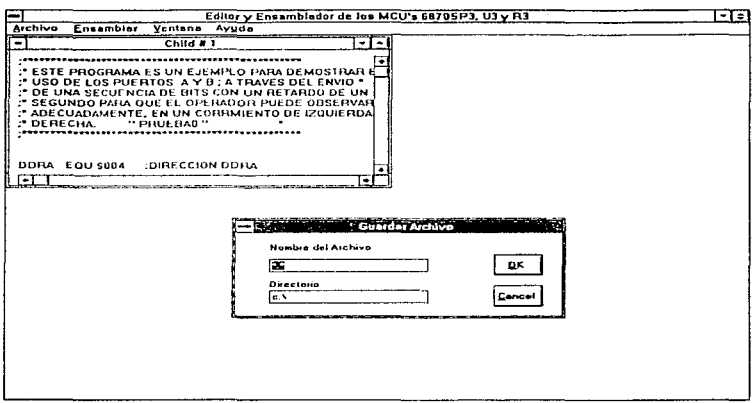


Figura 4 Guardar archivo

Sale de la aplicación ; al igual que sucede con la opción Close all deberá salvar sus archivos antes de finalizar la aplicación (figura 5).

Ensamblar:

Ensambla el archivo visualizado en la ventana activa (figura 6).

Este ensamblador es un ensamblador de dos pasadas que le reportará si existe algún error de sintaxis en su programa fuente, si estos existieran usted deberá corregirlos primero, estos son reportados con el número de línea que es utilizada por su editor; por lo tanto deberá reeditarlos en este.

Una vez que el programa este libre de errores, el ensamblador generará dos archivos, con el mismo nombre que el original, pero con diferentes extensiones, uno con la extensión "**.prn**" y otro con la extensión "**.mik**. El archivo ya ensamblado con la extensión "**prn**" puede mandarse a impresión con el editor, el archivo "**mik**" es el archivo objeto. El archivo "**prn**" contendrá su programa fuente ya ensamblado con sus direcciones correspondientes y los códigos de operación que serán reconocidos por su microcontrolador.

Nota: Las siguientes pseudo instrucciones tienen las equivalencias que se proporcionan:
RMB = **DB**
FCB = **DW**

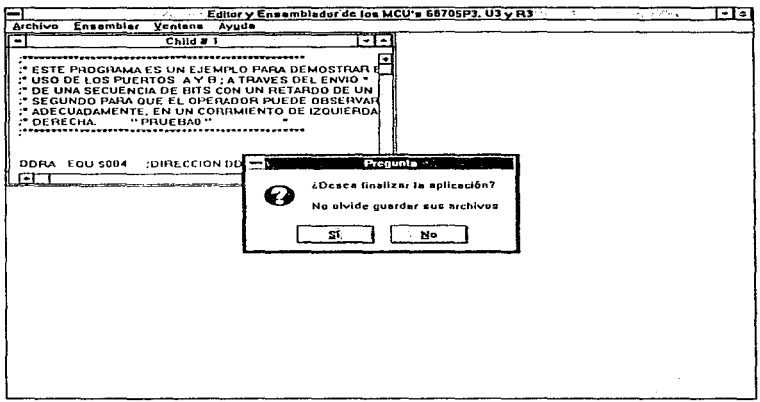


Figura 5 Salir

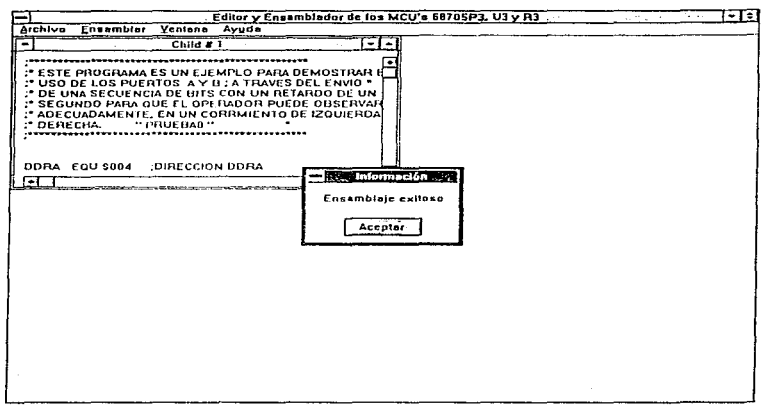


Figura 6 Ensamblar

Deberán utilizarse las que están con negrillas

Ventana:

Permite visualizar las ventanas de diferentes formas (figura 7):

Cascada:

Organiza las ventanas MDI en cascada (figura 8).

Mosaico:

Organiza las ventanas MDI en forma de mosaico (figura 9).

Organiza Íconos:

Organiza iconos (figura 10).

Cuenta ventanas:

Cuenta el número de ventanas abiertas (figura 11).

Acerca de...:

Muestra la información sobre la creación del ensamblador (figura 12).

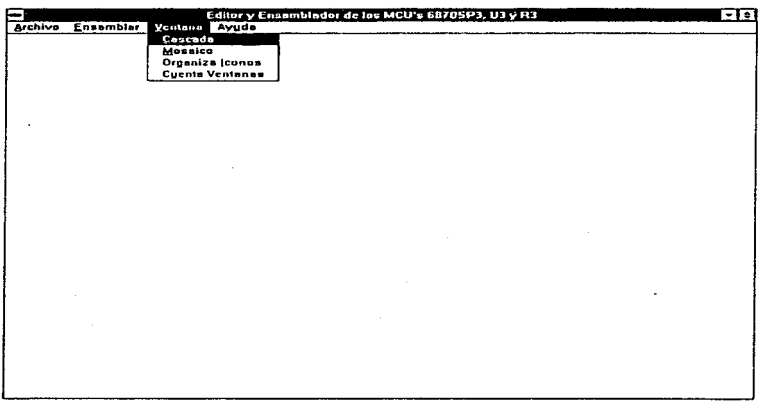


Figura 7 Opciones de ventana

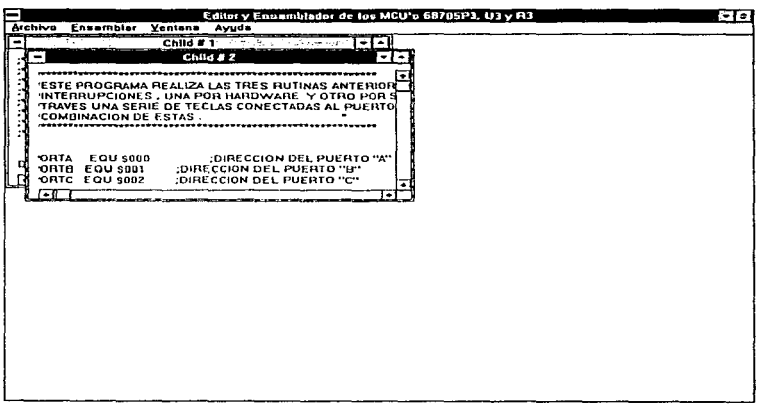


Figura 8 Cascada

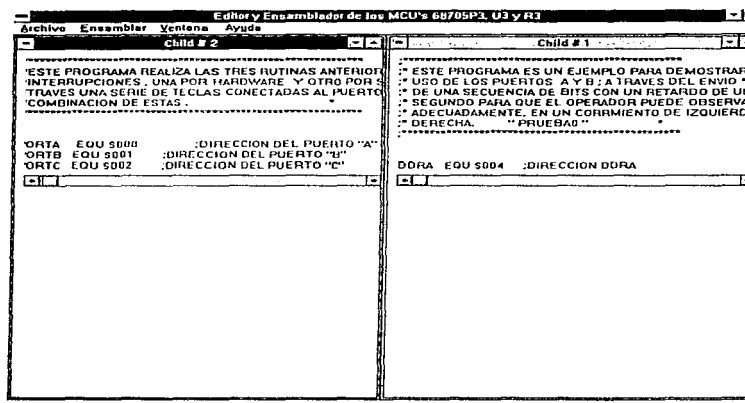


Figura 9 Mosaico

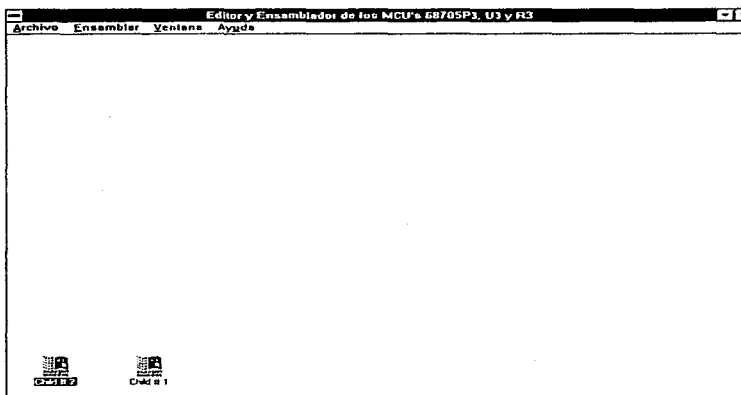


Figura 10 Organizar íconos

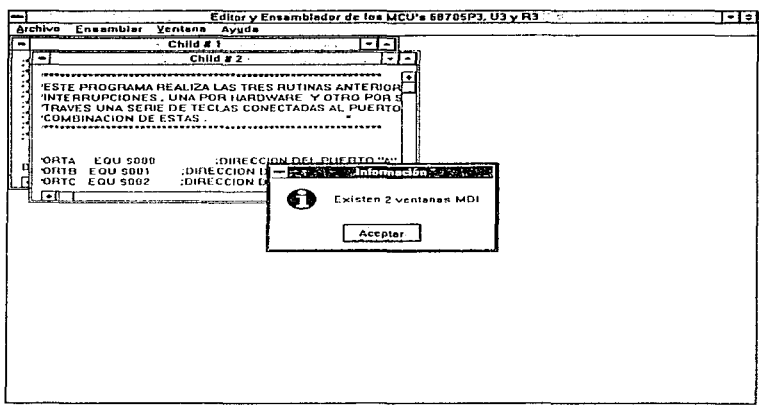


Figura 11 Contar ventanas

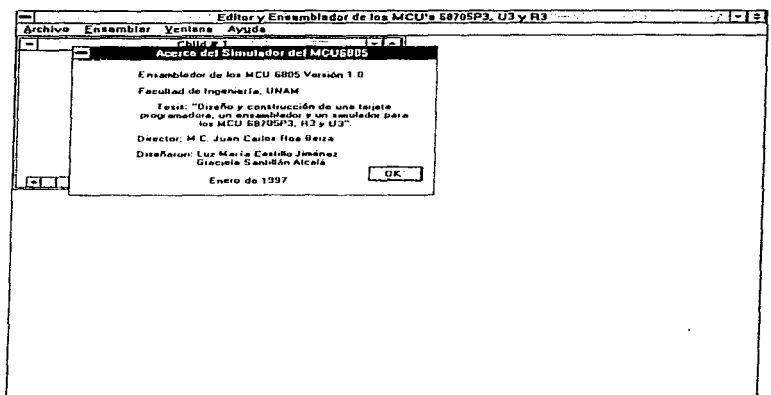


Figura 12 Acerca de...

MENSAJES DE ERROR

Si el programa no se ensambló correctamente pueden aparecer los siguientes errores (figura 13 y 14):

1. **Direccionamiento no permitido.**
   Indica que una instrucción no puede tener ese tipo de direccionamiento.
2. **Falta especificar dirección.**
   No se le indicó la dirección de alguna instrucción de salto.
3. **Dirección no válida**
   La dirección no se encuentra en el rango permitido o no es un número. hexadecimal.
4. **Dirección número 2 no válida.**
   Ver error 3.
5. **Dirección de inicio desconocida.**
   No se le indica el "ORG" de inicio del programa.
6. **Dirección relativa no válida.**
   La dirección relativa especificada no se encuentra en el rango permitido.
7. **Etiqueta no encontrada**
   La variable o etiqueta utilizada no está declarada en el programa.
8. **Bit no valido (Bit Set Clear)**
   El bit utilizado en la instrucción Bit Set Clear no está dentro del rango de 0 a 7.
9. **Operando no válido**
   El operando utilizado en un direccionamiento directo no se le especificó

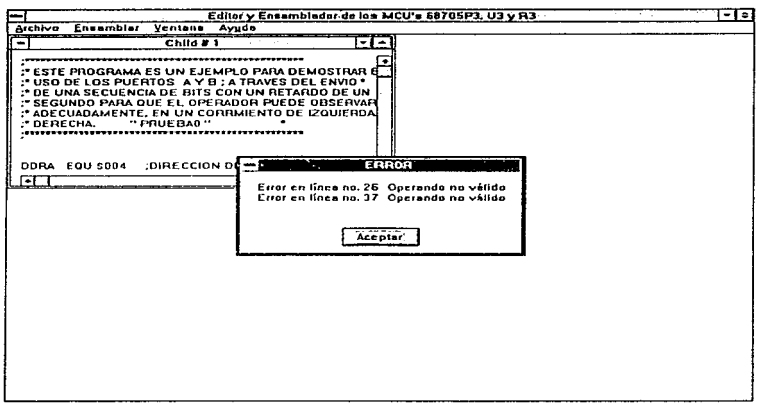


Figura 13 Mensajes de error

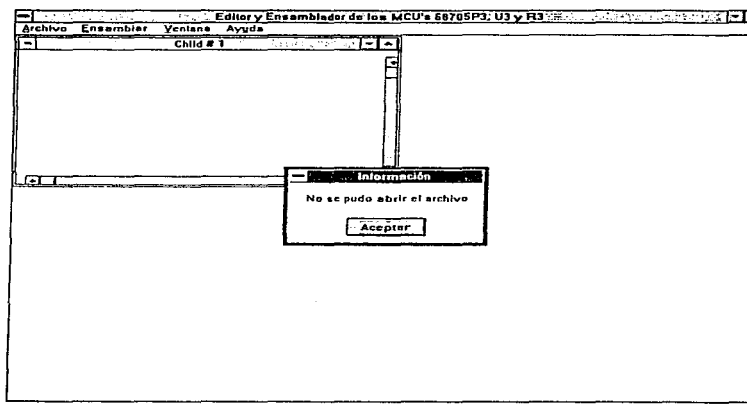


Figura 14 Mensajes de error

adecuadamente (#)

10. **No se puede abrir el archivo**
    El nombre del archivo o la ruta especificada son incorrectos o no existen.
11. **Nmónico no válido**
    Aparece cuando el nmónico del programa fuente no corresponde al conjunto de instrucciones permitidas.

NOTAS

Con la tecla ALT y cualquiera de las teclas indicadas en los títulos de los manejadores de menú y que están en negritas podrá ejecutar los elementos de menú de la misma manera que los ejecuta con el ratón.

Para abrir un archivo sea para captura de código o para abrir uno ya existente siempre hay que seleccionar la opción "Crear" del mismo menú.

Para grabar el programa objeto a la memoria hay que usar un convertidor de hexadecimal a binario, para que el archivo en binario sea utilizado en el grabador de memoria EPROM, se puede usar cualquier convertidor hexadecimal a binario que sea compatible con INTEL o TEXAS.

Simulador de los
MCU 68705P3, R3 y U3
Versión 1.0

Para ejecutar el simulador del los MCU seguir las siguientes instrucciones:

Se inicia de igual manera que en el Ensamblador.

Y al abrirse la ventana de simulador se podrá hacer lo siguiente:

1. Por medio del menú Abrir Archivo poner el nombre del archivo extensión prn y el directorio en el que se encuentra como el siguiente ejemplo:

   archivo: proy0.prn directorio: c:\roa\proyecto

   Si hubo un error se puede cancelar la operación por medio del botón "Cancel"

2. A continuación se carga el programa y se inicializan todos los datos del simulador.

3. Se puede ejecutar el Simulador por medio de la sección llamada "Controles de Simulación" que son:

Paso a Paso	Ejecuta las instrucciones línea por línea
Simula	Ejecuta las instrucciones de forma continua
Reset	Inicializa nuevamente el simulador
Paro	Detiene si se encuentra en proceso de simula

Salir Termina la simulación

Algunas instrucciones al ejecutarse se encontrarán inhibidas ya que no deberán ejecutarse en ese momento.

4. Se pueden realizar modificaciones de valor solamente en las siguientes partes de la ventana principal:

NOTA:

Para realizar un cambio primero borrar el elemento seleccionado y luego agregar el nuevo valor.

En el CPU:

-Acumulador A, tanto en la parte binaria como en la hexadecimal
-Índice X, tanto en la parte binaria como en la hexadecimal
-Timer Data o TDR, tanto en la parte binaria como en la hexadecimal
-Timer Control o TCR, tanto en la parte binaria como en la hexadecimal
-Códigos de condición, en todos sus bits indicados
-Contador del programa

En la memoria:

-Cualquier elemento de ella

Recomendación

NO USAR LAS DIRECCIONES

Entrada/Salida:

-DDR A, tanto en la parte binaria como en la hexadecimal
-PORT A, tanto en la parte binaria como en la hexadecimal
-DDR B, tanto en la parte binaria como en la hexadecimal
-PORT B, tanto en la parte binaria como en la hexadecimal
-DDR C, tanto en la parte binaria como en la hexadecimal
-PORT C, tanto en la parte binaria como en la hexadecimal
-PORT D, tanto en la parte binaria como en la hexadecimal

5. La Ayuda del simulador únicamente indica los datos de quienes crearon el simulador.

A continuación se muestran algunas figuras de las opciones del simulador.

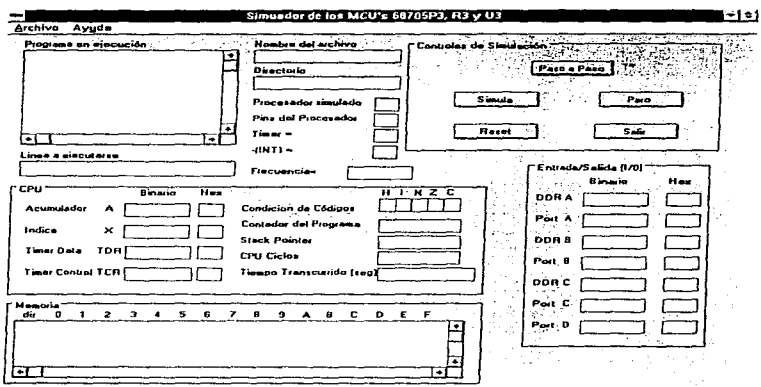


Figura 1 Ventana principal del simulador de los MCU 68705P3, R3 y U3

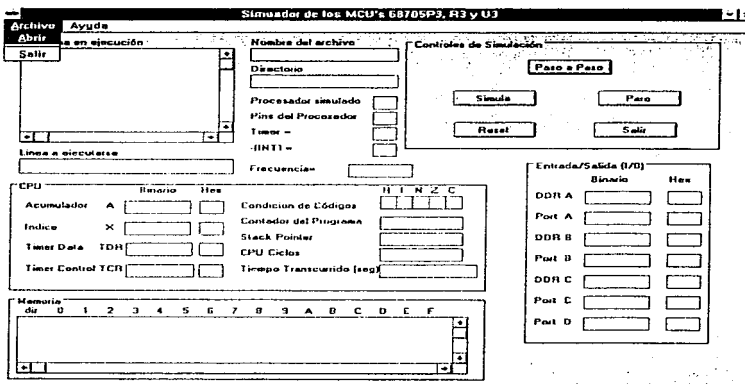


Figura 2a Opciones del menú. Archivo

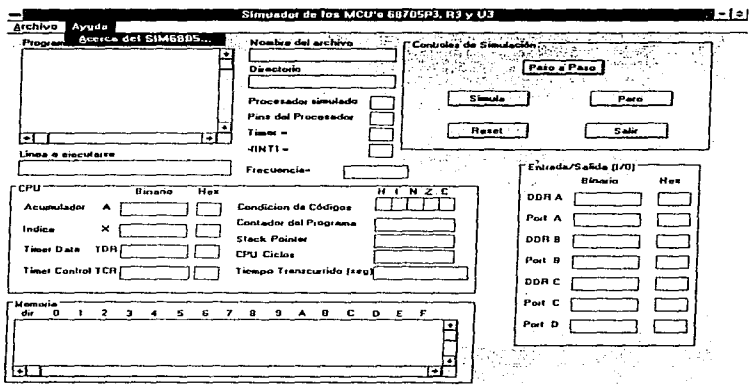


Figura 2b Opciónes del menú. Ayuda

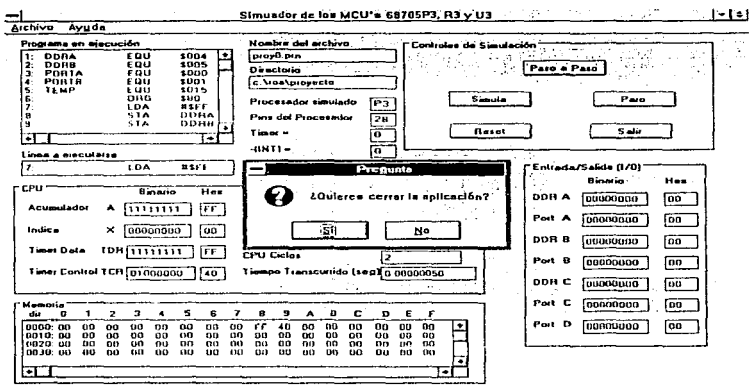


Figura 2c Botón Salir

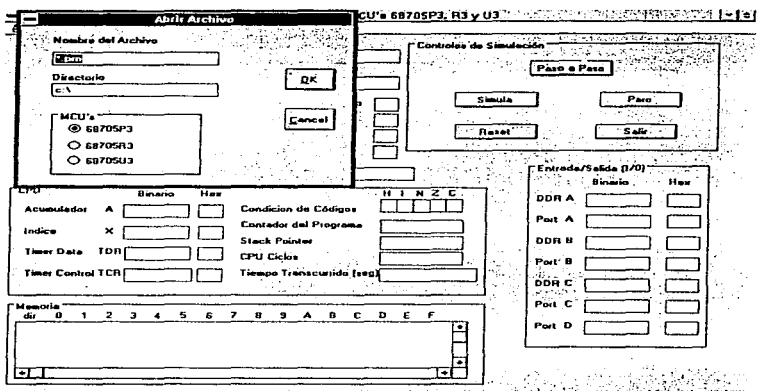


Figura 3a Opciónes del menú. Abrir

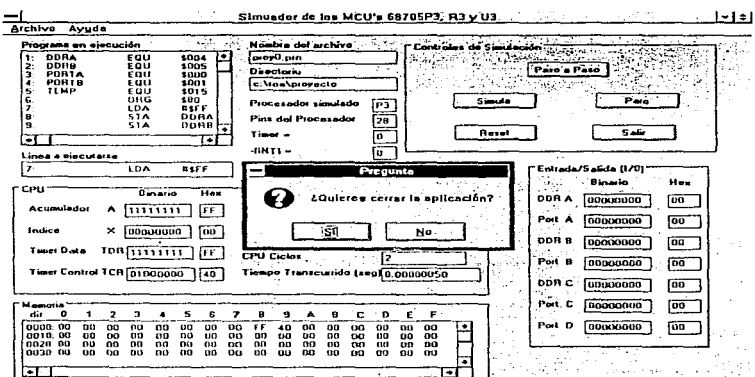


Figura 3b Opciónes del menú. Salir

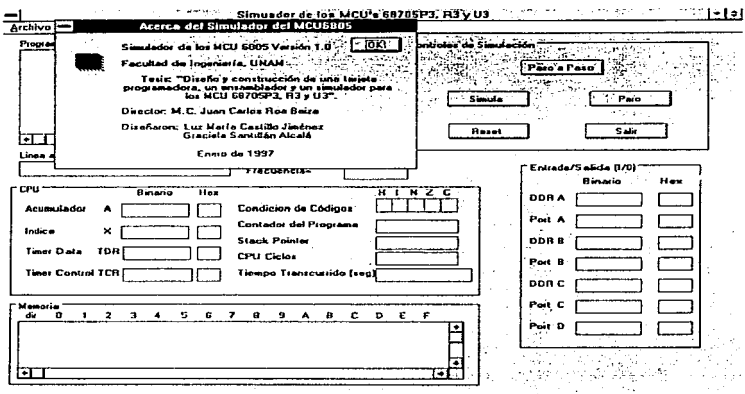


Figura 3c Opciónes del menú. Acerca del Simulador...

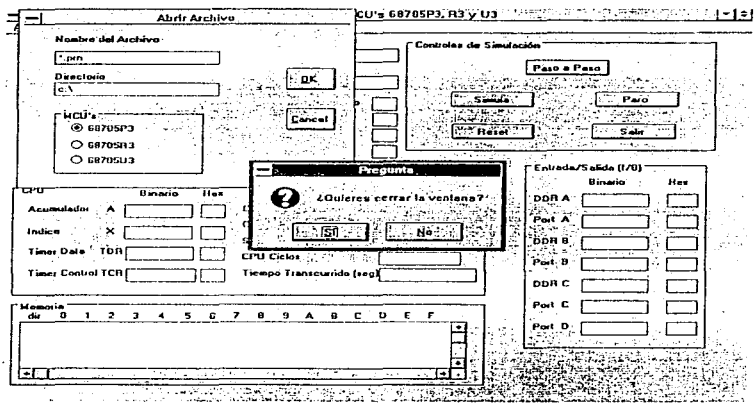


Figura 3d Opciónes del menú. Abrir archivo. Botón Cancel

Apéndice C

Programación de la EPROM de los MCU 68705P3, R3 y U3

Programación de la EPROM

Introducción

Generalidades

La familia M6805 HMOS/M146805 CMOS de MCU usa cualquier ROM enmascarada en el chip o EPROM en el chip para almacenar programas. La Memoria Programable Borrable Solamente para Lectura (EPROM) permite idear programas para ser escritos dentro de la memoria y, si se desea, después borrarla con luz ultravioleta y hacer correcciones.

Estas características da el usar una memoria alterable, no volátil. Cada EPROM en esta familia incluye una rutina bootstrap en la ROM enmascarada, la cual hace la programación relativamente fácil. Actualmente cuatro dispositivos EPROM existen, tres de los cuales son implementados en la familia M6805 CMOS.* Estos dispositivos pueden ser usados para emular varias versiones de ROM enmascarable de otros miembros de la familia. Los dispositivos EPROM tienen más capacidad que las versiones hechas de la ROM enmascarable, de este modo, permiten que algunos dispositivos EPROM emulen más de una versión de ROM.

Cada EPROM incluye un Registro de Opción Enmascarada (MOR) el cual es

implementado en la EPROM. El MOR esta localizado en la dirección $784 en el MC68705P3, $F38 en el MC68705R3, $F38 en el MC698705U3 y $1FF5 en el MC1468705G2. En la familia M6805 HMOS el MOR es usado para determinar cualquiera de las opciones del timer a ser usadas y seleccionar el circuito oscilador de reloj (cristal o RC), mientras que en la familia M146805 CMOS el MOR es usada para seleccionar el circuito oscilador de reloj, fija la relación de división de éste, así como el tipo de interrupción de entrada. Todas las localidades de la EPROM contienen ceros después de borrarla. La tabla 1 da una descripción de las funciones de cada bit del MOR utilizado en la familia M6805 HMOS y la tabla 2 provee una información equivalente del MOR para el MC7468705G2.

Bootstrap de la familia M6805 HMOS

Cada miembro de la familia M6805 HMOS de dispositivos EPROM contiene un programa bootstrap el cual es implementado dentro del chip en una ROM enmascarada. El reloj del programa bootstrap es a través de un contador externo el cual es usado para generar una dirección. La dirección entonces es usada para leer una localidad en una memoria externa. Los datos de la memoria externa son presentados a la EPROM vía un puerto de E/S. Después de que los datos de esta localidad son cargados en la EPROM, la rutina de reloj del bootstrap, el contador incrementa la dirección y lee la siguiente localidad. Después que todos los datos de todas las localidades son cargados en la EPROM del MCU su contenido es comparado con los de la memoria externa. El estado de la programación es indicado por dos LEDs (ver

la tabla 3).

Tabla 1. Registro de Opción Enmascarable de la familia M6805 HMOS

BIT	b7	b6	b5	b4	b3	b2	b1	b0
OPCION	CLK	TOPT	CLS			P2	P1	P0

b7, CLK	Tipo de reloj oscilador 1 = RC 0 = Cristal NOTA V_{mrr} en el TIMER/BOOT pin (8) forza el modo de cristal.
b6, TOPT	Opción del Timer 1 = Timer/prescalador. Todos los bits excepto el 3, 6, 7 del registro de control del timer (TCR) son invisibles al usuario. Los bits 5, 2, 1 y 0 de la opción del MOR determinan la equivalencia de las opciones enmascarables de la familia M6805 HMOS. 0 = Todos los bits del TCR son activados para programar al timer por software. EL estado de los bits del MOR 5,4,2,1,0 fija el valor inicial de los bits respectivos del TCR (el TCR es controlado por software después de la inicialización).

<table>
<tr><td>b5, CLS</td><td>Fuente del Timer/Reloj
1 = pin TIMER externo
0 = interno ϕ_2</td></tr>
<tr><td>b4</td><td>No usado si MOR TOPT =1
Fija el valor inicial del TIE del TCR si MOR TOPT = 0</td></tr>
<tr><td>b3</td><td>No usado</td></tr>
<tr><td>b2, P2
b1, P1
b0, P0</td><td>Opción del Prescalador. Los niveles lógicos de esos bits, selecciona una de las 8 etapas en el prescalador del timer. La división resultante de codificar las combinaciones de esos tres bits se muestran aquí:
Prescalador
<table>
<tr><th>TCR2</th><th>TCR1</th><th>TCR0</th><th>División</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>4</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>8</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>16</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>32</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>64</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>128</td></tr>
</table></td></tr>
</table>

Tabla 2. Registro de Opción Enmascarable de la familia M146805 CMOS

BIT	b7	b6	b5	b4	b3	b2	b1	b0
OPCION	CLK	DIV		INT				

b7, CLK	Tipo de reloj oscilador 1 = RC 0 = Cristal
b6, DIV	Determina la división de la oscilación del reloj 1 = Divide en 2 el reloj del oscilador 0 = Divide en 4 el reloj del oscilador
b5, CLS	No usado
b4, INT	Determina el tipo de interrupción de entrada. 1 = Ambos edge-sensitive y level-sensitive. 0 = Interrupción Edge-sensitive solamente.
b3, b2, b1, b0	No usados

Tabla 3. Led de resultados de la EPROM del M6805 HMOS.

LED	
DS1(PB1)	Prendido (cuando **PB1** baja) indica que la EPROM esta siendo programada.
DS2(PB2)	Prendido (cuando **PB2** baja) indica que el contenido de la EPROM es verificado sucesivamente (aproximadamente 2 segundos después de que DS1 es encendido). La programación y verificación han terminado.

Programación

Familia M6805 HMOS

La figura 1 contiene un diagrama esquemático de un circuito el cual puede ser usado para programar la EPROM de los MCU's MC68705P3, MC68705R3, MC68705U3.

La figura 2a y 2b muestra la tableta de circuito impreso y la tabla 4 provee una lista de las partes.

Excepto por el socket usado para montar el dispositivo MCU EPROM, todos los MCU's pueden usar una EPROM de 2k (MCM2716) o 4k (MCM2532) para el U2, la

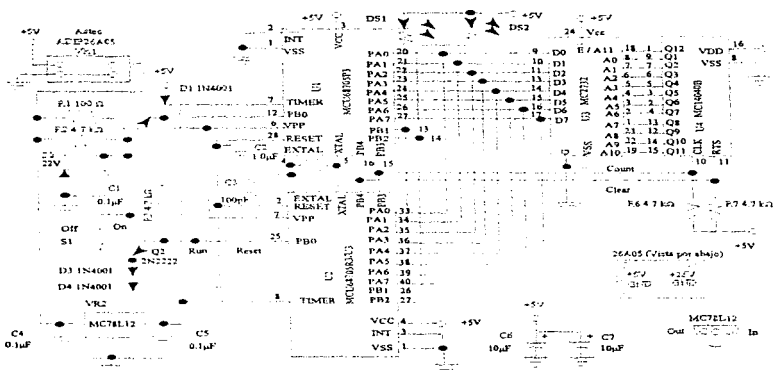


Figura 1 Diagrama esquemático de los componentes del circuito

Figuras 2a y 2b Frente y vuelta del circuito impreso

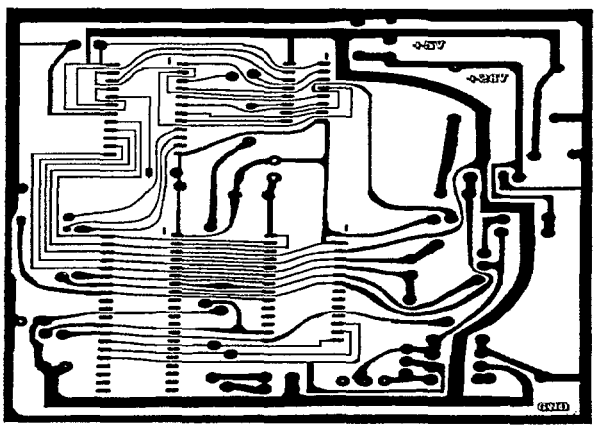


Figura 2a

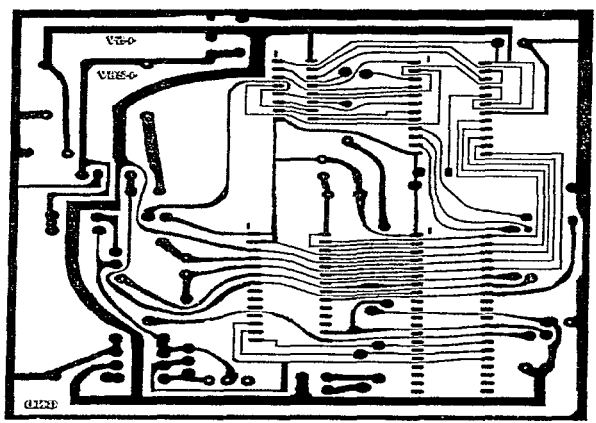


Figura 2b

Tabla 4. Lista de partes para la construcción de la tarjeta programadora

R1	100 kΩ
R2	4.7 kΩ
R3	4.7 kΩ
R4	510 Ω
R5	510 Ω
R6	4.7 kΩ
R7	4.7 kΩ
C1	0.1 µF
C2	1.0 µF
C3	100 pF
C4	1.0 µF
C5	1.0 µF
C6	10 µF
C7	10 µF
D1	1N4001
D2	22V Zener (1N4748A)
D3	1N4001
D4	1N4001

Q1	2N2222
Q2	2N2222
Y1	1 Mhz (100Ω máx.)
U1	MC68705P3
U2	MC68705R3/U3
U3	MCM2732
U4	MC14040B
VR1	ASTEC Convertidor de voltaje 26A05
VR2	MC78L12
DS1	LED Rojo
DS2	LED Verde
PCB1	Tableta fenólica 15x15cm
Misc:	1 Base 40 pines con palanca
	1 Base 28 pines con palanca
	1 Base 24 pines con palanca
	1 Base 16 pines con palanca
	2 Switches

programación para cualquiera de los MCU EPROM es básicamente la misma, el procedimiento para la programación del MC68705P3 es descrita primero y en seguida el procedimiento para el MC68705R3/U3.

Programación del MC68705P3

El programa bootstrap del MCU puede usarse para programar la EPROM del MCU.

Una EPROM 2732 UV primero debe programarse con la misma información que se transferirá a la EPROM del MCU.

Antes de la programación de la EPROM MC68705P3 debe borrarse con la exposición de una luz ultravioleta de alta intensidad (UV) con una longitud de onda de 2537 angstrom. La dosis recomendada (intensidad x tiempo de exposición de UV) es de 15Ws/cm^2 . La lámpara de UV debe ser usada sin filtros de onda corta y el MC68705P3 puede posicionarse a una pulgada de los tubos UV. Se recomienda proteger la ventana de la EPROM de la luz excepto cuando se borre.

La EPROM MCM2732 es insertada en el U3 durante la programación del MC68705P3. Antes de que el MC68705P3 sea programado la EPROM MCM2732 UV deberá primero ser programada con una copia exacta de la información que será transferida al MC68705P3.

NOTA

Los primeros 128 bytes de la EPROM (MCM2732) son ignorados; la localidad

$80 de la EPROM es puesta en la localidad $80 del MC68705P3.

Paso 1. Cerrar los interruptores S1 y S2 y que el voltaje (+5 V en este caso) no sea aplicado a la tableta del circuito.

Paso 2. Insertar el MCM2732 en el socket U3 e insertar el MC68705P3 dentro del socket U1.

Paso 3. Aplicar +5 V a la tableta del circuito.

Paso 4. Abrir el interruptor S1 para aplicar V_{PP} al MCU y entonces abrir el interruptor S2 para quitar el reset.

NOTA

Solo inserte un tipo de microcontrolador a la vez, para evitar la destrucción de su tarjeta programadora.

Una vez que el MCU sale de reset, la línea de control de salida CLEAR (PB4) genera un pulso, entonces el contador MC14040B es sincronizado por la salida PB3 (COUNT). El contador selecciona el byte de la EPROM MCM2732 la cual esta lista para cargar el byte equivalente al MC68705P3 EPROM por el programa bootstrap del MCU. Una vez que el dato es programado, el COUNT incrementa el contador a la siguiente localidad. Esto continua hasta que el MCU es programado completamente.

Paso 5. Revisar que el LED indicador de la programación encienda y en seguida el LED indicador de la verificación. Estas señales indican que la EPROM del MPU ha sido programada correctamente.

Paso 6. Cerrar el interruptor S1 para quitar el V_{PP} y V_{HTP}. Cerrar el interruptor S2 para reinicializar el MCU.

Paso 7. Desconectar (o apagar) la entrada de +5 V a la tableta del circuito, entonces quitar nuevamente la EPROM de su socket.

Paso 8. Quitar la EPROM del socket U3 si no es requerido en la programación.

Programación del MC68705R3/MC68705U3

Programación

El programa bootstrap del MCU puede usarse para programar la EPROM del MCU. Los vectores alternados utilizados para implementar la autoverificación se utilizan al comenzar la ejecución del programa bootstrap.

El programa bootstrap del MCU puede ser usado para programar la EPROM del MCU. EL vector alterno usado para implementar la autorevisión, es usado para iniciar la ejecución del programa bootstrap.

La EPROM UV MCM2732 (otro estándar industrial de la EPROM puede ser usado) deberá primero ser programado con la misma información que será transferida a la EPROM del MCU.

La programación de cualquiera de las EPROM de los MCU es similar al que se describió para el MC68705P3 con tres pequeñas excepciones.

Estas son:

1. El MCM2732 EPROM UV es colocado en el socket U3 durante la programación de cualquiera de las EPROM de los MCU MC68705R3 o MC68705U3. Esta EPROM UV deberá ser programada con una copia exacta de la información para ser transferida a el MC68705R3 o MC68705U3.

2. En el paso 2 el MCM2732 es insertado dentro del socket U3 y el MC68705U3 es insertado dentro del socket U2.

3. En la nota bajo el paso 4, la operación del MCM2732 y el MC68705R3/U3 es idéntica a la que se describió para el MCM2732 y MC68705P3.

Emulación

El MC68705P3 emula al MC6805P2 y MC6805P6 "exactamente". El MC6805P2/P6 toma características implementadas en el registro de opción enmascarable (MOR), las de la EPROM hacen lo mismo en el MC68705P3, Una minoría de excepciones de emulación se listan a continuación:

1. El área de ROM futura en el MC68705P2/P6 se implementa en el MC68705P3, y estos 704 bytes pueden no programarse para simular exactamente el MC6805P2/P6. El MC6805P2/P6 lee todos los ceros de esta área.

2. Las áreas reservadas de ROM en el MC6805P2/P6 y el MC68705P3 tienen diferentes tipos de almacenamiento de datos. Este dato esta sujeto a cambiar sin aviso. El MC6805P2/P6 usa la ROM reservada para autoverificar sus características, y el MC68705P3 utiliza esta área para el programa bootstrap.

3. El MC6805 primero lee toda el área de sus 48 bytes (característica de la RAM). Esta RAM no se implementa, pero si se implementa en las versiones enmascarables del MC6805P2/P6.

4. La línea del V_{PP} (pin 6) en el MC68705P3 debe vincularse con V_{CC} para un modo de operación normal. En el MC6805P2/P6 el pin 6 es el NUM y es tierra en el modo de operación normal.

5. Las características del LCI no se habilitan en el P3, procesando diferencias no

presenta compatibilidad con el diseño propio de sus características en la versión EPROM.

La operación del resto del circuito se duplica exactamente en sus dispositivos incluyendo interrupciones, timer, puertos de datos y los registros de direccionamiento de dirección de datos (DDR). Una meta del diseño es el proveer al usuario con un ahorro mediante una vía económica para verificar el diseño de un programa y un sistema antes de que se programe en la ROM de fábrica.

MC68705U3

El MCU68705U3 emula exactamente los MCU's MC6805U2 y el MC6805U3. EL MC6805U2 y MC6805U3 de características enmascarables son implementados en la opción de registro enmascarado de los bytes de la EPROM.

Excepciones a la emulación exacta

1. Las áreas del MC6805U2 "futura ROM" son implementadas en el MC68705U3 y 1728 bytes pueden ser dejados sin programar al simular con precisión el MC6805U2.

2. Las áreas reservadas de la ROM tienen diferentes datos almacenados en ellos. En el MC6805U2 estas áreas son usadas para la autorevisión y en el

MC68705U3 esta área es usada para programar el bootstrap.

3. El MC6805U2 lee puros unos en el byte 48 "futura área de la RAM". Esta área no es implementada en las versiones de RAM enmascaradas de los MC6805U2/U3, pero es implementada en la MC68705U3.

4. La línea de V_{pp} (pin 7) del MC68705U3 es vinculado al V_{CC} durante la operación normal. En MC6805U2, este pin es conectado a tierra durante operaciones normales y en el MC6805U3, este pin no es conectado.

Apéndice D

Listados de la programación

Código del ensamblador de los MCU 68705P3, R3 y U3

Versión 1.0

```
//Programa CLASES.CPP
//Diseñado por: Luz María Castillo Jiménez

#include "clases.h"

////////////////////////////////////////////////////////////////////////
void tablacomp::impr(void)
{

cout << pc << opcod << etiq << nmo << oper << tipins;
}

////////////////////////////////////////////////////////////////////////
void tablacomp::salvar(FILE *arch, int b)
{
  if(b)
  {
    fprintf(arch," %4s %4s     %4s   %3s %4s     %s
\n",pc_ens,cod_ens, etiq, nmo, oper, coment);
  }
  else
  {
    fprintf(arch,"  %6s          %4s    %3s  %4s
%s\n",cod_ens, etiq, nmo, oper, coment);
  }
  /* close the file */
}
////////////////////////////////////////////////////////////////////////
void tablacomp::salvar_mik(FILE *arch, int b, int &total,
int & pc_lin)
{
  int c_car=0;
  char n_h="\0";
  char c[10];
  char ax[10];
  int exces;
  int len;
  int comp;

  limp_aux(ax,10);
  limp_aux(c,10);

  comp = strcmp(nmo, "ORG");

    total=0;
    fprintf(arch,"\n%s",pc_ens);
    c_car = strlen(pc_ens);
    pc_lin=pc;
    total+=c_car;
    c_car=strlen(cod_ens);
    total+=c_car;
    if(total>=36)
    {
      pc_lin+=(36/2)-2;
      strupr(_itoa(pc_lin,n_h,16));
     len=strlen(n_h);
     if(len<4)
     {
       strcpy(c,"0");
       for(int i=0; i<(4-len); i++)
       {
         strcat(c,"0");
       }
       strcat(c,n_h);
     }
     exces=total-36;
     if(exces==0)
     {
       fprintf(arch,"%s",cod_ens);
       fprintf(arch,"\n%s",c);
       total=0;
     }
     else{
     for(int i=0; i<(c_car-exces); i++)
     {
       ax[i]=cod_ens[i];
     }
     fprintf(arch, "%s", ax);
     limp_aux(ax,4);
     for(i=c_car; i<=c_car-exces+1 ;i--)
     {
       ax[4-i]=cod_ens[i];
     }
     fprintf(arch, "\n%s%s",c, ax);
     total=0;
    }
    }
    else
    {
      fprintf(arch,"%s",cod_ens);
    }
  }
}
////////////////////////////////////////////////////////////////////////
void tablacomp::tabla_simb(FILE *arch)
{
  char pcet_h[10];
  char scat[10];
  char scatt[10];

  strcpy(scat,"00");
  strupr(_itoa(pcet,pcet_h,16));
  strcat(scat,pcet_h);
  int i=cuenta_car(scat);
  if(i<4)
  {

    strcpy(scatt,scat);
    for(int h=0; h<10; h++)
    {
      scat[h]='\0';
    }
    for(int i=0; i<4-i; i++)
    {
      strcat(scat,"0");
    }
    strcat(scat,scatt);

  }

  fprintf(arch,"%s \t %s \n", etiq, scat);

  /* close the file */
}
////////////////////////////////////////////////////////////////////////

//La siguiente función se ejecuta solamente si falla new
void newError(void)
{
  cerr << "Memoria  insuficiente";
  exit(1);   //Termina el programa
}

////////////////////////////////////////////////////////////////////////
int tablacomp::ext_pc(void)
{
  return(pc);
}
////////////////////////////////////////////////////////////////////////
int tablacomp::ext_pcet(void)
```

```
{
  return(pcet);
}
//////////////////////////////////////////////////////////////////////////
int tablacomp::ext_tipdir(void)
{
  return(tipdir);
}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_etiq(char cad[])
{
  strcpy(etiq,cad);

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_tipdir(int d)
{
  tipdir = d;

}
//////////////////////////////////////////////////////////////////////////
void  tablacomp::ext_nmo(char aux_nmo[])

{
  strcpy(aux_nmo,nmo);
}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_oper(char cad[])
{
  strcpy(oper,cad);

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_tipins(char cad)
{
  tipins = cad;

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_opcod(char cad[])
{
  strcpy(opcod,cad);

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_nmo(char cad[])
{
  strcpy(nmo,cad);

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_coment(char cad[])
{
  strcpy(coment,cad);

}

//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_pcet(int pc)
{
  pcet = pc;

}
//////////////////////////////////////////////////////////////////////////
void tablacomp::asig_pc_ens(char* pc)
{
  strcpy(pc_ens,pc);

}
//////////////////////////////////////////////////////////////////////////
```

```
void tablacomp:: pci(int l)
{
  pc = l;
}
//////////////////////////////////////////////////////////////////////////
int verifnmo(tablacomp *Nmo, instruc * INSTRUCC, int &
comp)
{
  comp = strcmp(Nmo->nmo,INSTRUCC->nmo);
  if (!(comp))
  {
    return (1);
  }
  return 0;
}

//////////////////////////////////////////////////////////////////////////
char instruc ::veritip(void)
{
  return(tipins);
}
//////////////////////////////////////////////////////////////////////////
char * instruc ::veritopcod(void)
{
  return(opcod);
}
//////////////////////////////////////////////////////////////////////////
int tablacomp::veritopi(tablacomp *tabla1, int  &sum)
{
  int n;
  char * str = "";
  char aux[70] = "\0";
  int i;
  int T = 0;
  int b = 0;

  sum = 0;
  switch (oper[0])
  {
    case '#':
    {
      tabla1->inc_sig_pc(2,pc);
      T = 1; tipdir = 2;
    }break;
    case ',':
    {
      tabla1->inc_sig_pc(1,pc); T = 1; tipdir = 5;
    }break;
    case '$':
    {
            i = 1;
            i = ut_aux(i,oper,aux,' ');
            b = conv_dec(aux,n);
            if(b) //si fue un digito
            {
              if(ver_val(n,sum,256)) tipdir = 3;
              if(!ver_val(n,sum,256)) tipdir = 4;
              T = 1;
            }
      if (!b)
            {
            //cerr << "Operando no válido \n";
            T = 0;
            }
            // T = 1;
    } break;
  }//switch
if(T == 0)
{
    switch(tipins)
    {
          case 'C':
```

```
                {
                  tabla1->inc_sig_pc(2,pc); T = 1; tipdir = 8;
                } break;
                case 'D':
                {
                  tabla1->inc_sig_pc(3,pc); T = 1; tipdir = 9;
                } break;
                case 'F':
                {
                  tabla1->inc_sig_pc(2,pc); T = 1; tipdir = 10;
                } break;
                case 'A':
                {
                  T = 0;
                } break;
                case 'E':
                {                       /*    if(!(strcmp(nmo,"JSR"))
||!(strcmp(nmo,"JMP")))
                  {
                    tabla1->inc_sig_pc(3,pc);
                    T = 1;
                    tipdir = 4;
                  } */
                  T = 0;
                } break;
              } //switch
            }//if
            if (T == 1) return T;
            return T;
          }
////////////////////////////////////////////////////////////////////////////
void tablacomp::verifop2(int i, tablacomp *tabla1[], int &
sum,CString & mensaje, int & error)
{
  int n,nn,dir = 0;
  char aux[70] = "\0";
  int i = 0;
  int b = 0;
  char str[7] = "\0";
  strcpy(str,oper);
  char *n_h="";
  char temp[5];

  for(int h = 0; h < 5; h + +)
  {
    temp[h] = '\0';
  }

  b = conv_dec(str,n);
  if !b)
  {
    i = ut_aux(i,oper,aux,',');
    b = conv_dec(aux,nn);
    if(b)
    {
      if(ver_val(nn,sum,256)) tipdir = 6;
      if(!ver_val(nn,sum,256)) tipdir = 7;
    }
  }
  if(!b)
  {
    i = 0;
    if(oper[strlen(oper)-1] = = 'X')
    {
      i = ut_aux(i,oper,aux,',');
            int w = 0;
            dir = -1;
            if(aux[0] = = '$')
            {
              ut_aux(1,aux,temp,' ');
              b = conv_dec(temp,n);
                if(b)
                {
                  if(ver_val(n,sum,256)) tipdir = 7;
                  if(!ver_val(n,sum,256)) tipdir = 6;
                }
                else
                {
                  msj_err(9,i,mensaje,error);
                }
              }
            else
              {
          while(dir = = -1 &&  w < = l)
          {
                dir = tabla1[w]->bus_et(aux);
                w = w + 1;
          }
              if(dir! = -1)
              {
                if(ver_val(dir,sum,100)) tipdir = 7;
                if(!ver_val(dir,sum,100))) tipdir = 6;
              }
              if(dir = = -1)
          {
                nn = atoi(aux);
                if(ver_val(nn,sum,256)) tipdir = 6;
                if(!ver_val(nn,sum,256)) tipdir = 7;
                 strupf( itoa(nn,n_h,16));
                 strcat(cód_ens, n_h);
          }
        }
            /*    strupf( itoa(nn,aux,16))
              //if(b)
              {
                if(ver_val(nn,sum,256)) tipdir = 6;
                if(!ver_val(nn,sum,256)) tipdir = 7;*/
            //  }
    //}                                   //encontrar las hojas
cuadro.
  // }
    else
    {
      int w = 0;
      dir = -1;
      while(dir = = -1 &&  w < = l)
      {
            dir = tabla1[w]->bus_et(oper);
            w = w + 1;
      }

      else
      {
        if(!(strcmp(nmo,"JSR")) ||!(strcmp(nmo,"JMP")))
        {
          tabla1[i + 1]->inc_sig_pc(3,pc);
          // T = 1;
          tipdir = 4;
        }
        else msj_err(9,i,mensaje, error);
      } // carr < < "Operando no válido";
    }
  }
}
```

```
////////////////////////////////////////////////////////////////////////
int tablacomp::bus_et(char Oper[])
{
  int c = 1;
  int diraux;

  c = strcmp(Oper, etiq);
  if(c == 0)
  {
     diraux = pcet;
  }
  if(c1 = 0)
  {
     diraux = -1;
  }
  return(diraux);
}

////////////////////////////////////////////////////////////////////////
void instruc::init(char Nmo[], char Opcod[], char Tipins)
{
  strcpy(nmo,Nmo);
  strcpy(opcod,Opcod);
  tipins = Tipins;
}
////////////////////////////////////////////////////////////////////////
int tablacomp::ventbl(char cad[])
{
  if(strlen(cad) == 0)  return 0;
  else return 1;
}
////////////////////////////////////////////////////////////////////////
int tablacomp::verifet(char cad[])
{
  if(cad[0] == ' ' || cad[0] == '\t') return 0;
  else
     return 1;
}
////////////////////////////////////////////////////////////////////////
int ut_aux(int i, char CAD[],char aux[],char c_fin)
{
 int j = 0;
 if(c_fin == ' ')
 {
  while(CAD[i] != c_fin && CAD[i] != '\t' && CAD[i] != '\0')
  {
     aux[j] = CAD[i];
     i++;
     j++;
  }
 }
 else
 {

     while(CAD[i] != c_fin && CAD[i] != '\0')
  {
     aux[j] = CAD[i];
     i++;
     j++;
     }
 }
 //}
 aux[j] = '\0';
 return i;
}
////////////////////////////////////////////////////////////////////////
void limp_aux(char aux[], int tam)
{
  for (int h = 0; h<tam; h++)
  {
    aux[h] = '\0';
  }
}
////////////////////////////////////////////////////////////////////////
int tablacomp::recorre(int i, char cad[])
{
  while((cad[i] == ' ' || cad[i] == '\t') && cad[i] != '\0')
  {
     i++;
  }
  return i;
}
////////////////////////////////////////////////////////////////////////
void tablacomp::inc_sig_pc(int i_com, int pcm)
{
  pc = pcm + i_com;
}
////////////////////////////////////////////////////////////////////////
int tablacomp::ver_val(int n, int &sum, int n_compara)
//////
{
  if( n > n_compara) {sum = pc + 3; return 1;}  //3 bytes
  if( n < n_compara) {sum = pc + 2; return 0;}
  return 0;
}
////////////////////////////////////////////////////////////////////////
int tablacomp::toma_lin2(int i,int Lin, tablacomp * Tabla1[],
CString & mensaje, int & error)
{
  char temp[5] = "\0";
  char temp2[5] = "\0";

    if(!strcmp(Tabla1[i]->nmo,"EQU"))
    {
       strcpy(temp2,Tabla1[i]->oper);
       strcpy(Tabla1[i]->cod_ens,Tabla1[i]->opcod);
       ut_aux(1,temp2,temp," ");
       strcat(Tabla1[i]->cod_ens,temp);
    }
    if(!strcmp(Tabla1[i]->nmo,"END"))
    {
       strcpy(Tabla1[i]->cod_ens,"");
    }
                        if(!strcmp(Tabla1[i]->nmo,"DB")    ||
!strcmp(Tabla1[i]->nmo,"DW"))
    {
       strcpy(Tabla1[i]->cod_ens,Tabla1[i]->opcod);
    }
    if(!strcmp(Tabla1[i]->nmo,"ORG"))
    {
       strcpy(Tabla1[i]->cod_ens,"");
    }
    else{
    switch(Tabla1[i]->tipins)
    {
          case 'A':
          {
            switch (Tabla1[i]->tipdir)
            {
               case 1: Tabla1[i]->direc1(); break;
                       case 3:  Tabla1[i]->direc3(Lin,Tabla1,i,
mensaje, error); break;
               case 5: Tabla1[i]->direc5(); break;
                       case 6:  Tabla1[i]->direc6(Lin,  Tabla1,i,
mensaje, error); break;
                       default: msj_err(1,i,mensaje, error);
            } //switch tipdir
          } //case A
          break;

          case 'B':
          {
            switch (Tabla1[i]->tipdir)
            {
```

```
                case 1:{Tabla1[i]->direc1();} break;
                default: msj_err(1,i,mensaje, error);
              }


            } //case B
            break;

            case 'C':
            {
              switch (Tabla1[i]->tipdir)
              {
                    case 8: {Tabla1[i]->direc8(Lin, Tabla1,i,
mensaje, error);} break;
                default: msj_err(1,i,mensaje, error);
              }
            } //case C
            break;

            case 'D':
            {
              switch (Tabla1[i]->tipdir)
              {
                    case 9: {Tabla1[i]->direc9(Lin, Tabla1,i,
mensaje, error);} break;
                default: msj_err(1,i,mensaje, error);
              }

            } //case D
            break;

            case 'E':
            {
        char *n_h = "";
        int dir = 1;
        int w = 0;

              if(!strcmp(Tabla1[i]->nmo,"JSR") ||
                 !strcmp(Tabla1[i]->nmo,"JMP"))
              {
          strcpy(cod_ens, opcod);
          while(dir != -1 && w <= Lin)
          {
            dir = Tabla1[w]->bus_et(oper);
            w++;
          }
          if(dir != -1)
          {
            strupr(_itoa(dir,n_h,16));
            int l;
            l=strlen(n_h);
            if(l<4)
            for(int h = 0; h<4-l; h++)
            {
              strcat(cod_ens,"0");
            }
            strcat(cod_ens,n_h);
          }
        }
        else
        {
              switch (Tabla1[i]->tipdir)
              {
                case 2: Tabla1[i]->direc2(Lin, Tabla1); break;
                case 3: Tabla1[i]->direc3(Lin, Tabla1,i,
mensaje, error); break;
                case 4: Tabla1[i]->direc4(Lin, Tabla1,i,
mensaje, error); break;
                case 5: Tabla1[i]->direc5(); break;
                case 6: Tabla1[i]->direc6(Lin, Tabla1,i,
mensaje,error); break;
                case 7: Tabla1[i]->direc7(Lin, Tabla1,i,
mensaje, error); break;
                default: msj_err(1,i,mensaje, error);
              } //switch tipdir
            }

            } //case E
            break;

            case 'F':
            {
              switch (Tabla1[i]->tipdir)
              {
                    case 10: Tabla1[i]->direc10(Lin, Tabla1,i,
mensaje, error); break;
                default: msj_err(1,i,mensaje, error);
              }
            }
  } //switch tipins

 }
 return 0;
}

////////////////////////////////////////////////////////////////////
int tablacomp::salto_rel(int Dir_et, int PC, int &n_bytes)
{
  int s = 0;
  if(Dir_et<PC)
  {
    s= 256-(PC-Dir_et);
    n_bytes = Dir_et-PC;
    s-= 2;
    return s;
  }
  if(Dir_et>PC)
  {
    s = Dir_et-PC;
    n_bytes = s;
    s-= 2;
    return s;
  }
  return -1;
}
////////////////////////////////////////////////////////////////////
int tablacomp::cuenta_car(char C[])
{
  int h = 0;
  while(C[h] != '\0' && h<5) h++;
  return h;
}
////////////////////////////////////////////////////////////////////
int tablacomp::val_asc(char C[], int i)
{
  if (C[i]>=48 && C[i]<=57 )
  {
    return (C[i]-48);
  }
  if(C[i]>=65 && C[i]<=70)
  {
    return(C[i]-55);
  }
  else
  {
          return -1;
  }
}
////////////////////////////////////////////////////////////////////
int tablacomp::conv_dec(char *Car, int &entero)
{
```

```
    int tam;
    int pot;
    int val;

    entero = 0;
    tam = strlen(Car);

    pot = int(pow(double(16),double(tam-1)));

    for(int i = 0;i < tam;i + +)
    {
      val = val_asc(Car,i);
      if(val = = -1)
      {
        entero = 0;
        return 0;
      }

      else
      {
        entero + = val * pot;
        pot = pot/16;
      }
    }
    return 1;
}

////////////////////////////////////////////////////////////////////////////
void tablacomp::direc1(void)
{

    strcpy(cod_ens,opcod);
}

////////////////////////////////////////////////////////////////////////////
void tablacomp::direc2(int Lin, tablacomp * tabla1[])
{
    int dir = -1;
    int w = 0;
    char aux[70] = "\0";
    int n = 0;
    char *n_h = "";
    char *dir_h = "";

    strcpy(cod_ens,opcod);

    if(oper[1] = = '$')
    {
      ut_aux(2,oper,aux,' ');
      strcat(cod_ens, aux);
    }
    else
    {
      while(dir = = -1 && w < = Lin)
      {
        dir = tabla1[w]->bus_et(oper);
              w + = 1;
      }
      if(dir = = -1)
      {
              ut_aux(1,oper,aux,' ');
              n = atoi(aux);
              strupr(_itoa(n,n_h,16));
              strcat(cod_ens, n_h);
      }
      if(dir! = -1)
      {
        strupr(_itoa(dir,dir_h,16));
              strcat(cod_ens,dir_h);
      }
    }
    limp_aux(aux,70);
}
////////////////////////////////////////////////////////////////////////////
void tablacomp::direc3(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
    int dir = -1;
    int w = 0;
    char *dir_h = "";
    char v_opl[10];

    for(int h = 0; h < 10; h + +)
    {
      cod_ens[h] = '\0';
    }

    strcpy(cod_ens,opcod);
    if (oper[0] = = '$')
    {
      int j = strlen(oper);
      for(int h = 0; h < 10; h + +)
      {
        v_opl[h] = '\0';
      }
      for(int i = 1; i < j; i + +)
      {
        v_opl[i-1] = oper[i];
      }
      strcat(cod_ens,v_opl);
    } //if
    else
    {
      while(dir = = -1 && w < = Lin)
      {
        dir = tabla1[w]->bus_et(oper);
              w + = 1;
      }

              _strupr(_itoa(dir,dir_h,16));
              int j = 0;

              j = strlen(dir_h);
        if( j < 2)
              {
                cod_ens[2] = '0';
              }
              strcat(cod_ens,dir_h);


              msj_err(9,lin,mensaje, error);
    }
}
////////////////////////////////////////////////////////////////////////////
void tablacomp::direc4(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
    int dir = -1;
    int w = 0;
    char *n_h = "";
    char aux[70] = "\0";

    strcpy(cod_ens, opcod);

    if(oper[0] = = '$')
    {
      ut_aux(1,oper,aux,Lin);
```

```
    strcat(cod_ens,aux);
  }
  else
  {
    ut_aux(0,oper,aux,' ');
    while(dir = = -1 && w < = Lin)
    {
            dir = tabla1[w]-> bus_et(aux);
            w + = 1;
    }
    if (dir = = -1)
    {
            msj_err(7,lin,mensaje, error);
    }
    else
    {
             _strupr( itoa(dir, n_h,16));
            strcat(cod_ens,n_h);
    }
  }
  limp_aux(aux,70);
}
//////////////////////////////////////////////////////////////////////
void tablacomp::direc5(void)
{
  strcpy(cod_ens, opcod);
}
//////////////////////////////////////////////////////////////////////
void tablacomp::direc8(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
  char *str = "";
  int checa_h = 1;
  int i = 0;
  char aux[70] = "\0";
  int n = 0;
  int dir = -1;
  int w = 0;
  char *dir_h = "";
  char scat[10];
  char temp[5];

  for(int h = 0; h <5; h + + )
  { temp[h] = '\0';}

  strcpy(cod_ens,opcod);

  ut_aux(0,oper,aux,',');
  strcpy(str,aux);
  if(aux[0] = = '$')
  {
     ut_aux(1,aux,temp,' ');
     strcat(cod_ens,temp);
  }
  else
  {
  if (conv_dec(str,n))
  {
    int j = cuenta_car(str);
    if(j < 2)
    {
      strcpy(scat,"0");
      strcat(scat,str);
      strcat(cod_ens,scat);
    }
  }
  else
  {
          while(dir = = -1 && w < = Lin)
    {
            dir = tabla1[w]-> bus_et(aux);
            w = w + 1;
    }
    if(dir! = -1)
    {
            _strupr( itoa(dir,dir_h,16));
            strcat(cod_ens,dir_h);
    }
    else
    {
      msj_err(7,lin,mensaje, error);
    }
  }
  limp_aux(aux,70);
  }
}
//////////////////////////////////////////////////////////////////////
void tablacomp::direc7(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
  int n = 0;
  char aux[70] = "\0";
  int checa_h = 1;
  int i = 0;
  int dir = -1;
  int w = 0;
  char *dir_h = "";

  char n_h [10];


  strcpy(cod_ens,opcod);
  strcat(cod_ens, "00");

  ut_aux(0,oper,aux,',');
  if (conv_dec(aux,n))
  {
    strcat(cod_ens,aux);
  }
  else
  {
          while(dir = = -1 && w < = Lin)
    {
      dir = tabla1[w]-> bus_et(aux);
              w + = 1;
    }

    if(dir! = -1)
    {
       _strupr( itoa(dir,n_h,16));
      strcat(cod_ens,n_h);
          } //if no checa
    else
    {
      msj_err(3,lin,mensaje, error);
    }
  } //else
  limp_aux(aux,70);
}
//////////////////////////////////////////////////////////////////////
void tablacomp::direc8(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
  int j = 0;
  int n = 0;
  int dir = -1;
  int w = 0;
  int op_cod = 0;
  char n_h [10];
  char *aux_opc = "";
  int caso = 0;
  char aux[70] = "\0";

  limp_aux(aux,70);
```

```
    j=ut_aux(0,oper,aux,',');
    n=atoi(aux);
    if(n==0 && aux!="0" || (n<0 || n>7))  cout<<"Bit
invalido";
    if(n>=0 && n<=7)
    {
      limp_aux(aux,70);
      ut_aux(j+1,oper,aux,' ');
      while(dir==-1 && w<=Lin)
      {
              dir=tabla1[w]->bus_et(aux);
              w++;
      }
      if(dir<=256 && dir!=-1)
      {
              caso=atoi(opcod);
              switch(caso)
              {
                case 1:
                  {
                    op_cod=10+2*n;
                    _strupr(_itoa(op_cod,n_h,16));
                strcpy(cod_ens,n_h);
                 strupr(_itoa(dir,n_h,16));
                strcat(cod_ens, n_h);
              }break;
                case 0:
                  {
                          op_cod=11+2*n;
                          _strupr(_itoa(op_cod,n_h,16));
                          strcpy(cod_ens,n_h);
                          strcat(cod_ens,"00");
                          _strupr(_itoa(dir,n_h,16));
                strcat(cod_ens, n_h);
                        }break;
                } //caso
      }   //if (dir)
      else msj_err(3,lin,mensaje, error);
    }   //if n
    else {msj_err(8,lin,mensaje, error);}
}

////////////////////////////////////////////////////////////////////////
void tablacomp::direc9(int Lin,tablacomp * tabla1[], int lin,
CString& mensaje, int & error)
{
    int j=0;
    int caso=0;
    int n=0;
    int dir1=-1;
    int dir2=-1;
    int w=0;
    int op_cod=0;
    char aux[70]="\0";
    char * aux_opc="";
    char n_h[70];
    int n_bytes=0;
    int s_r=0;

    limp_aux(aux,70);
    j=ut_aux(0,oper,aux,',');
    n=atoi(aux);
    if(n==0 && aux!="0" || (n<0 || n>7)) {cerr<<"Bit
invalido";}
    if(n>=0 && n<=7)
    {
      limp_aux(aux,70);
      j=ut_aux(j+1,oper,aux,',');
      while(dir1==-1 && w<=Lin)
      {
              dir1=tabla1[w]->bus_et(aux);
              w++;
      }
      if(dir1<256 && dir1!=-1)
      {
              limp_aux(aux,70);
              j=ut_aux(j+1,oper,aux,' ');
              w=0;
              while(dir2==-1 && w<=Lin)
      {
              dir2=tabla1[w]->bus_et(aux);
              w++;
      }
      if(dir2!=-1)
      {
        s_r=salto_rel(dir2,pc,n_bytes);
                        if(s_r!=-1  && n_bytes  <=129  &&
n_bytes>=-126 )
                {
                  caso=atoi(opcod);
                  switch(caso)
                  {
                    case 1:
                    {
                            op_cod=2*n;
                            _strupr(_itoa(op_cod,n_h,16));
                            strcpy(cod_ens,"0");
                  strcat(cod_ens,n_h);
                   strupr(_itoa(dir1,n_h,16));
                  strcat(cod_ens, n_h);
                   strupr(_itoa(n_bytes,n_h,16));
                  strcat(cod_ens, n_h);
                          }break;
                          case 0:
                          {
                            op_cod=1+2*n;
                            _strupr(_itoa(op_cod,n_h,16));
                            strcpy(cod_ens,"0");
                  strcat(cod_ens,n_h);
                   strupr(_itoa(dir1,n_h,16));
                  strcat(cod_ens, n_h);
                   strupr(_itoa(n_bytes,n_h,16));
                  if(strlen(n_h)==2) strcat(cod_ens,"0");
                  strcat(cod_ens, n_h);
                          }break;
                  } //caso
                }
                else
                {
                  msj_err(3,lin,mensaje, error);
                }
              }//if dir1 254
              else
                {msj_err(3,lin,mensaje, error);}
            }//if dir2 256
            else
          {msj_err(3,lin,mensaje, error);
          }
    }
    limp_aux(aux,70);
}
////////////////////////////////////////////////////////////////////////
void tablacomp::direc10(int Lin,tablacomp * tabla1[], int
lin, CString& mensaje, int & error)
{
    int dir=-1;
    int w=0;
    int s_r=0;
```

```
    char *n_h="";
    int n_bytes=0;

    strcpy(cod_ens, opcod);
    while(dir==-1 && w<=Lin)
    {
      dir=tabla1[w]->bus_et(oper);
      w++;
    }
    s_r=salto_rel(dir,pc,n_bytes);
    if(s_r!=-1 && n_bytes<=129 && n_bytes>=-126 )
    {
      strupr(_itoa(s_r,n_h,16));
      int j=strlen(n_h);
      if(j<2)
      {
        char c[2]="0";
        // strcat(c,"0");
        strcat(cod_ens,c);
      }
      strcat(cod_ens,n_h);
    }
    else {msj_err(6,lin,mensaje, error);}
}
//////////////////////////////////////////////////////////////////////////
void tablacomp::direc_END(char cad[], int &i)
{
  char aux[70];

  limp_aux(aux,70);
  strcpy(opcod,"");
  i=ut_aux(i,cad,aux,' ');
  strcpy(oper,aux);
  pc=0;
  pcet=-1;
  i=recorre(i,cad);
  if(cad[i]==';')
  {
    i=ut_aux(i,cad,aux,'\0');
    strcpy(coment,aux);
    limp_aux(aux,70);
  }
  else
  {
    strcpy(coment,"\0");
  }

}
//////////////////////////////////////////////////////////////////////////
/
void tablacomp::direc_ORG(char cad[], int &i, tablacomp
*tabla1[], int lin, CString& mensaje, int & error)
{
  char aux[70];
  char temp[20];
  int c;

  limp_aux(aux,70);
  limp_aux(temp,20);

  i=tabla1[lin]->recorre(i,cad);
  if(cad[i]=='\0')
  {
    msj_err(2,lin,mensaje, error);
  }
  else
  {
    c=-1;
    i=ut_aux(i,cad,aux,' ');
    tabla1[lin]->asig_oper(aux);
    if(aux[0]=='$')
    {
      ut_aux(1,aux,temp,' ');
      tabla1[lin]->conv_dec(temp,c);
    }
    else
    {
      int w=0;
        char ax[20];
        strcpy(ax,aux);
        while(c==-1 && w<=lin)  //realmente w<l?
verificar
        {
          c=tabla1[w]->bus_et(ax);
          w++;
        }
          if(c==-1)
          {
            msj_err(5,lin,mensaje, error);
          }
      }
          tabla1[lin]->pc(c);
          tabla1[lin+1]->inc_sig_pc(0,c);

          tabla1[lin]->asig_opcod(aux);
          tabla1[lin]->asig_etiq(aux);
          i=tabla1[lin]->recorre(i,cad);
          if(cad[i]==';')
          {
            i=ut_aux(i,cad,aux,'\0');
            tabla1[lin]->asig_coment(aux);
          }
  } //else
}
//////////////////////////////////////////////////////////////////////////
/
void tablacomp::direc_EQU(char cad[], int &i, tablacomp
*tabla1[], int lin, CString& mensaje, int & error)
{

  char temp[20];
  int c;
  char aux[70];



  i=tabla1[lin]->recorre(i,cad);
  if(cad[i]=='\0')
  {
    i=ut_aux(i,cad,aux,' ');
    tabla1[lin]->asig_oper(aux);
    tabla1[lin]->pc(0);
    tabla1[lin+1]->pc(0);
    ut_aux(1,aux,temp,' ');
    tabla1[lin]->conv_dec(temp,c);
    limp_aux(aux,70);
    tabla1[lin]->asig_pcet(c);
    i=tabla1[lin]->recorre(i,cad);
    if(cad[i]==';')
    {
      i=ut_aux(i,cad,aux,'\0');
      tabla1[lin]->asig_coment(aux);
      limp_aux(aux,70);
    }
    else
    {
      tabla1[lin]->asig_coment("\0");
    }
  }//if
  else
  {
    msj_err(2,lin,mensaje, error);
```

```
}
}//eau
////////////////////////////////////////////////////////////////////////////

void  tablacomp::direc_DW_DB(char  cad[],  int  &i,
tablacomp * tabla1[], int lin, char aux[], CString& mensaje,
int & error)
{
  char temp[5];
  for(int j = 0 ; j < 5; j + + )
  {
    temp[j] = '\0';
  }

  int p = tabla1[lin]->ext_pc();
  if(!strcmp(aux, "DB"))
  {
    tabla1[lin + 1]->inc_sig_pc(1,p);
  }
  if(!strcmp(aux, "DW"))
  {
    tabla1[lin + 1]->inc_sig_pc(2,p);
  }
  limp_aux(aux,70);
  i = tabla1[lin]->recorreli,cad);
  if(cad[i] = '\0')
  {
    i = ut_aux(i,cad,aux,' ');
    tabla1[lin]->asig_oper(aux);
    ut_aux(i ,aux,temp, ' ');
    tabla1[lin]->asig_opcod(temp);
    limp_aux(aux,70);
    tabla1[lin]->asig_pcet(-1);
    i = tabla1[lin]->recorreli,cad);
    if(cad[i] = = ' ')
    {
      i = ut_aux(i,cad,aux,'\0');
      tabla1[lin]->asig_coment(aux);
      limp_aux(aux,70);
    }
    else
    {
      tabla1[lin]->asig_coment("\0");
    }
  }//if
  else
  {
    msj_err(2,lin,mensaje, error);
  }
}//db_dw
////////////////////////////////////////////////////////////////////////////

void tablacomp::vacio(void)
{
  strcpy(pc_ens,"");
  strcpy(cod_ens,"");
  strcpy(etq,"");
  strcpy(inmo,"");
  strcpy(oper,"");
  pcet = -1;
  pc = -1;
  strcpy(opcod,"");
}

////////////////////////////////////////////////////////////////////////////

void msj_err(int No_msj,int Linea, CString & mensaje, int&
error)
{
  char lin[5];
  Linea + = 1;
  _itoa(Linea,lin,10);

  switch(No_msj)
  {
    case 1: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin; mensaje + = " Direccionamiento no
permitido\r\n"; error = 1; } break;
    case 2: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Falta especificar direcci%n\r\n";
error = 1; } break;
    case 3: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Direcci%n no vßlida\r\n"; error = 1;
} break;
    case 4: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Direcci%n no. 2 no vßlida\r\n";
error = 1; } break;
    case 5: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Direcci%n de inicio
desconocida\r\n"; error = 1; } break;
    case 6: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Direcci¾n relativa no vßlida\r\n";
error = 1; } break;
    case 7: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Etiqueta no econtrada\r\n";
error = 1; } break;
    case 8: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Bit no vßlido\r\n"; error = 1; } break;
    case 9: { mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Operando no vßlido\r\n"; error = 1;
} break;
    case 10:{ mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " No se puede abrir el archivo\r\n";
error = 1; } break;
    case 11:{ mensaje + = "Error en lÍnea no. ": mensaje
+ = lin;mensaje + = " Nm%nico no vßlido\r\n"; error = 1;
} break;


                                    /* case 1 :{
cout < < "LÍnea" < < Linea + 1 < < "Error:Direccionamiento
no permitido";} break;
      case 2:{ cout < < "LÍnea" < < Linea + 1 < < "Falta
especificar direcci%n";} break;
    case 3:{ cout < < "LÍnea" < < Linea + 1 < < "Direcci%n
no vßlida";} break;
    case 4:{ cout < < "LÍnea" < < Linea + 1 < < "Direcci%n
no. 2 no vßlida";} break;
    case 5:{ cout < < "LÍnea" < < Linea + 1 < < "Direcci%n
de inicio desconocida";} break;
      case 6:{ cout < < "LÍnea" < < Linea + 1 < < "Dirección
relativa no vßlida";} break;
      case 7:{ cout < < "LÍnea" < < Linea + 1 < < "Etiqueta
no econtrada";} break;
      case 8:{ cout < < "LÍnea" < < Linea + 1 < < "Bit no
vßlido";} break;
      case 9:{ cout < < "LÍnea" < < Linea + 1 < < "Operando
no vßlido";} break;
      case 10:{ cout < < "LÍnea" < < Linea + 1 < < "No se
puede abrir el archivo";} break;
                                    case 11:{
cout < < "LÍnea" < < Linea + 1 < < "Nm%nico no vßlido";}
break;*/
  }
}
```

D - 14

```
//Programa CLASES.H
//Diseñado por: Luz María Castillo Jiménez
```

```
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <alxwin.h>
#include <process.h>
#include <iomanip.h>
//#include <new.h>
//void newError(void);

class tablacomp;                    //Referencia anticipada

//void cargar(FILE &);              //Carga el archivo a ensamblar
retorna el apunte
                                    //dor al archivo
int ut_aux(int i, char cad[], char aux[], char c_fin);
void limp_aux(char aux[], int tam);
void msj_err(int No_msj,int Linea, CString&, int &);

////////////////////////////////////////////////////////////////////////

const NO_INSTRUC = 85;              //Número de instrucciones
almacenadas por default
const L_OP = 20;                    //Número de caracteres
permitidos para el operando
const TAMI = 90;                    //Número de caracteres
permitidos para el nombre
                                    //del archivo y para la cadena
que contiene la
                                    //instrucción
const NO_INST_USR = 180;

////////////////////////////////////////////////////////////////////////
class instruc                       //Almacena las instrucciones por
default creando
                                    //para comparaciones
posteriores
{
  private:
    char nmo[6];          //Nmónico
    char opcod[3];        //Código de operación
    char tipins;          //Tipo de instrucción
  public:
    void init(char Nmo[], char Opcod[], char Tipins);
                                    //Esta función recibe como
parámetros el nmónico,
                                    //el código de operación y el
tipo de instrucción
                                    //para asignarlos a los
miembros privados de la
                                    //clase y así introducir las
instrucciones por
                                    //default
    char verifTip(void);  //Retorna el tipo de instrucción
        int friend verifNmo(tablacomp *Nmo, instruc
*INSTRUCC, int &);
                                    //Realiza las comparaciones
entre los campos nmo
                                    //de los dos parámetro se que
recibe para verificar
                                    //si el primer parámetro es
válido o no.
    char * verifopcod(void);
};

////////////////////////////////////////////////////////////////////////
class tablacomp                     //Tabla de simbolos y
compilación
{
  private:
    int tipdir;           //Tipo de direccionamiento
    char tipins;          //Tipo de instrucción
    int pc;               //Program Counter
    int pcet;             //Dirección de etiqueta
    char pc_ens[10];      //pc en hexadecimal
    char eti[6];          //Etiqueta
    char nmo[6];          //Nmónico
    char opcod[10];       //Código de operación

    char cod_ens[10];     //Código ensamblado
    char operL_OP;        //Operando
    char comen[TAMI];

  public:
    void pci(int);        //Esta función inicializa el PC
        int friend verifnmo(tablacomp *Nmo, instruc
*INSTRUCC, int&);
    void asig_eti(char cad[]);
    void asig_pce(int);
    int verifOp(tablacomp *tabla1, int &sum);   //Recibe
y retorna el número de linea.
                                    //Verifica operandos que
comiencen con "#", ",",
                                    //"$" y de instrucciones
C,O,F,A,E para incremen
                                    //tar el PC
    void verifop2(int I, tablacomp *tabla1[], int &sum,
CString&, int&);  //Verifica los operandos de instrucciones
A y E
                                    //retorna la linea
    int bus_et(char Oper[]);        // a la dirección de una
etiqueta
    int verifIb(char cad[]);
    int verifet(char []);

    int recorre(int i, char[]);
    void inc_sig_pc(int val, int);
    int ver_val(int n, int & sum, int);
      int forma_lin2(int i,int Lin, tablacomp * Tabla1[],
CString&, int&);
    int salto_rel(int ,int, int &);
    int cuenta_car(char C[]);
    int val_asc(char C[], int i);
    int conv_dec(char *Car, int &entero);
    void asig_nmo(char cad[]);
    void asig_opcod(char cad[]);
    void asig_tipins(char);
    void asig_oper(char cad[]);
    void asig_tipdir(int d);
    void direc1(void);
    void direc2(int Lin, tablacomp *tabla1[]);
        void direc3(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
        void direc4(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
    void direc5(void);
        void direc6(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
        void direc7(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
        void direc8(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
        void direc9(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
        void direc10(int Lin, tablacomp *tabla1[], int
lin,CString&, int&);
    void impr(void);
    int ext_pc(void);
```

```
    int ext_pcet(void);
    int ext_tipdir(void);
    void salvar(FILE *arch, int b);
    void salvar_mik(FILE *arch, int b, int &, int &);
    void tabla_simb(FILE *arch);
    void asig_pc_ens(char * pc);
    void ext_nmb(char aux_nmo[]);

    void  direc_END(char cad[], int &i);
          void  direc_ORG(char cad[],  int  &i,  tablacomp
*tabla1[], int lin,CString& ,int&);
          void  direc_EQU(char cad[],  int  &i,  tablacomp
*tabla1[], int lin,CString& ,int&);
          void  direc_DW_DB(char cad[], int &i, tablacomp
*tabla1[], int lin, char aux[],CString&, int&);
    void asig_coment(char cad[]);
    void vacia(void);

};
```

```
//Programa ENS6805.CPP
//Diseñado por: Luz María Castillo Jiménez

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <afxwin.h>
#include "ens6805.h"
#include "resource.h"
#include "clases.h"

const char *MenuResName = "MAINMENU";
const Hctl = 200;
const Vctl = 400;
const MaxEditLen = 70;

// declara una funci%n global para calcular el tama±o y la
ubicaci%n de un control
void makeRect(int nX, int nY, int nW, int nH, CRect &r)
{
  r.top = nY;
  r.left = nX;
  r.bottom = nY + nH;
  r.right = nX + nW;
}

// define una clase de aplicaci%n derivada de CWinApp
class CxEdit: public CEdit
{
public:
   int GetLineLength(int nLineNumber);
};

/////////////////////////////////////////////////////////////////////////////
// CxDialog dialog

CxDialog::CxDialog()
        : CDialog(CxDialog::IDD)
{
        //{{AFX_DATA_INIT(CxDialog)
        m_archivo = "c:\\";
        m_directorio = "c:\\";
        //}}AFX_DATA_INIT
}

void CxDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CxDialog)
        DDX_Text(pDX, ID_ARCHIVO_EDIT, m_archivo);
        DDV_MaxChars(pDX, m_archivo, 12);
        DDX_Text(pDX,        ID_DIRECTORIO_EDIT,
m_directorio);
        DDV_MaxChars(pDX, m_directorio, 30);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CxDialog, CDialog)
        //{{AFX_MSG_MAP(CxDialog)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////

// CxGDialog dialog

CxGDialog::CxGDialog()
        : CDialog(CxGDialog::IDD)
{
        //{{AFX_DATA_INIT(CxGDialog)
        m_arch = "c:\\";
        m_direc = "c:\\";
        //}}AFX_DATA_INIT
}

void CxGDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CxGDialog)
        DDX_Text(pDX, ID_ARCHIVO_EDIT, m_arch);
        DDV_MaxChars(pDX, m_arch, 12);
        DDX_Text(pDX, ID_DIRECTORIO_EDIT, m_direc);
        DDV_MaxChars(pDX, m_direc, 30);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CxGDialog, CDialog)
        //{{AFX_MSG_MAP(CxGDialog)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
class CWindowApp: public CWinApp
{
public:
  virtual BOOL InitInstance();
};

class CAppMDIChild: public CMDIChildWnd
{

public:
  int nChildNum;
  // apuntador al control de cuadro de edici%n
  CxEdit *TextBox;

  /////
  char archv[MaxEditLen + 1];
  ////
  int bandera;


  CAppMDIChild();
  ~CAppMDIChild();
  BOOL Create(LPSTR szTitle, int nChildNum);
protected:
  // maneja el cierre de la ventana hija MDI
  afx_msg void OnClose();
  // maneja la destrucci%n de una ventana hija MDI
  afx_msg void OnDestroy();
  afx_msg int OnCreate(LPCREATESTRUCT lpCS);
  afx_msg void CMGuardar();
  afx_msg void CMAbrir();
```

```
    void CMEnsamblar();
    void CMAbout();
    // declara la macro de mapeo de mensajes
};

class CAppMDIFrame: public CMDIFrameWnd
{

public:
    // indicador para cerrar rßpidamente todas las ventanas
hijas MDI
    BOOL bExpressClose;
    int nLastMDIChild;

    CAppMDIFrame();
    ~CAppMDIFrame() {delete pMenu;};

    // crea una nueva ßrea de ventana cliente
    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
    // crea una ventana hija MDI
    afx_msg void CMCreateChild();
    // pone en cascada a las hijas MDI
    afx_msg void CMCascadeChildren() {MDICascade();}
    // pone en mosaico a las hijas MDI
    afx_msg void CMTileChildren() {MDITile();}
    // acomoda los iconos hijos MDI
    afx_msg void CMArregeIcons() {MDIIconArrange();}
    // cierra todas las hijas MDI
    afx_msg void CMCloseChildren();
    // obtiene el n·mero de hijas MDI
    int GetChildCount() {return nNumMDIChildren;}
    // maneja el comando para la cuenta de hijas MDI
    afx_msg void CMCountChildren();
    // manija el mensaje de destrucci¾n de hija
    afx_msg LONG OnChildDestroy(UINT wParam, LONG
lParam);
    // maneja la salida de la aplicaci¾n
    afx_msg void OnExit()
    {   SendMessage(WM_CLOSE); }

protected:

    CMenu *pMenu;
    int nNumMDIChildren;
    // maneja el cierre de una ventana marco MDI
    virtual void OnClose();

    // declara el macro de mapeo de mensaje
    DECLARE_MESSAGE_MAP();
};


int CxEdit::GetLineLength(int nLineNumber)
{
    int nStartPos = -1;

    if(nLineNumber > -1)
       nStartPos = LineIndex(nLineNumber);
       return    (WORD)   SendMessage(EM_LINELENGTH,
nStartPos);
}


CAppMDIChild::CAppMDIChild()
{
    TextBox = NULL;
    bandera = 0;
}

CAppMDIChild::~CAppMDIChild()
{
    delete TextBox;
}

//////////////////////////////////////////////////////////////////////

BOOL    CAppMDIChild::Create(LPSTR    szTitle,    int
nChildNumber)
{
    nChildNum = nChildNumber;
                                  //  D  W  O  R  D
CMDIStyle = /*WS_VSCROLL|WS_HSCROLL*/ES_AUTO
HSCROLL|ES_AUTOVSCROLL;
       return      CMDIChildWnd::Create(NULL,
szTitle/*,CMDIStyle*/);
}

CAppMDIChild::OnCreate(LPCREATESTRUCT lpCS)
{
    CString s;
    CRect r;
                                  //  D  W  O  R  D
dwStaticStyle = WS_CHILD|WS_VISIBLE|ES_LEFT;
                                  D     W  O  R  D
dwEditStyle = WS_CHILD|WS_VISIBLE|ES_LEFT|ES_MU
LTILINE|ES_UPPERCASE|
            ES_AUTOHSCROLL|ES_AUTOVSCROLL
            |WS_VSCROLL|WS_HSCROLL;


  // obtiene el tama±o  la ubicaci¾n del control a crear
  makeRect(10, 10,Wctl,Hctl,r);

    // crea el cuadro de edici¾n
    TextBox = new CxEdit();
    TextBox->Create(dwEditStyle,r,this,ID_TEXT_EDIT);

  return 0;
}

void CAppMDIChild::OnClose()
{
    CAppMDIFrame    *pParent = (CAppMDIFrame*)
(GetParentFrame());
    // regresa TRUE segn estado del miembro
bExpressClose
  // ¬Est§ puesto el indicador express-close de la ventana
madre
  if (pParent->bExpressClose == TRUE)
    return;
  else
    // pregunta al usustio y regresa el resultado de la
pregunta
       if (MessageBox("¬Desea cerrar esta ventana
MDI?","Pregunta",MB_YESNO|MB_ICONQUESTION) ==
IDYES)
    {  MDIDestroy(); }
}

void CAppMDIChild::OnDestroy()
{
                  C  A  p  p  M  D  I  F  r  a  m  e
*pParent = (CAppMDIFrame*)GetParentFrame();

pParent->SendMessage(WM_CHILDDESTROY,(UINT)m_
hWnd,0);
}

CAppMDIFrame::CAppMDIFrame()
```

```
{
    Create(NULL, "Editor y Ensamblador de los MCU's 6870SP3, U3 y R3" WS_OVERLAPPEDWINDOW|
        WS_MAXIMIZE,rectDefault, NULL, MenuRezName);
  // borra el indicador express-close
  bExpressClose = FALSE;
  // inicializa la cantidad de hijas MDI
  nLastMDIChild = 0;
  nNumMDIChildren = 0;
}

int CAppMDIFrame::OnCreate(LPCREATESTRUCT lpCS)
{
  pMenu = new CMenu();
  pMenu->LoadMenu(MenuRezName);
  CreateClient(lpCS,pMenu->GetSubMenu(0));
  return 0;
}

void CAppMDIFrame::CMCreateChild()
{
  char s[81];
  CRect r;
  CAppMDIChild *pChild = new CAppMDIChild();

makeRect(10*nNumMDIChildren,10*nNumMDIChildren,
Wctl,Hctl,r);
  nLastMDIChild++;
  sprintf(s, "Child # %d", nLastMDIChild);
  if(!pChild->Create(s, nLastMDIChild))
    {
      delete pChild;
      nLastMDIChild--; // decrementa el Ýndice de la hija
mayor
      return;
    }
  nNumMDIChildren++;
  // muestra la nueva ventana hija MDI
  pChild->ShowWindow(SW_SHOW);
}

void CAppMDIChild::CMGuardar()
{
    int nLineSize;
    int nLineCount;
    char s[TAM1];
    char cadena[TAM1];
    char archivo[MaxEditLen + 1];
    FILE *p;
    int tam;
    int cmp1;
    CxDialog dlg;
    dlg.DoModal();

    strcpy(archivo,dlg.m_direc);
    tam = strlen(archivo);

    strcat(archivo,dlg.m_arch);

    nLineCount = TextBox->GetLineCount();

    cmp1 = strcmp(archivo,"c:\\*.*");

    if(!bandera && cmp1)
    {
    if((p = fopen(archivo,"w")) == NULL)
    {
                MessageBox("No se pudo guardar el
archivo","Informaci%n",MB_OK);
      bandera = 1;
    }
    else
    {
      for(int i = 0; i<nLineCount;i++)
      {
        nLineSize = TextBox->GetLineLength(i);
        TextBox->GetLine(i,s,nLineSize);
        strncat(cadena,s,nLineSize);
        strcat(cadena,"\n");
        fputs(cadena,p);
        strcpy(cadena,"\0");
      }
    }
    fclose(p);
}

void CAppMDIChild::CMAbrir()
{
  char s[TAM1];
  CString cadena;
  char archivo[MaxEditLen + 1];
  char aux[TAM1];
  FILE *p;
  int tam;
  int cmp1;

  CxDialog dlg;
  dlg.DoModal();

  strcpy(aux,"\0");
  cadena = "\0";
  strcpy(archivo,dlg.m_directorio);
  tam = strlen(archivo);
  if(archivo[tam-1] != '\\' && dlg.m_directorio != "c:\\")
    strcat(archivo,"\\");
  strcat(archivo,dlg.m_archivo);
  cmp1 = strcmp(archivo,"c:\\*.*");

  if(!bandera && cmp1)
  {
  if((p = fopen(archivo,"r")) == NULL)
  {
                MessageBox("No se pudo abrir el
archivo","Informaci%n",MB_OK);
    bandera = 1;
  }
  else
  {
    while ((fgets(s,TAM1,p)) != NULL && !feof(p))
          //for(int i = 0; i<nLineCount;i++)
    {
      strncat(aux,s,strlen(s)-1);
      cadena += aux;
      cadena += "\r\n";
      strcpy(aux,"\0");
    }
    TextBox->SetWindowText((const char*)cadena);
  fclose(p);
  bandera = 0;
  strcpy(archv,(const char *) archivo);
  }
  }
}

////////////////////////////////////////////////////////////////
void CAppMDIChild::CMEnsamblar()
{
```

```
    instruc *tabla[NO_INSTRUC];   //Crea un arreglo de
apuntadores a las instrucciones
                                  //por default
  tablacomp *tabla1[NO_INST_USR];     //Crea un arreglo
de apuntadores a las instrucciones
                                  //ensambladas en la primera
pasada
  char aux[70] = "\0";
    char cad[TAM];                //Cadena que contiene la
instrucción
  int *flag = new int[NO_INST_USR];
  CString mensaje;

  char archp[TAM];
  char archm[TAM];
  int lin = 0;
  mensaje = "\0";
  int error = 0;

  for (int i = 0; i<NO_INSTRUC; i + +)
  {
    tabla[i] = new instruc;
  }
  for (i = 0; i<NO_INST_USR; i + +)
  {
        tabla1[i] = new tablacomp;
    flag[i] = 0;
  }

    tabla[0]->init("ADC","A9","E");
  tabla[1]->init("ADD","AB","E");
  tabla[2]->init("AND","A4","E");
  tabla[3]->init("ASL","48","A");
  tabla[5]->init("ASRA","47","A");
  tabla[6]->init("ASRX","57","A");
  tabla[7]->init("BCC","24","F");
  tabla[8]->init("BCLR","2 ","C");
  tabla[9]->init("BCS","25","F");
  tabla[10]->init("BEQ","27","F");
  tabla[11]->init("BHCC","28","F");
  tabla[12]->init("BHCS","29","F");
  tabla[13]->init("BHI","22","F");
  tabla[14]->init("BLS","23","F");
  tabla[15]->init("BIH","2F","F");
  tabla[16]->init("BIL","2E","F");
  tabla[17]->init("BIT","A5","E");
  tabla[18]->init("BLO","25","F");
  tabla[19]->init("BLS","23","F");
  tabla[20]->init("BMC","2C","F");
  tabla[21]->init("BMI","2B","F");
  tabla[22]->init("BMS","2D","F");
  tabla[23]->init("BNE","26","F");
  tabla[24]->init("BPL","2A","F");
  tabla[25]->init("BRA","20","F");
  tabla[26]->init("BRN","21","F");
  tabla[27]->init("BRCLR","","D");

  tabla[28]->init("BRSET","","D");
  tabla[29]->init("BSET","","C");
  tabla[30]->init("BSR","AD","E");
  tabla[31]->init("CLC","98","B");
  tabla[32]->init("CLI","9A","B");
  tabla[33]->init("CLR","4F","A");
  tabla[34]->init("CLRA","4F","A");
  tabla[35]->init("CLRX","5F","A");
  tabla[36]->init("CMP","A1","E");
  tabla[37]->init("COM","43","A");
  tabla[38]->init("COMA","43","A");
  tabla[39]->init("COMX","53","A");
  tabla[40]->init("CPX","A3","E");
  tabla[41]->init("DEC","4A","A");
  tabla[42]->init("DECA","4A","A");
  tabla[43]->init("DECX","5A","A");
  tabla[44]->init("EOR","A8","E");
  tabla[45]->init("INC","4C","A");
  tabla[46]->init("INCA","4C","A");
  tabla[47]->init("INCX","5C","A");
    tabla[48]->init("JMP","AC","E"); //VALOR INICIAL
ARBITRARIO
    tabla[49]->init("JSR","AD","E");  //VALOR INICIAL
ARBITRARIO
  tabla[50]->init("LDA","A6","E");
  tabla[51]->init("LDX","AE","E");
  tabla[52]->init("LSL","48","A");
  tabla[53]->init("LSLA","48","A");
  tabla[54]->init("LSLX","58","A");
  tabla[55]->init("LSR","44","A");
  tabla[56]->init("LSRA","44","A");
  tabla[57]->init("LSRX","54","A");
  tabla[58]->init("NEG","40","A");
  tabla[59]->init("NEGA","40","A");
  tabla[60]->init("NEGX","50","A");
  tabla[61]->init("NOP","9D","B");
  tabla[62]->init("ORA","AA","E");
  tabla[63]->init("ROL","49","A");
  tabla[64]->init("ROLA","49","A");
  tabla[65]->init("ROLX","59","A");
  tabla[66]->init("ROR","46","A");
  tabla[67]->init("RORA","46","A");
  tabla[68]->init("RORX","56","A");
  tabla[69]->init("RSP","9C","B");
  tabla[70]->init("RTI","80","B");
  tabla[71]->init("RTS","81","B");
  tabla[72]->init("SBC","A2","E");
  tabla[73]->init("SEC","99","B");
  tabla[74]->init("SEI","9B","B");
    tabla[75]->init("STA","A7","E");  //VALOR INICIAL
ARBITRARIO
    tabla[76]->init("STOP","8E","B");
    tabla[77]->init("STX","AF","E");  //VALOR INICIAL
ARBITRARIO
  tabla[78]->init("SUB","A0","E");
  tabla[79]->init("SWI","83","B");
  tabla[80]->init("TAX","97","B");
  tabla[81]->init("TSTA","4D","A");
  tabla[82]->init("TSTX","5D","A");
  tabla[83]->init("TXA","9F","B");
  tabla[84]->init("WAIT","8F","B");
                                  //Cargo el archivo deseado
                                  //Tomo líneas por líneas y
asigno PC, etiqueta.

                                  //instrucción (in TABLA
INSTRUCCIONES)
////////////////////////////////////////////////////////////////////////

  lin = -1;
  int pc_im = 0;
  int v_direc = 0;

  int cod = 0;
  int in = 0;
  int sum = 0;
  int n = 0;
  char code[10];
  char temp[5];
  FILE *ip;

  for(int w = 0; w<5; w + +)
  {
    temp[w] = '\0';
```

```
}
strcpy(archp, archv);
if ((fp = fopen(archp,"r")) == NULL)
{
   msj_err(10,-1,mensaje, error);
   exit(1);
}

else
 while(fgets(cad,TAMI,fp) != NULL)
 {
    lin++;
    v_direc = 0;
    in = 0;
    n = strlen(cad);
    cad[n-1] = '\0';
    n = 0;
    if((tabla1[lin]->verifib(cad)))
    {
       lin--;
       in = 1;
    }

    i = 0;
    i = tabla1[lin]->recorre(i,cad);
    if(cad[i] == ';')
    {
       tabla1[lin]->asig_coment(cad);
       int g_pc = tabla1[lin]->ext_pc();
       tabla1[lin]->vacio();
       tabla1[lin + 1]->pc(g_pc);
    }
    if(cad[i] != ';' && cad[i] != '\0')
    {
       tabla1[lin]->asig_coment("\0");
       if(!(tabla1[lin]->verifet(cad)))
       {
          tabla1[lin]->asig_etiq("\0");
          tabla1[lin]->asig_pcet(-1);
          i = tabla1[lin]->recorre(i,cad);
       }
       else
       {
          limp_aux(aux,70);
                i = ut_aux(i,cad,aux,' ');
                tabla1[lin]->asig_etiq(aux);          //asigna
etiqueta
                pc_m = tabla1[lin]->ext_pc();
                tabla1[lin]->asig_pcet(pc_m);
                limp_aux(aux,70);
                i = tabla1[lin]->recorre(i,cad);
       } strcpy(temp,"\0");

       strcpy(codc,"");
       limp_aux(aux,70);
       while(cad[i] != '\0')
       {
          limp_aux(aux,70);
                i = ut_aux(i,cad,aux,' ');
                tabla1[lin]->asig_nmo(aux);          //asigna
el nmónico

                if(!(strcmp(aux,"END")))
                {
                   tabla1[lin]->direc_END(cad,i);
                   v_direc = 1;
                }
          if(!(strcmp(aux,"ORG")))
                {
                   tabla1[lin]->direc_ORG(cad, i, tabla1, lin,
mensaje, error);
                   v_direc = 1;
                }
                if(!(strcmp(aux,"EQU")))
                {
                   tabla1[lin]->direc_EQU(cad, i, tabla1, lin,
mensaje, error);
                   flag[lin] = 1;
                   v_direc = 1;
                }
          if(!(strcmp(aux,"DW")) || !(strcmp(aux,"DB")))
                {
                   tabla1[lin]->direc_DW_DB(cad, i, tabla1,
lin, aux, mensaje, error);
                   v_direc = 1;
                }

          if(!v_direc)
          {
             limp_aux(aux,70);
             n = 0;

             cod = 0;                                  //verifica si es un
nmónico permitido
                /* while(cod == 0 && n<NO_INSTRUC)
                 {
                   cod = verifnmo(tabla1[lin],tabla[n]);
                   n++;
                 } */

                int inic_array = 1;
                int fin_array = 85;
                int comp;
                int pos_act = (inic_array + fin_array)/2;
                cod = verifnmo(tabla1[lin],tabla[pos_act-1],
comp);
                int pos_ant = 0;
                int coinc = 0;
                if(cod == 0)
                {
                while(cod == 0 && coinc == 0)
                {
                   pos_ant = pos_act;
                   if(comp < 0)
                   {
                      fin_array = pos_act;
                      pos_act = (inic_array + fin_array)/2;
                   }

                   if(comp > 0)
                   {
                      inic_array = pos_act;
                      pos_act = (inic_array + fin_array)/2;
                   }

                   if(comp == 0)
                   {
                      n = pos_ant;  //código encontrado
                   }
                   if(pos_ant != pos_act)
                   {

cod = verifnmo(tabla1[lin],tabla[pos_act-1],comp);
                      n = pos_act;
                   }
                   else
                      coinc = 1;
                }
                }
```

```
            if (cod = = 0)
            {
              ms|_err(11,lin,mensaje, error);

              while(cad[i]! = '\0')
              {
                i = tabla1[lin]- > recorre(i,cad);
              }
              int p = tabla1[lin]- > ext_pc();
              tabla1[lin + 1]- > inc_sig_pc(1,p);

              tabla1[lin]- > asig_opcod(aux);
              tabla1[lin]- > asig_tipins('\0');
              tabla1[lin]- > asig_tipdir(0);
              tabla1[lin]- > asig_oper(aux);

      }// if cod = 0

      if(codi = 0) //aquí se asigna el código de operación
            {
              strcpy(codc,tabla1[n-1]- > verifopcod());
              int c;
              tabla1[lin]- > conv_dec(codc,c);
              char tip_i;
              tip_i = tabla1[n-1]- > verifitipi();
              tabla1[lin]- > asig_tipins(tip_i);
              i = tabla1[lin]- > recorre(i,cad);
              if(cad[i] = = ':')
                {
            i = ut_aux(i,cad,aux,'\0');
            tabla1[lin]- > asig_coment(aux);

          }


                        tabla1[lin]- > asig_oper("\0");  //Si el
campo está vacío
                  pc_m = tabla1[lin]- > ext_pc();
                  tabla1[lin + 1]- > inc_sig_pc(1,pc_m);
                  tabla1[lin]- > asig_tipdir(1);
                                        //es una instrucción de
1 byte
                }                       //direccionamiento
inherente.
                if(cad[i]! = '\0')
                {
                        i = ut_aux(i,cad,aux,' ');
                        tabla1[lin]- > asig_oper(aux);
                        limp_aux(aux,70);
            int b = 0;
                                                b =
tabla1[lin]- > verifop(tabla1[lin + 1],sum);  //al verificar el
operando se ve
                                        //cuantos bytes hay
que sumarle al PC
                        if(!b)
                        {
                                tabla1[lin]- > verifop2(lin,
tabla1,sum,mensaje, error);
                        }
                        if(sum! = 0)
                        {
                  tabla1[lin + 1]- > inc_sig_pc(sum,0);
                        }
                i = tabla1[lin]- > recorre(i,cad);
                        if(cad[i] = = ':')
                        {
                          limp_aux(aux,70);
                        i = ut_aux(i,cad,aux,'\0' );
                        tabla1[lin]- > asig_coment(aux);
                        limp_aux(aux,70);
                        }
              }//if l = 0
              int td = 0;
              td = tabla1[lin]- > ext_tipdir();
              switch(tip_i)
              {
                case 'A':
                {
              switch(td)
                {
                case 3:
                                { c = c-16;}break;
                          case 5:
                                { c = c + 48;}break;
                          case 6:
                                { c = c + 32;}break;
                        }
                      }break;
                case 'E':
                {
                        switch(td)
                        {
                          case 3:
                                { c + = 16;}break;
                          case 4:
                                { c + = 32;}break;
                          case 5:
                  { c + = 80;}break;
                          case 6:
                                { c + = 64;}break;
                          case 7:
                        { c + = 48;}break;
                }
              }break;
                }//switch
            char n_h [10];
                  situpr( itoa(c,n_h,16));
                  strcpy(codc,n_h);
                tabla1[lin]- > asig_opcod(codc);
          } //cod
}//while nuevo
// para EQU
} //Si no es comentario
else
{
  if(cad[i] = = ':')
  {}
  else
  {
    if(!lin)
      lin--;
  }
}

limp_aux(cad,TAM);

} //gets
fclose(p);
    i = 0;
    int pcm = 0;
    if(error)
    {
      MessageBox(mensaje,"ERROR",MB_OK);
    }
    else
    {
```

```
            if (cod = = 0)
            {
              msj_err(11,lin,mensaje, error);
              while(cad[i]! = '\0')
              {
                i = tabla1[lin]- > recorre(i,cad);
              }
              int p = tabla1[lin]- > ext_pc();
              tabla1[lin + 1]- > inc_sig_pc(1,p);
              tabla1[lin]- > asig_opcod(aux);
              tabla1[lin]- > asig_tipins('\0');
              tabla1[lin]- > asig_tipdir(0);
              tabla1[lin]- > asig_oper(aux);
        }// if cod = 0
        if(cod! = 0) //aquí se asigna el código de operación
            {
              strcpy(codc,tabla1[n-1]- > verifopcod());
              int c;
              tabla1[lin]- > conv_dec(codc,c);
              char tip_i;
              tip_i = tabla1[n-1]- > veritipi();
              tabla1[lin]- > asig_tipins(tip_i);
              i = tabla1[lin]- > recorre(i,cad);
              if(cad[i] = = ';')
                {
          i = ut_aux(i,cad,aux,'\0');
          tabla1[lin]- > asig_coment(aux);
            }
                        tabla1[lin]- > asig_oper("\0");  //Si el
campo está vacío
                    pc_m = tabla1[lin]- > ext_pc();
                    tabla1[lin + 1]- > inc_sig_pc(1,pc_m);
                    tabla1[lin]- > asig_tipdir(11);
                                        //es una instrucción de
1 byte
                  }                                  //direccionamiento
inherente.
                  if(cad[i]! = '\0')
                  {
                        i = ut_aux(i,cad,aux,' ');
                        tabla1[lin]- > asig_oper(aux);
                        limp_aux(aux,70);
              int b = 0;                                    b =
tabla1[lin]- > verifop(tabla1[lin + 1],sum);  //al verificar el
operando se ve
                                        //cuantos bytes hay
que sumarle al PC
                        if(!b)
                        {
                                    tabla1[lin]- > verifop2(lin,
tabla1,sum,mensaje, error);
                        }
                        if(sum! = 0)
                        {
                tabla1[lin + 1]- > inc_sig_pc(sum,0);
                        }
              i = tabla1[lin]- > recorre(i,cad);
                        if(cad[i] = = ';')
                        {
                          limp_aux(aux,70);
                          i = ut_aux(i,cad,aux,'\0' );
                          tabla1[lin]- > asig_coment(aux);
                          limp_aux(aux,70);
                        }
                }//if l = 0
                int td = 0;
                td = tabla1[lin]- > ext_tipdir();
                switch(tip_i)
                {
                  case 'A':
                    {
              switch(td)
                    {
                      case 3:
                                  { c = c-16;}break;
                              case 5:
                                  { c = c + 48;}break;
                              case 6:
                                  { c = c + 32;}break;
                            }
                          }break;
                      case 'E':
                      {
                              switch(td)
                              {
                                case 3:
                                  { c + = 16;}break;
                                case 4:
                                  { c + = 32;}break;
                                case 5:
                               { c + = 80;}break;
                                case 6:
                                  { c + = 64;}break;
                                case 7:
                               { c + = 48;}break;
                      }
                    }break;
                        }//switch
                char n_h [10];
                    strupr( itoa(c,n_h,16));
                    strcpy(codc,n_h);
                tabla1[lin]- > asig_opcod(codc);
            } //cod
        }//while nuevo
        }// para EQU
        ;//Si no es comentario
        else
        {
          if(cad[i] = = ';')
          {}
          else
          {
            if(!lin)
              lin--;
          }
        }
        limp_aux(cad,TAM);
        } //gets
        fclose(fp);
            i = 0;
            int pcm = 0;
            if(error)
            {
              MessageBox(mensaje,"ERROR",MB_OK);
            }
            else
            {
```

```
                                                    MessageBox("Ensamblaje
exitoso",*Informaci%n",MB_OK);
    }

    //* * * *hasta aqui termina la primera pasada
    for (i = 0; i<NO_INSTRUC; i + +)
    {
            delete tabla1[i];
    }
    i = 0;
    while(i< = lin)
    {
       char p_h[10];
            char scat[10];
            pcm = tabla1[i]->ext_pc();

            if(pcm = = -1)
            {
             strcpy(scat,"\0");
            }
            else
            {
              itmp_aux(p_h,10);
              itmp_aux(scat,10);
       _strupr(_itoa(pcm,p_h,16));
              int j = strlen(p_h);
              if(j < 4)
              {
           strcpy(scat,"0");
                 i + +;
                 for(int w = 0; w < 4-j; w + +)
                    strcat(scat,"0");
        }
              strcat(scat,p_h);
              tabla1[i]->elig_pc_ens(scat);
               tabla1[i]->toma_lin2(i,lin,tabla1, mensaje,
error);
            }
            i + +;
    }

    FILE *prn;
    FILE *mik;

    strcpy(archm, archp);
    strcat(archp, ".prn\0");
    prn = fopen(archp,"w + ");
    strcat(archm,".mik\0");
    mik = fopen(archm,"w + ");
    int total = 0;
    int pc_lin = 0;

    for(i = 0;i< = lin; i + +)
    {
      if(flag[i] = = 0)
        {
                tabla1[i]->salvar(prn,1);
                tabla1[i]->salvar_mik(mik,1,pc_lin, total);
            }
            else
            {
              tabla1[i]->salvar(prn,0);
            }

    }
    fclose(prn);
    fclose(mik);
    prn = fopen(archp,"a");
    fprintf(prn,"\n \n \n \n \n");
    fprintf(prn, "%s", "---Tabla de SYmbolos---");
    fprintf(prn, "\n \n");
    for(i = 0; i< = lin;i + +)
    {
      int p;
      p = tabla1[i]->ext_pcet();
      if(pl = -1)
      {
        tabla1[i]->tabla_simb(prn);
      }
    }
    fclose(prn);


    for (i = 0; i<NO_INST_USR; i + +)
    {
        delete tabla1[i];
    }

}

//////////////////////////////////////////////////////////////////////

void CAppMDIFrame::CMCloseChildren()
{
                                        C A p p M D I C h i l d
*pChild = (CAppMDIChild*)MDIGetActive();

  // pone el indicador bExpressClose
  bExpressClose = TRUE;
  while (pChild)
  {
    // cierra la hija MDI activa
    pChild->MDIDestroy();
    MDINext; // obtiene la siguiente hija MDI
    pChild = (CAppMDIChild*)MDIGetActive();
  }
  // borra el indicador bExpressClose
  bExpressClose = FALSE;
}


// despliega un cuadro de mensaje que muestra la cantidad
de hijas
void CAppMDIFrame::CMCountChildren()
{
  char msgStr[81];

        sprintf(msgStr, "Existen %d ventanas MDI",
nNumMDIChildren);
                        MessageBox(msgStr,      "Informaci%n",
MB_OK|MB_ICONINFORMATION);
}

LONG CAppMDIFrame::OnChildDestroy(UINT wParam,
LONG lParam)
{
  nNumMDIChildren--;
  // vuelve a poner a nLastMDIChild en 0 si es que no hay
hijas MDI
nLastMDIChild = (nNumMDIChildren = = 0)?0:nLastMDIC
hild;
  return 0;
}

void CAppMDIFrame::OnClose()
{
  if (MessageBox("¿Desea finalizar la aplicaci%n? \r \nNo
o l v i d e      g u a r d a r      s u s
```

```
archivos* "Pregunta",MB_YESNO|MB_ICONQUESTION)
  == IDYES)

      DestroyWindow();

}

void CAppMDIChild::CMAbout()
{
   CAbout aboutDlg;
   aboutDlg.DoModal();
}

IMPLEMENT_DYNCREATE(CAppMDIFrame,CMDIFram e
Wnd)

BEGIN_MESSAGE_MAP(CAppMDIChild,CMDIChildWnd)
   ON_COMMAND(CM_GUARDAR,CMGuardar)
   ON_COMMAND(CM_ABRIR,CMAbrir)
   ON_COMMAND(CM_ENSAMBLAR,CMEnsamblar)
   ON_COMMAND(CM_ABOUT,CMAbout)
   ON_WM_CREATE()
   ON_WM_CLOSE()
   ON_WM_DESTROY()
END_MESSAGE_MAP()

BEGIN_MESSAGE_MAP(CAppMDIFrame,CMDIFrameWnd)
   ON_WM_CREATE()
   ON_COMMAND(CM_CREATECHILD,CMCreateChild)

ON_COMMAND(CM_CASCADECHILDREN,CMCascadeC
hildren)
   ON_COMMAND(CM_TILECHILDREN,CMTileChildren)
   ON_COMMAND(CM_ARRANGEICONS,CMArrageIcons)

ON_COMMAND(CM_CLOSECHILDREN,CMCloseChildren)

ON_COMMAND(CM_COUNTCHILDREN,CMCountChildren)
   ON_MESSAGE(WM_CHILDDESTROY,OnChildDestroy)
   ON_COMMAND(CM_EXIT,OnExit)
   ON_WM_CLOSE()

END_MESSAGE_MAP()

// construye el dato miembro m_pMaindWnd de
CWindowApp
BOOL CWindowApp::InitInstance()
{
   m_pMainWnd = new CAppMDIFrame();
   m_pMainWnd->ShowWindow(m_nCmdShow);
   m_pMainWnd->UpdateWindow();
   return TRUE;
}

// el constructor de la aplicaci%n inicializa y ejecuta la
aplicaci%n
CWindowApp WindowApp;

/////////////////////////////////////////////////////////////////////////////
// CAbout dialog

CAbout::CAbout(CWnd* pParent /* = NULL*/)
        : CDialog(CAbout::IDD, pParent)
{
        //{{AFX_DATA_INIT(CAbout)
                // NOTE: the ClassWizard will add
member initialization here
        //}}AFX_DATA_INIT
}

void CAbout::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAbout)
                // NOTE: the ClassWizard will add DDX
and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAbout, CDialog)
        //{{AFX_MSG_MAP(CAbout)
                // NOTE: the ClassWizard will add
message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CxDialog dialog

class CxDialog : public CDialog
{
// Construction
public:
        CxDialog();     // standard constructor

// Dialog Data
        //{{AFX_DATA(CxDialog)
        enum { IDD = ID_ABRIR };
        CString m_archivo;
        CString m_directorio;
        //}}AFX_DATA
// Implementation
protected:
        virtual void DoDataExchange(CDataExchange*
pDX);   // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CxDialog)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
// CxGDialog dialog

class CxGDialog : public CDialog
{
// Construction
public:
        CxGDialog();    // standard constructor

// Dialog Data
        //{{AFX_DATA(CxGDialog)
        enum { IDD = ID_GUARDAR };
        CString m_arch;
        CString m_direc;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange*
pDX);   // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CxGDialog)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
// CAbout dialog

class CAbout : public CDialog
{
```

```
// Construction
public:
        CAbout(CWnd* pParent = NULL);           /  /
standard constructor

// Dialog Data
        //{{AFX_DATA(CAbout)
        enum { IDD = IDD_ABOUTBOX };
                // NOTE: the ClassWizard will add data
members here
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange*
pDX);   // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CAbout)
                //  NOTE: the ClassWizard will add
member functions here
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Código del simulador de los MCU 68705P3, R3 y U3 Versión 1.0

```
// Programa: convert.cpp
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 24 noviembre 1996
// Contiene: Programa que contiene la funciones de
conversi%n de hexadecimal y binario a entero,
// y de las hexadecimal a binario y viceversa

#include <math.h>
#include <string.h>
#include <afxwin.h>

int ConvierteHexadecimalaEntero(const char *auxmem)
{
  int tam;
  int entero = 0;
  char aux[9];
  int pot;

  strcpy(aux,auxmem);
  tam = strlen(aux);

  pot = int(pow(double(16),double(tam-1)));

  for(int i = 0;i < tam;i + + )
  {
    if (aux[i] > = 65)
      entero + = (aux[i]-55) * pot;
    else
      entero + = (aux[i]-48) * pot;
    pot = pot/16;
  }
  return entero;
}

CString ConvierteBinarioaHexadecimal(const char
*auxmem)
{
  int tam;
  int entero = 0;
  char aux[9];
  int pot;
  char hexadecimal[3];
  CString auxhex = "\0";

  strcpy(aux,auxmem);
  tam = strlen(aux);
  pot = int(pow(double(2),double(tam-1)));
  for(int i = 0; i < tam;i + + )
  {
    entero + = (aux[i]-48) * pot;
    pot = pot/2;
  }
  _itoa(entero,hexadecimal,16);


  auxhex + = hexadecimal;
  strcpy(aux,(const char *)auxhex);
  strupr(aux);
  auxhex = aux;
  return auxhex;
}

CString ConvierteHexadecimalaBinario(const char
*auxmem)
{
  int tam;
  int tambin = 8;
  int entero = 0;
  char aux[9];
  int pot;
  char binario[9];

  CString auxbin = "\0";

  tam = strlen(aux);

  pot = int(pow(double(16),double(tam-1)));

  for(int i = 0;i < tam;i + + )
  {
    if (aux[i] > = 65)
      entero + = (aux[i]-55) * pot;
    else
      entero + = (aux[i]-48) * pot;
    pot = pot/16;
  }
  _itoa(entero,binario,2);

  tambin = 8-strlen(binario);
  for (i = 0 ; i < tambin ;i + + )
  {
    auxbin + = "0";
  }
  auxbin + = binario;
  return auxbin;
}

int ConvierteBinarioaEntero(const char *auxmem)
{
  int tam;
  int entero = 0;
  char aux[9];
  int pot;

  strcpy(aux,auxmem);
  tam = strlen(aux);
  pot = int(pow(double(2),double(tam-1)));
  for(int i = 0; i < tam;i + + )
  {
    entero + = (aux[i]-48) * pot;
    pot = pot/2;
  }
  return entero;
}
```

```
// Programa: convert.h
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 24 noviembre 1996
// Contiene: Programa que contiene las cabeceras de las
funciones de conversi%n de hexadecimal
// y binario a entero, y de las hexadecimal a binario y
viceversa

int ConvierteHexadecimalaEntero(const char *auxmem);
CString ConvierteBinarioaHexadecimal(const char
*auxmem);
CString ConvierteHexadecimalaBinario(const char
*auxmem);
int ConvierteBinarioaEntero(const char *auxmem);
```

```
// Programa: funcion.cpp
// Elaborado por: Graciela Santillan Alcala
// Fecha de elaboracion:9 septiembre 1996
// Fecha de ultima correccion: 14 diciembre 1996
// Contiene: Programa que contiene la funciones miembro
de los nemonicos a ser ejecutados
// por el simulador

#include "funcion.h"
```

```
#include "xdialog.h"
#include "hexa.h"

void CxDialog::CalculaSegundos(void)
{
 double val;
//  double fval = 0.00000025;
  char seg[12];

  val = double(m_ciclos);
  segundos += (val * fval);
  sprintf(seg," %2.8f",segundos);
  m_segundos = seg;
}

void CxDialog::IntroduceaMemoria(CString dir, CString
hex, CString bin)
{
  if(dir >= "0000" && dir <= "0009")
  {
    switch(dir[3])
    {
        case '0':{
                  m_PORTAh = hex;
                  m_PORTAb = bin;
                  break;
                 }
        case '1':{
                  m_PORTBh = hex;
                  m_PORTBb = bin;
                  break;
                 }
        case '2':{
                  if(m_procesador == "P3")
                  {
                    if (hex >= "F0")
                    {
                      m_PORTCh = hex;
                      m_PORTCb = bin;
                    }
                    else
                      MessageBox("Error en la asignaci¾n
al Puerto C","Error",MB_OK);
                  }
                  else
                  {
                    m_PORTCh = hex;
                    m_PORTCb = bin;
                  }
                  break;
                 }
        case '3':{
                  if(m_procesador != "P3")
                  {
                    m_PORTDh = hex;
                    m_PORTDb = bin;
                  }
                  break;
                 }
        case '4':{
                  m_DDRAh = hex;
                  m_DDRAb = bin;
                  break;
                 }
        case '5':{
                  m_DDRBh = hex;
                  m_DDRBb = bin;
                  break;
                 }
        case '6':{
                  if(m_procesador == "P3")
                  {
                    if (hex >= "F0")
                    {
                      m_DDRCh = hex;
                      m_DDRCb = bin;
                    }
                    else
                      MessageBox("Error en la asignaci¾n
al Puerto C","Error",MB_OK);
                  }
                  else
                  {
                    m_DDRCh = hex;
                    m_DDRCb = bin;
                  }
                  break;
                 }
        case '8':{
                  m_TDRh = hex;
                  m_TDRb = bin;
                  break;
                 }
        case '9':{
                  m_TCRh = hex;
                  m_TCRb = bin;
                  break;
                 }
    }
  }
}

void CxDialog::ADC(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;
  int cmp1;
  int cmp2;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  switch(tipins)
  {
    case 4:{
            contenido = operando;
            m_ciclos += 2;
            break;
           }
    case 5:{
            dir = "00";
            dir += operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 4;
```

```
            break;
        }
    case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
        }
    case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
        }
    case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
        }
    case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
             strupr(_itoa(A,operando,16));
            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
        }
  }
  a1.Iniciadato((const char *)contenido);
  a2.Iniciadato((const char *)m_AAh);
  cmp1 = a1.binario[4];
  cmp2 = a1.binario[0];
  a1 = a1 + a2;
  a1 = a1 + (m_C[0]-48);
  m_N = a1.binario[0];
    if((m_AAb[4]-48)   &&   cmp1   ||   cmp1   &&
(a1.binario[4]-48) || (a1.binario[4]-48) &&(m_AAb[4]-48))
      m_H = '1';
  else
      m_H = '0';
    if((!(m_AAb[0]-48)   &&   cmp2   ||   cmp2   &&
(a1.binario[0]-48) || (a1.binario[0]-48) &&(m_AAb[0]-48))
      m_C = '1';
  else
      m_C = '0';
  m_AAh = a1.hexadecimal;
  m_AAb = a1.binario;
  if(m_AAb = = "00000000")
      m_Z = '1';
  else
      m_Z = '0';
  CalcDiaSegundos();
  indice + + ;
}

void CxDialog::ADD(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];

  char operando[5];
  int A;
  int B;
  int cmp1;
  int cmp2;

  if(CO.GetLength() > = 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() = = 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
      case 4:{
              contenido = operando;
              m_ciclos + = 2;
              break;
          }
      case 5:{
              dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 4;
              break;
          }
      case 6:{
              dir = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 5;
              break;
          }
      case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenido = ObtenIndice(dir);
              m_ciclos + = 4;
              break;
          }
      case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir + = a1.hexadecimal;
              contenido = ObtenIndice(dir);
              m_ciclos + = 5;
              break;
          }
      case 9:{
              A = ConvierteHexadecimalaEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalaEntero(aux);
              A = A + B;
               strupr(_itoa(A,operando,16));
              dir = "0";
              dir + = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 6;
              break;
          }
  }
```

```
    a1.Iniciadato((const char *)contenido);
    a2.Iniciadato((const char *)m_AAh);
    cmp1 = a1.binario[4];
    cmp2 = a1.binario[0];
    a1 = a1 + a2;
    m_N = a1.binario[0];
      if((m_AAb[4]-48)    &&    cmp1    ||    cmp1    &&
(a1.binario[4]-48) || (a1.binario[4]-48) && (m_AAb[4]-48))
      m_H = '1';
    else
      m_H = '0';
      if((m_AAb[0]-48)    &&    cmp2    ||    cmp2    &&
(a1.binario[0]-48) || (a1.binario[0]-48) && (m_AAb[0]-48))
      m_C = '1';
    else
      m_C = '0';
    m_AAh = a1.hexadecimal;
    m_AAb = a1.binario;
    if(m_AAb == "00000000")
      m_Z = '1';
    else
      m_Z = '0';

    CalculaSegundos();
    indice + + ;
}

void CxDialog::AND(int tipins)
{
    CString dir;
    CString contenido;

    char aux[3];
    char operando[5];
    int A;
    int B;

    if(CO.GetLength() > = 4)
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() = = 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
      operando[0] = '\0';

    switch(tipins)
    {
      case 4:{
              contenido = operando;
              m_ciclos + = 2;
              break;
             }
      case 5:{
              dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 4;
              break;
             }
      case 6:{
              dir = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 6;
              break;
             }
      case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenid5 = ObtenIndice(dir);
              m_ciclos + = 4;
              break;
             }
      case 8:{
              a1.Iniciadato((const char *)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir + = a1.hexadecimal;
              contenido = ObtenIndice(dir);
              m_ciclos + = 5;
              break;
             }
      case 9:{
              A = ConvierteHexadacimalaEntero(operando);
              strcpy(aux,(const char *) m_INDXh);
              B = ConvierteHexadacimalaEntero(aux);
              A = A + B;
              _itoa(A,operando,16);
              dir = "0";
              dir + = operando;
              contenido = ObtenIndice(dir);
              m_ciclos + = 6;
              break;
             }
    }
    a1.Iniciadato((const char *) contenido);
    a2.Iniciadato((const char *)m_AAh);
    a1 = a1&&a2;
    m_AAh = a1.hexadecimal;
    m_AAb = a1.binario;
    m_N = m_AAb[0];
    if(m_AAb = = "00000000")
      m_Z = '1';
    else
      m_Z = '0';

    CalculaSegundos();
    indice + + ;
}

void CxDialog::ASLoLSL(int tipins)
{
    CString dir;
    CString contenido;
    CString bin;
    CString hex;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char operando[5];
    int index;

    if(CO.GetLength() > = 4)
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() = = 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
```

```
    operando[0] = '\0';
  switch(tipins)
  {
     case 2:{
              contenido = m_AAh;
              m_ciclos + = 4;
              break;
            }
     case 3:{
              contenido = m_INDXh;
              m_ciclos + = 4;
              break;
            }
     case 5:{
              dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
            }
     case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
            }
     case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir + = a1.hexadecimal;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 7;
              break;
            }
  }
      bin = ConvierteHexadecimalaBinario((const  char  *)
contenido);
    m_C = bin[0];
    for(int i = 1;i < 7;i + + )
    {
      bin.SetAt(i-1,bin[i]);
    }
    bin.SetAt(7,'0');
    hex = ConvierteBinarioaHexadecimal((const char *) bin);
    if(tipins = = 2)
    {
      m_AAh = hex;
      m_AAb = bin;
    }
    if(tipins = = 3)
    {
      m_INDXh = hex;
      m_INDXb = bin;
    }
    if(tipins > = 4)
    {
      m_mem.SetAt(index,hex[0]);
      m_mem.SetAt(index + 1,hex[1]);
    }
    m_N = bin[0];
    if(bin = = "00000000")
      m_Z = 1;
    else
      m_Z = '0';
    IntroducesMemoria(dir,hex,bin);
    CalculaSegundos();
    indice + + ;
}
void CxDialog::ASR(int tipins)
{
    CString dir;
    CString contenido;
    CString bin;
    CString hex;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char operando[5];
    int index;

    if(CO.GetLength() > = 4)
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() > = 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
      operando[0] = '\0';

    switch(tipins)
    {
       case 2:{
                contenido = m_AAh;
                m_ciclos + = 4;
                break;
              }
       case 3:{
                contenido = m_INDXh;
                m_ciclos + = 4;
                break;
              }
       case 5:{
                dir = "00";
                dir + = operando;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 6;
                break;
              }
       case 7:{
                dir = "00";
                dir + = m_INDXh;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 6;
                break;
              }
       case 8:{
                a1.Iniciadato((const char*)m_INDXh);
                a2.Iniciadato(operando);
                a1 = a1 + a2;
                dir = "00";
                dir + = a1.hexadecimal;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 7;
                break;
              }
    }
        bin = ConvierteHexadecimalaBinario((const  char  *)
contenido);
    m_C = bin[7];
    for(int i = 7;i < 0;i--)
    {
```

```
    bin.SetAt(i,bin[i-1]);
  }
  hex = ConvierteBinarioaHexadecimal((const char *) bin);
  if(tipins == 2)
  {
    m_AAh = hex;
    m_AAb = bin;
  }
  if(tipins == 3)
  {
    m_INDXh = hex;
    m_INDXb = bin;
  }
  if(tipins > = 4)
  {
    m_mem.SetAt(index,hex[0]);
    m_mem.SetAt(index + 1,hex[1]);
  }
  m_N = bin[0];
  if(bin == "00000000")
    m_Z = '1';
  else
    m_Z = '0';
  IntroduceaMemoria(dir,hex,bin);
  CalculaSegundos();
  indice + + ;
}

void CxDialog::BCCoBHS(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos + = 4;
  CalculaSegundos();
  if(m_C = = '0')
  {
    if(strcmp(operando,"80") > = 0)
    {
      a2 = a2-a1;
      pc = pc-a2 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice + + ;
    }
  }
  else
    indice + + ;
}

void CxDialog::BEQ(int tipins)
{

  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos + = 4;
  CalculaSegundos();

  if(m_Z = = '1')
  {
    if(strcmp(operando,"80") > = 0)
    {
      a2 = a2-a1;
      pc = pc-a2 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
```

```
    {
      pc = pc + a1 + 2;
      _strupr(_itoa(pc,aux,16));
        dirSalto = "000";


      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
    indice++;
}

void CxDialog::BHCC(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

a1 = ConvierteHexadecimalaEntero(operando);
a2 = 256;
pc = ConvierteHexadecimalaEntero((const char*) m_cp);
m_ciclos += 4;
CalculaSegundos();

if(m_H == '0')
{
  if(strcmp(operando,"80") >= 0)
  {
    a2 = a2-a1;
    pc = pc-a2 + 2;
    _strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice--;
  }
  if(strcmp(operando,"7E") < 0)
  {
    pc = pc + a1 + 2;
    _strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice++;
  }
}
else
  indice++;
}

void CxDialog::BHCS(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

a1 = ConvierteHexadecimalaEntero(operando);
a2 = 256;
pc = ConvierteHexadecimalaEntero((const char*) m_cp);
m_ciclos += 4;
CalculaSegundos();

if(m_H == '1')
{
  if(strcmp(operando,"80") >= 0)
  {
    a2 = a2-a1;
    pc = pc-a2 + 2;
    _strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice--;
  }
  if(strcmp(operando,"7E") < 0)
  {
    pc = pc + a1 + 2;
    _strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
```

```
        while(pInstruccion[indice].PC != dirSalto)
          indice++;
    }
  }
  else
    indice++;
}

void CxDialog::BHI(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos += 4;
  CalculaSegundos();

  if(m_Z == '0' || m_C == '0')
  {
    if(strcmp(operando,"80") > 0)
    {
      a2 = a2-a1;
      pc = pc-a2+2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
    indice++;
}

void CxDialog::BIH(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos += 4;
  CalculaSegundos();

  if(m_INT == '0')
  {
    if(strcmp(operando,"80") > 0)
    {
      a2 = a2-a1;
      pc = pc-a2+2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
    indice++;
}

void CxDialog::BIL(int tipins)
{
```

```
char operando[5];
char aux[5];
CString dirSalto;
int a1;
int a2;
int pc;

if(CO.GetLength() >= 4)
{
   operando[0] = CO[2];
   operando[1] = CO[3];
   operando[2] = '\0';

   if(CO.GetLength() >= 6)
   {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
   }
}
else
   operando[0] = '\0';

a1 = ConvierteHexadecimalaEntero(operando);
a2 = 256;
pc = ConvierteHexadecimalaEntero((const char*) m_cp);
m_ciclos += 4;
CalculaSegundos();

if(m_INT == '1')
{
   if(strcmp(operando,"80") >= 0)
   {
      a2 = a2-a1;
      pc = pc-a2 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
         dirSalto = "000";
      if(strlen(aux) == 2)
         dirSalto = "00";
      if(strlen(aux) == 3)
         dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
         indice--;
   }
   if(strcmp(operando,"7E")<0)
   {
      pc = pc + a1 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
         dirSalto = "000";
      if(strlen(aux) == 2)
         dirSalto = "00";
      if(strlen(aux) == 3)
         dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
         indice++;
   }
}
else
   indice++;
}

void CxDialog::BIT(int tipins)
{
   CString dir;
   CString contenido;
   CHexadecimal a1("--");
   CHexadecimal a2("--");

char aux[3];
char operando[5];
int A;
int B;

if(CO.GetLength() >= 4)
{
   operando[0] = CO[2];
   operando[1] = CO[3];
   operando[2] = '\0';

   if(CO.GetLength() == 6)
   {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
   }
}
else
   operando[0] = '\0';

switch(tipins)
{
   case 4:{
            contenido = operando;
            m_ciclos += 2;
            break;
          }
   case 5:{
            dir = "00";
            dir += operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 4;
            break;
          }
   case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 5;
            break;
          }
   case 7:{
            dir = "00";
            dir += m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos += 4;
            break;
          }
   case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir += a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos += 5;
            break;
          }
   case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
            strupr( itoa(A,operando,16));
            dir = "0";
            dir += operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 6;
            break;
          }
}
a1.Iniciadato((const char *) contenido);
```

```
    a2.Iniciadato((const char *)m_AAh);
    a1 = a1&&a2;
    m_N = a1.binario[0];

    CalculeSegundos();
    indice++;
}


void CxDialog::BLOoBCS(int tipins)
{
    char operando[5];
    char aux[5];
    CString dirSalto;
    int a1;
    int a2;
    int pc;

    if(CO.GetLength() > = 4)
    {
        operando[0] = CO[2];
        operando[1] = CO[3];
        operando[2] = '\0';

        if(CO.GetLength() = = 6)
        {
            operando[2] = CO[4];
            operando[3] = CO[5];
            operando[4] = '\0';
        }
    }
    else
        operando[0] = '\0';

    a1 = ConvierteHexadecimalaEntero(operando);
    a2 = 256;
    pc = ConvierteHexadecimalaEntero((const char*) m_cp);
    m_ciclos + = 10;
    CalculaSegundos();

    if(m_C = = '1')
    {
        if(strcmp(operando,"80") > = 0)
        {
            a2 = a2-a1;
            pc = pc-a2 + 2;
            _strupr( _itoa(pc,aux,16));
            if(strlen(aux) = = 1)
                dirSalto = "000";
            if(strlen(aux) = = 2)
                dirSalto = "00";
            if(strlen(aux) = = 3)
                dirSalto = "0";
            dirSalto + = aux;
            while(p[instruccion[indice].PC ! = dirSalto)
                indice--;
        }
        if(strcmp(operando,"7E") < 0)
        {
            pc = pc + a1 + 2;
            _strupr( _itoa(pc,aux,16));
            if(strlen(aux) = = 1)
                dirSalto = "000";
            if(strlen(aux) = = 2)
                dirSalto = "00";
            if(strlen(aux) = = 3)
                dirSalto = "0";
            dirSalto + = aux;
            while(p[instruccion[indice].PC ! = dirSalto)
                indice + + ;
        }
    }
    else
        indice + + ;
}

void CxDialog::BLS(int tipins)
{
    char operando[5];
    char aux[5];
    CString dirSalto;
    int a1;
    int a2;
    int pc;

    if(CO.GetLength() > = 4)
    {
        operando[0] = CO[2];
        operando[1] = CO[3];
        operando[2] = '\0';

        if(CO.GetLength() = = 6)
        {
            operando[2] = CO[4];
            operando[3] = CO[5];
            operando[4] = '\0';
        }
    }
    else
        operando[0] = '\0';

    a1 = ConvierteHexadecimalaEntero(operando);
    a2 = 256;
    pc = ConvierteHexadecimalaEntero((const char*) m_cp);
    m_ciclos + = 4;
    CalculaSegundos();

    if(m_Z = = '1' || m_C = = '1')
    {
        if(strcmp(operando,"80") > = 0)
        {
            a2 = a2-a1;
            pc = pc-a2 + 2;
            _strupr( _itoa(pc,aux,16));
            if(strlen(aux) = = 1)
                dirSalto = "000";
            if(strlen(aux) = = 2)
                dirSalto = "00";
            if(strlen(aux) = = 3)
                dirSalto = "0";
            dirSalto + = aux;
            while(p[instruccion[indice].PC ! = dirSalto)
                indice--;
        }
        if(strcmp(operando,"7E") < 0)
        {
            pc = pc + a1 + 2;
            _strupr( _itoa(pc,aux,16));
            if(strlen(aux) = = 1)
                dirSalto = "000";
            if(strlen(aux) = = 2)
                dirSalto = "00";
            if(strlen(aux) = = 3)
                dirSalto = "0";
            dirSalto + = aux;
            while(p[instruccion[indice].PC ! = dirSalto)
                indice + + ;
        }
    }
    else
        indice + + ;
}
```

```
void CxDialog::BMC(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos + = 4;
  CalculaSegundos();

  if(m_f = = '0')
  {
    if(strcmp(operando,"80") > = 0)
    {
      a2 = a2-a1;
      pc = pc-a2 + 2;
       strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC ! = dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
       strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC ! = dirSalto)
        indice + + ;
    }
  }
  else
    indice + + ;
}

void CxDialog::BMI(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos + = 4;
  CalculaSegundos();

  if(m_N = = '1')
  {
    if(strcmp(operando,"80") > = 0)
    {
      a2 = a2-a1;
      pc = pc-a2 + 2;
       strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC ! = dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
       strupr( itoa(pc,aux,16));
      if(strlen(aux) = = 1)
        dirSalto = "000";
      if(strlen(aux) = = 2)
        dirSalto = "00";
      if(strlen(aux) = = 3)
        dirSalto = "0";
      dirSalto + = aux;
      while(pInstruccion[indice].PC ! = dirSalto)
        indice + + ;
    }
  }
  else
    indice + + ;
}

void CxDialog::BMS(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() > = 4)
```

```
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';
      if(CO.GetLength() == 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
      operando[0] = '\0';
    a1 = ConvierteHexadecimalaEntero(operando);
    a2 = 256;
    pc = ConvierteHexadecimalaEntero((const char*) rn_cp);
    m_ciclos += 4;
    CalculaSegundos();
    if(m_I == '1')
    {
      if(strcmp(operando,"80") >= 0)
      {
        a2 = a2-a1;
        pc = pc-a2+2;
        strupr( itoa(pc,aux,16));
        if(strlen(aux) == 1)
          dirSalto = "000";
        if(strlen(aux) == 2)
          dirSalto = "00";
        if(strlen(aux) == 3)
          dirSalto = "0";
        dirSalto += aux;
        while(pInstruccion[indice].PC != dirSalto)
          indice--;
      }
      if(strcmp(operando,"7E") < 0)
      {
        pc = pc + a1 + 2;
        strupr( itoa(pc,aux,16));
        if(strlen(aux) == 1)
          dirSalto = "000";
        if(strlen(aux) == 2)
          dirSalto = "00";
        if(strlen(aux) == 3)
          dirSalto = "0";
        dirSalto += aux;
        while(pInstruccion[indice].PC != dirSalto)
          indice++;
      }
    }
    else
      indice++;
}

void CxDialog::BNE(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';
    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos += 4;
  CalculaSegundos();

  if(m_Z == '0')
  {
    if(strcmp(operando,"80") >= 0)
    {
      a2 = a2-a1;
      pc = pc-a2+2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = pc + a1 + 2;
      strupr( itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)



      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
    indice++;
}

void CxDialog::BPL(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
```

```
    operando[4] = '\0';
  }
}
else
  operando[0] = '\0';

a1 = ConvierteHexadecimalaEntero(operando);
a2 = 256;
pc = ConvierteHexadecimalaEntero((const char*) m_cp);
m_ciclos += 4;
CalculaSegundos();

if(m_N == '0')
{
  if(strcmp(operando,"80") >= 0)
  {
    a2 = a2-a1;
    pc = pc-a2+2;
     strupr( itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice--;
  }
  if(strcmp(operando,"7E") < 0)
  {
    pc = pc + a1 + 2;
     strupr( itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice++;
  }
}
else
  indice++;
}

void CxDialog::BRA(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

a1 = ConvierteHexadecimalaEntero(operando);
a2 = 256;
pc = ConvierteHexadecimalaEntero((const char*) m_cp);
m_ciclos += 4;
CalculaSegundos();

if(strcmp(operando,"80") >= 0)
{
  a2 = a2-a1;
  pc = pc-a2+2;
   strupr( itoa(pc,aux,16));
  if(strlen(aux) == 1)
    dirSalto = "000";
  if(strlen(aux) == 2)
    dirSalto = "00";
  if(strlen(aux) == 3)
    dirSalto = "0";
  dirSalto += aux;
  while(pInstruccion[indice].PC != dirSalto)
    indice--;
}
if(strcmp(operando,"7E") < 0)
{
  pc = pc + a1 + 2;
   strupr( itoa(pc,aux,16));
  if(strlen(aux) == 1)
    dirSalto = "000";
  if(strlen(aux) == 2)
    dirSalto = "00";
  if(strlen(aux) == 3)
    dirSalto = "0";
  dirSalto += aux;
  while(pInstruccion[indice].PC != dirSalto)
    indice++;
}
}

void CxDialog::BRN(int tipins)
{
  char operando[5];
  char aux[5];
  CString dirSalto;
  int a1;
  int a2;
  int pc;

  if(CO.GetLength() >= 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() == 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  a1 = ConvierteHexadecimalaEntero(operando);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*) m_cp);
  m_ciclos += 4;
  CalculaSegundos();

  if(strcmp(operando,"80") >= 0)
  {
    a2 = a2-a1;
    pc = pc-a2+2;
```

```
    strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice--;
  }
  if(strcmp(operando,"7E") < 0)
  {
    pc = pc + a1 + 2;
    strupr(_itoa(pc,aux,16));
    if(strlen(aux) == 1)
      dirSalto = "000";
    if(strlen(aux) == 2)
      dirSalto = "00";
    if(strlen(aux) == 3)
      dirSalto = "0";
    dirSalto += aux;
    while(pInstruccion[indice].PC != dirSalto)
      indice++;
  }
}

void CxDialog::BSR(int tipins)
{
  char operando[5];
  char aux[5];
  CString dir;
  CString dirSalto;
  CString incPC;
  CString contenido;
  CHexadecimal a1("");
  int n1;
  int n2;
  int pc;
  int index;

  operando[0] = CO[2];
  operando[1] = CO[3];
  operando[2] = '\0';

  pc = ConvierteHexadecimalaEntero((const char*)m_cp);
  pc = pc + 2;
  strupr(_itoa(pc,aux,16));
  if(strlen(aux) == 1)
    incPC = "000";
  if(strlen(aux) == 2)
    incPC = "00";
  if(strlen(aux) == 3)
    incPC = "0";
  incPC += aux;

  aux[0] = m_SP[1];
  aux[1] = m_SP[2];
  aux[2] = '\0';

  a1.Iniciadato(aux);
  a1 = a1-2;
  m_SP = "0";
  m_SP = a1.hexadecimal;
  dir = "0";
  dir += m_SP;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,incPC[0]);
  m_mem.SetAt(index+1,incPC[1]);
  if (m_mem[2] != '\r')
  {
    m_mem.SetAt(index+5,incPC[2]);
    m_mem.SetAt(index+6,incPC[3]);
  }
  else
  {
    m_mem.SetAt(index+10,incPC[2]);
    m_mem.SetAt(index+11,incPC[3]);
  }
  n1 = ConvierteHexadecimalaEntero(operando);
  n2 = 256;

  m_ciclos += 8;
  CalculaSegundos();

  if(m_SP > "060" && m_SP < "0FF")
  {
    if(strcmp(operando,"80") >= 0)
    {
      n2 = n2-n1;
      pc = pc-n2;
      strupr(_itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc += n1;
      strupr(_itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
  {
    MessageBox("Stack Sobrecargado","Error",MB_OK);
    indice++;
  }
}

void CxDialog::CLC(int tipins)
{
  m_C = "0";
  m_ciclos += 2;
  CalculaSegundos();

  indice++;
}

void CxDialog::CLI(int tipins)
{
  m_I = "0";
  m_ciclos += 2;
  CalculaSegundos();

  indice++;
}

void CxDialog::CLR(int tipins)
{
```

```
CString dir;
char operando[5];
CString contenido;
CString hex;
CString bin;
int index;
CHexadecimal a1("");
CHexadecimal a2("");

if(CO.GetLength() > = 4)
{
   operando[0] = CO[2];
   operando[1] = CO[3];
   operando[2] = '\0';

   if(CO.GetLength() = = 6)
   {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
   }
}
else
   operando[0] = '\0';

switch(tipins)
{
   case 2:{
            m_ciclos + = 4;
            m_AAD = "0000000000";
            m_AAh = "00";
            break;
          }
   case 3:{
            m_ciclos + = 4;
            m_INDXb = "0000000000";
            m_INDXh = "00";
            break;
          }
   case 5:{
            m_ciclos + = 6;
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir,index);
            break;
          }
   case 7:{
            m_ciclos + = 6;
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir,index);
            break;
          }
   case 8:{
            m_ciclos + = 7;
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir,index);

            break;
          }
}
if(tipins > = 5)
{
   m_mem.SetAt(index,'0');
   m_mem.SetAt(index + 1,'0');
}
m_N = '0';
m_Z = '1';
CalculaSegundos();
hex = "00";
bin = "00000000";
IntroduceaMemoria(dir,hex,bin);
indice + +;
}

void CxDialog::CMP(int tipins)
{
   CString dir;
   CString contenido;
   CString bin;
   CHexadecimal a1("");
   CHexadecimal a2("");
   char aux[3];
   char operando[5];
   int A;
   int B;
   int cmp1;

   if(CO.GetLength() > = 4)
   {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() = = 6)
      {
         operando[2] = CO[4];
         operando[3] = CO[5];
         operando[4] = '\0';
      }
   }
   else
      operando[0] = '\0';

   switch(tipins)
   {
      case 4:{
               contenido = operando;
               m_ciclos + = 2;
               break;
             }
      case 5:{
               dir = "00";
               dir + = operando;
               contenido = ObtenIndice(dir);
               m_ciclos + = 4;
               break;
             }
      case 6:{
               dir = operando;
               contenido = ObtenIndice(dir);
               m_ciclos + = 5;
               break;
             }
      case 7:{
               dir = "00";
               dir + = m_INDXh;
               contenido = ObtenIndice(dir);
               m_ciclos + = 4;
               break;
             }
      case 8:{
               a1.Iniciadato((const char*)m_INDXh);
               a2.Iniciadato(operando);
               a1 = a1 + a2;
               dir = "00";
               dir + = a1.hexadecimal;
               contenido = ObtenIndice(dir);
               m_ciclos + = 5;
```

```
            break;
          }
      case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;

            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
          }
    }

    cmp1 = a1.binario[0]-48;
    a1 = a2-a1;
    m_N = a1.binario[0];
    if(!strcmp(a1.binario,"00000000"))
      m_Z = "1";
    else
      m_Z = "0";
        if((m_AAb[0]-48)   &&     cmp1    ||    cmp1    &&
(a1.binario[0]-48)          ||          (a1.binario[0]-48)          &&
(m_AAb[0]-48))
      m_C = "1";
    else
      m_C = "0";
    CalculaSegundos();

    indice + + ;
}

void CxDialog::COM(int tipins)
{
  CString dir;
  CString contenido;
  CString bin;
  CString hex;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char operando[5];
  int index;

  if(CO.GetLength() > = 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() = = 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
      case 2:{
            contenido = m_AAh;
            m_ciclos + = 4;
            break;
          }
      case 3:{
            contenido = m_INDXh;
            m_ciclos + = 4;
            break;
          }
      case 5:{
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 6;
            break;
          }
      case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 6;
            break;
          }
      case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 7;
            break;
          }
  }
  a1.Iniciadato((const char*)contenido);
  a1 = -a1;

  m_C = "1";

  if(tipins = = 2)
  {
    m_AAh = a1.hexadecimal;
    m_AAb = a1.binario;
  }
  if(tipins = = 3)
  {
    m_INDXh = a1.hexadecimal;
    m_INDXb = a1.binario;
  }
  if(tipins > = 4)
  {
    m_mem.SetAt(index,a1.hexadecimal[0]);
    m_mem.SetAt(index + 1,a1.hexadecimal[1]);
  }

  m_N = a2.binario[0];

  if(!strcmp(a1.binario,"00000000"))
    m_Z = "1";
  else
    m_Z = "0";
  CalculaSegundos();
  IntroducealMemoria(dir, a1.hexadecimal,a1.binario);
  indice + + ;
}

void CxDialog::CPX(int tipins)
{
  CString dir;
  CString contenido;
  CString bin;
  CString hex;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char operando[5];
  char aux[5];
  int A;
  int B;
```

```
int cmp1;
if(CO.GetLength() > = 4)
{
   operando[0] = CO[2];
   operando[1] = CO[3];
   operando[2] = '\0';

   if(CO.GetLength() = = 6)
   {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
   }
}
else
   operando[0] = '\0';

switch(tipins)
{
   case 4:{
            contenido = operando;
            m_ciclos + = 2;
            break;
          }
   case 5:{
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
          }
   case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
          }
   case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
          }
   case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
          }
   case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
            strupr(_itoa(A,operando,16));
            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
          }
}
a1.Iniciadato((const char*)contenido);
a2.Iniciadato((const char*)m_INDXh);
cmp1 = a2.binario[0]-48;
a2 = a2-a1;

   if((cmp1 && (a1.binario[0]-48) || (a1.binario[0]-48) &&
(a2.binario[0]-48) || (a2.binario[0]-48) && !cmp1)
      m_C = '1';
   else
      m_C = '0';

   m_N = a2.binario[0];

   if(!strcmp(a2.binario,"00000000"))
      m_Z = '1';
   else
      m_Z = '0';
   CalculaSegundos();

   indice + +;
}

void CxDialog::DEC(int tipins)
{
   CString dir;
   CString contenido;
   CString bin;
   CString hex;
   CHexadecimal a1("");
   CHexadecimal a2("");
   char operando[5];
   int index;

   if(CO.GetLength() > = 4)
   {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() = = 6)
      {
         operando[2] = CO[4];
         operando[3] = CO[5];
         operando[4] = '\0';
      }
   }
   else
      operando[0] = '\0';

   switch(tipins)
   {
      case 2:{
               contenido = m_AAh;
               m_ciclos + = 4;
               break;
             }
      case 3:{
               contenido = m_INDXh;
               m_ciclos + = 4;
               break;
             }
      case 5:{
               dir = "00";
               dir + = operando;
               contenido = ObtenIndice(dir, index);
               m_ciclos + = 6;
               break;
             }
      case 7:{
               dir = "00";
               dir + = m_INDXh;
               contenido = ObtenIndice(dir, index);
               m_ciclos + = 6;
               break;
             }
      case 8:{
```

```
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 7;
            break;
          }
  }
  a1.Iniciadato((const char*)contenido);
  a1 = a1 - 1;

  m_C = a1.binario[0];

  if(tipins = = 2)
  {
    m_AAh = a1.hexadecimal;
    m_AAb = a1.binario;
  }
  if(tipins = = 3)
  {
    m_INDXh = a1.hexadecimal;
    m_INDXb = a1.binario;
  }
  if(tipins > = 5)
  {
    m_mem.SetAt(index, a1.hexadecimal[0]);
    m_mem.SetAt(index + 1, a1.hexadecimal[1]);
  }

  m_N = a1.binario[0];

  if(!strcmp(a1.binario, "00000000"))
    m_Z = '1';
  else
    m_Z = '0';
  CalculaSegundos();
  IntroduceMemoria(dir, a1.hexadecimal, a1.binario);
  indice + + ;
}

void CxDialog::EOR(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  switch(tipins)
  {
    case 4:{
            contenido = operando;
            m_ciclos + = 2;
            break;
           }
    case 5:{
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
           }
    case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
           }
    case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
             strupr(_itoa(A,operando,16));
            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
           }
  }
  a1.Iniciadato((const char *) contenido);
  a2.Iniciadato((const char *)m_AAh);
  a1 = a1 ^ a2;
  m_AAh = a1.hexadecimal;
  m_AAb = a1.binario;
  m_N = m_AAb[0];
  if(m_AAb = = "00000000")
    m_Z = '1';
  else
    m_Z = '0';
  CalculaSegundos();

  indice + + ;
}

void CxDialog::INC(int tipins)
{
  CString dir;
  CString contenido;
  CString bin;
  CString hex;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char operando[5];
```

```
int index;
if(CO.GetLength() > = 4)
{
   operando[0] = CO[2];
   operando[1] = CO[3];
   operando[2] = '\0';
   if(CO.GetLength() = = 6)
   {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
   }
}
else
   operando[0] = '\0';
switch(tipins)
{
   case 2:{
            contenido = m_AAh;
            m_ciclos + = 4;
            break;
          }
   case 3:{
            contenido = m_INDXh;
            m_ciclos + = 4;
            break;
          }
   case 5:{
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 6;
            break;
          }
   case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 6;
            break;
          }
   case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir, index);
            m_ciclos + = 7;
            break;
          }
}
a1.Iniciadato((const char*)contenido);
a1 = a1 + 1;
m_C = a1.binario[0];
if(tipins = = 2)
{
   m_AAh = a1.hexadecimal;
   m_AAb = a1.binario;
}
if(tipins = = 3)
{
   m_INDXh = a1.hexadecimal;
   m_INDXb = a1.binario;
}
if(tipins > = 5)
{
   m_mem.SetAt(index,a1.hexadecimal[0]);
   m_mem.SetAt(index + 1,a1.hexadecimal[1]);
}
m_N = a1.binario[0];
if(strcmp(a1.binario,"00000000") = = 0)
   m_Z = '1';
else
   m_Z = '0';
CalculaSegundos();
IntroducesMemoria(dir, a1.hexadecimal,a1.binario);
indice + +;
}
void CxDialog::JMP(int tipins)
{
   CString dir;
   CString contenido;
   CString incPC;
   CString dirSalto;
   CHexadecimal a1("");
   CHexadecimal a2("");
   char aux[5];
   char operando[5];
   int A;
   int B;
   if(CO.GetLength() > = 4)
   {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';
      if(CO.GetLength() = = 6)
      {
         operando[2] = CO[4];
         operando[3] = CO[5];
         operando[4] = '\0';
      }
   }
   else
      operando[0] = '\0';
   switch(tipins)
   {
      case 5:{
               dirSalto = "00";
               dirSalto + = operando;
               m_ciclos + = 3;
               break;
             }
      case 6:{
               dirSalto = operando;
               m_ciclos + = 4;
               break;
             }
      case 7:{
               dirSalto = "00";
               dirSalto + = m_INDXh;
               m_ciclos + = 3;
               break;
             }
      case 8:{
               a1.Iniciadato((const char*)m_INDXh);
               a2.Iniciadato(operando);
               a1 = a1 + a2;
               dirSalto = "00";
               dirSalto + = a1.hexadecimal;
               m_ciclos + = 4;
               break;
             }
```

```
      case 9:{
              A = ConvierteHexadecimalaEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalaEntero(aux);
              A = A + B;
               _itoa(A,operando,16));
              dirSalto = "0";
              dirSalto + = operando;
              m_ciclos + = 5;
              break;
            }
  }
  if (dirSalto > m_cp)
  {
    while(pInstruccion[indice].PC != dirSalto)
      indice + +;
  }
  else
  {
    while(pInstruccion[indice].PC != dirSalto)
      indice--;
  }

  CalculaSegundos();
}

void CxDialog::JSR(int tipins)
{
  CString dir;
  CString contenido;
  CString incPC;
  CString dirSalto;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[5];
  char operando[5];
  int A;
  int B;
  int pc;
  int index;
  char progcount[5];

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  pc = ConvierteHexadecimalaEntero((const char*)m_cp);
  switch(tipins)
  {
    case 5:   pc + = 2; break;
    case 6:   pc + = 3; break;
    case 7:   pc + = 1; break;
    case 8:   pc + = 2; break;
    case 9:   pc + = 3; break;
  }
  _strupr( _itoa(pc,progcount,16));
  if(strlen(progcount) = = 3)
    incPC = "0";
  if(strlen(progcount) = = 2)
    incPC = "00";

  incPC + = progcount;

  aux = SP;

  a1.Iniciadato(aux);
  a1 = a1 - 2;
  m_SP = "0";
  m_SP + = a1.hexadecimal;
  if( m_SP > "060" && m_SP < "080")
  {
    dir = '0';                    //falta verificar que m_SP > 60
hex
    dir + = m_SP;
    contenido = ObtenIndice(dir,indox);
    switch(tipins)
    {
      case 5:{
               dirSalto = "00";
               dirSalto + = operando;
               m_ciclos + = 7;
               break;
             }
      case 6:{
               dirSalto = operando;
               m_ciclos + = 8;
               break;
             }
      case 7:{
               dirSalto = "00";
               dirSalto + = m_INDXh;
               m_ciclos + = 7;
               break;
             }
      case 8:{
               a1.Iniciadato((const char*)m_INDXh);
               a2.Iniciadato(operando);
               a1 = a1 + a2;
               dirSalto = "00";
               dirSalto + = a1.hexadecimal;
               m_ciclos + = 8;
               break;
             }
      case 9:{
               A = ConvierteHexadecimalaEntero(operando);
               strcpy(aux,(const char*) m_INDXh);
               B = ConvierteHexadecimalaEntero(aux);
               A = A + B;
               _strupr( _itoa(A,operando,16));
               dirSalto = "0";
               dirSalto + = operando;
               m_ciclos + = 9;
               break;
             }
    }
    m_mem.SetAt(index,incPC[0]);
    m_mem.SetAt(index + 1,incPC[1]);
    if(m_mem[index + 2] != '\r')
    {
      m_mem.SetAt(index + 5,incPC[2]);
      m_mem.SetAt(index + 6,incPC[3]);
    }
    else
    {
      m_mem.SetAt(index + 10,incPC[2]);
      m_mem.SetAt(index + 11,incPC[3]);
    }

      while(pInstruccion[indice].PC != dirSalto)
        indice + +;
```

```
        while(pinstruccion[indice].PC != dirSalto)
          indice--;
      CalculaSegundos();
    }
    else
    {
      MessageBox("Stack Sobrecargado","Error",MB_OK);
      indice++;
    }
}

void CxDialog::LDA(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;

  if(CO.GetLength() >= 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() == 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
     case 4:{
              contenido = operando;
              m_ciclos += 2;
              break;
            }
     case 5:{
              dir = "00";
              dir += operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 4;
              break;
            }
     case 6:{
              dir = operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 5;
              break;
            }
     case 7:{
              dir = "00";
              dir += m_INDXh;
              contenido = ObtenIndice(dir);
              m_ciclos += 4;
              break;
            }
     case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir += a1.hexadecimal;
              contenido = ObtenIndice(dir);
              m_ciclos += 5;
              break;
            }
     case 9:{
              A = ConvierteHexadecimalsEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalsEntero(aux);
              A = A + B;
               strupr(_itoa(A,operando,16));
              dir = "0";
              dir += operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 6;
              break;
            }
  }
  m_AAh = contenido;
  strcpy(aux,(const char*) contenido);
  m_AAb = ConvierteHexadecimalaBinario(aux);
  m_N = m_AAb[0];
  if(m_AAb == "00000000")
     m_Z = '1';
  else
     m_Z = '0';

  CalculaSegundos();
  indice++;
}

void CxDialog::LDX(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;

  if(CO.GetLength() >= 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() == 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
     case 4:{
              contenido = operando;
              m_ciclos += 2;
              break;
            }
     case 5:{
              dir = "00";
              dir += operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 4;
              break;
            }
```

```
    case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
           }
    case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
            _strupr(_itoa(A,operando,16));
            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
           }
  }
  m_INDXh = contenido;
  strcpy(aux,(const char*) contenido);
  m_INDXb = ConvierteHexadecimalaBinario(aux);
  m_N = m_INDXb[0];
  if(m_INDXb = = "00000000")
     m_Z = '1';
  else
     m_Z = '0';

  CalculaSegundos();
  indice + + ;
}

void CxDialog::LSR(int tipins)
{
  CString dir;
  CString contenido;
  CString bin;
  CString hex;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char operando[5];
  int index;

  if(CO.GetLength() > = 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() = = 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
     case 2:{
             contenido = m_AAh;
             m_ciclos + = 4;
             break;
            }
     case 3:{
             contenido = m_INDXh;
             m_ciclos + = 4;
             break;
            }
     case 5:{
             dir = "00";
             dir + = operando;
             contenido = ObtenIndice(dir, index);
             m_ciclos + = 6;
             break;
            }
     case 7:{
             dir = "00";
             dir + = m_INDXh;
             contenido = ObtenIndice(dir, index);
             m_ciclos + = 6;
             break;
            }
     case 8:{
             a1.Iniciadato((const char*)m_INDXh);
             a2.Iniciadato(operando);
             a1 = a1 + a2;
             dir = "00";
             dir + = a1.hexadecimal;
             contenido = ObtenIndice(dir, index);
             m_ciclos + = 7;
             break;
            }
  }

     bin = ConvierteHexadecimalaBinario((const char *)
  contenido);
    m_C = bin[7];
    for(int i = 7;i< 0;i--)
    {
      bin.SetAt(i,bin[i-1]);
    }
    bin.SetAt(0,'0');
    hex = ConvierteBinarioaHexadecimal((const char *) bin);

    if(tipins = = 2)
    {
       m_AAh = hex;
       m_AAb = bin;
    }
    if(tipins = = 3)
    {
       m_INDXh = hex;
       m_INDXb = bin;
    }
    if(tipins > = 4)
    {
       m_mem.SetAt(index,hex[0]);
       m_mem.SetAt(index + 1,hex[1]);
    }
    m_N = bin[0];
    if(bin = = "00000000")
       m_Z = '1';
    else
       m_Z = '0';
    IntroduceaMemoria(dir,hex,bin);
```

```
    CalculaSegundos();
    indice++;
}
void CxDialog::NEG(int tipins)
{
    CString dir;
    CString contenido;
    CHexadecimal a1("--");
    CHexadecimal a2("--");
    char operando[5];
    int index;

    if(CO.GetLength() >= 4)
    {
        operando[0] = CO[2];
        operando[1] = CO[3];
        operando[2] = '\0';

        if(CO.GetLength() == 6)
        {
            operando[2] = CO[4];
            operando[3] = CO[5];
            operando[4] = '\0';
        }
    }
    else
        operando[0] = '\0';

    switch(tipins)
    {
        case 2:{
                contenido = m_AAh;
                m_ciclos += 4;
                break;
               }
        case 3:{
                contenido = m_INDXh;
                m_ciclos += 4;
                break;
               }
        case 5:{
                dir = "00";
                dir += operando;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 6;
                break;
               }
        case 7:{
                dir = "00";
                dir += m_INDXh;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 6;
                break;
               }
        case 8:{
                a1.Iniciadato((const char*)m_INDXh);
                a2.Iniciadato(operando);
                a1 = a1 + a2;
                dir = "00";
                dir += a1.hexadecimal;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 7;
                break;
               }
    }
    a1.Iniciadato((const char *) contenido);
    a1 = -a1;
    a1 = a1 + 1;
    m_N = a1.binario[0];
    if(strcmp(a1.binario,"00000000") == 0)
        m_Z = '1';
    else
        m_Z = '0';
    if(strcmp(a1.binario,"11111111") == 0)
        m_C = '1';
    else
        m_C = '0';
    if(tipins == 2)
    {
        m_AAh = a1.hexadecimal;
        m_AAb = a1.binario;
    }
    if(tipins == 3)
    {
        m_INDXh = a1.hexadecimal;
        m_INDXb = a1.binario;
    }
    if(tipins >= 5)
    {
        m_mem.SetAt(index,a1.hexadecimal[0]);
        m_mem.SetAt(index + 1,a1.hexadecimal[1]);
    }
    CalculaSegundos();
    IntroduceaMemoria(dir,a1.hexadecimal,a1.binario);
    indice++;
}
void CxDialog::NOP(int tipins)
{
    m_ciclos += 2;
    CalculaSegundos();

    indice++;
}
void CxDialog::ORA(int tipins)
{
    CString dir;
    CString contenido;
    CHexadecimal a1("--");
    CHexadecimal a2("--");
    char aux[3];
    char operando[5];
    int A;
    int B;

    if(CO.GetLength() >= 4)
    {
        operando[0] = CO[2];
        operando[1] = CO[3];
        operando[2] = '\0';

        if(CO.GetLength() == 6)
        {
            operando[2] = CO[4];
            operando[3] = CO[5];
            operando[4] = '\0';
        }
    }
    else
        operando[0] = '\0';

    switch(tipins)
    {
        case 4:{
                contenido = operando;
                m_ciclos += 2;
                break;
               }
        case 5:{
```

```
            dir = "00";
            dir += operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 4;
            break;
        }
    case 5:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 5;
            break;
        }
    case 7:{
            dir = "00";
            dir += m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos += 4;
            break;
        }
    case 8:{
            a1.Iniciadato((const char*)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 + a2;
            dir = "00";
            dir += a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos += 5;
            break;
        }
    case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
            sprintf(_itoa(A,operando,16));
            dir = "0";
            dir += operando;
            contenido = ObtenIndice(dir);
            m_ciclos += 6;
            break;
        }
}
a1.Iniciadato((const char *) contenido);
a2.Iniciadato((const char *)m_AAh);
a1 = a1 | a2;
m_AAh = a1.hexadecimal;
m_AAb = a1.binario;
m_N = m_AAb[0];
if(m_AAb == "00000000")
    m_Z = '1';
else
    m_Z = '0';
CalculaSegundos();
indice++;
}

void CxDialog::ROL(int tipins)
{
    CString dir;
    CString contenido;
    CString bin;
    CString hex;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char operando[5];
    char c;
    int index;

    if(CO.GetLength() >= 4)
    {
        operando[0] = CO[2];
        operando[1] = CO[3];
        operando[2] = '\0';
        if(CO.GetLength() == 6)
        {
            operando[2] = CO[4];
            operando[3] = CO[5];
            operando[4] = '\0';
        }
    }

    switch(tipins)
    {
        case 2:{
                contenido = m_AAh;
                m_ciclos += 4;
                break;
            }
        case 3:{
                contenido = m_INDXh;
                m_ciclos += 4;
                break;
            }
        case 5:{
                dir = "00";
                dir += operando;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 6;
                break;
            }
        case 7:{
                dir = "00";
                dir += m_INDXh;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 6;
                break;
            }
        case 8:{
                a1.Iniciadato((const char*)m_INDXh);
                a2.Iniciadato(operando);
                a1 = a1 + a2;
                dir = "00";
                dir += a1.hexadecimal;
                contenido = ObtenIndice(dir, index);
                m_ciclos += 7;
                break;
            }
    }
                bin = ConvierteHexadecimalaBinario((const
char*)contenido);
    c = bin[0];
    for(int i = 1;i < 8;i++)
        bin.SetAt(i-1,bin[i]);

    bin.SetAt(7,m_C[0]);
    hex = ConvierteBinarioaHexadecimal((const char*)bin);

    m_C = c;
    m_N = bin[0];
    if(tipins == 2)
    {
        m_AAh = hex;
        m_AAb = bin;
    }
    if(tipins == 3)
    {
        m_INDXh = hex;
        m_INDXb = bin;
    }
    if(tipins > 5)
```

```
    {
      m_mem.SetAt(index,hex[0]);
      m_mem.SetAt(index + 1,hex[1]);
    }
    if(bin = = "00000000")
      m_Z = '1';
    else
      m_Z = '0';
    CalculaSegundos();
    IntroduceaMemoria(dir,hex,bin);
    indice + + ;
}
void CxDialog::ROR(int tipins)
{
    CString dir;
    CString contenido;
    CString bin;
    CString hex;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char operando[5];
    char c;
    int index;
    if(CO.GetLength() > = 4)
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';
      if(CO.GetLength() = = 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
      operando[0] = '\0';
    switch(tipins)
    {
      case 2:{
              contenido = m_AAh;
              m_ciclos + = 4;
              break;
             }
      case 3:{
              contenido = m_INDXh;
              m_ciclos + = 4;
              break;
             }
      case 5:{  dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
             }
      case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
             }
      case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir + = a1.hexadecimal;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 7;
              break;
             }
    }
              bin = ConvierteHexadecimalaBinario((const
char*)contenido);
    c = bin[7];
    for(int i = 7;i>0;i--)
      bin.SetAt(i,bin[i-1]);
    bin.SetAt(0,m_C[0]);
    hex = ConvierteBinarioaHexadecimal((const char*)bin);
    m_C = c;
    m_N = bin[0];
    if(tipins = = 2)
    {
      m_AAh = hex;
      m_AAb = bin;
    }
    if(tipins = = 3)
    {
      m_INDXh = hex;
      m_INDXb = bin;
    }
    if(tipins > 3)
    {
      m_mem.SetAt(index,hex[0]);
      m_mem.SetAt(index + 1,hex[1]);
    }
    if(bin = = "00000000")
      m_Z = '1';
    else
      m_Z = '0';
    CalculaSegundos();
    IntroduceaMemoria(dir,hex,bin);
    indice + + ;
}
void CxDialog::RSP(int tipins)
{
    int index;
    int sp;
    char aux[4];
    CString dir;
    CString contenido;
    if(m_SP != "07F")
    {
      dir = '0';
      dir + = m_SP;
      while(m_SP != "07F")
      {
        contenido = ObtenIndice(dir,index);
        m_mem.SetAt(index,'0');
        m_mem.SetAt(index + 1,'0');
        if(m_mem[index + 2] != '\r')
        {
          m_mem.SetAt(index + 5,'0');
          m_mem.SetAt(index + 6,'0');
        }
        else
        {
          m_mem.SetAt(index + 10,'0');
          m_mem.SetAt(index + 11,'0');
        }
```

```
                sp = ConvierteHexadecimalaEntero((const
char*)m_SP);
        sp + = 2;
        _strupr(_itoa(sp,aux,16));
        dir = "00";
        dir + = aux;
        m_SP = "0";
        m_SP + = aux;
    }
  }
  m_ciclos + = 2;
  CalculaSegundos();
  indice + + ;
}

void CxDialog::RTI(int tipins)
{

  int index;
  CString incPC;
  CString CC;
  CString CChex;
  CHexadecimal a2("");
  CString dir;
  CString contenido;
  CString bin;
  char aux[5];

  aux[0] = m_SP[1];
  aux[1] = m_SP[2];
  aux[2] = '\0';
  a2.Iniciadato(aux);
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
      CC = ConvierteHexadecimalaBinario((const  char*)
contenido);
  m_H = CC[3];
  m_I = CC[4];
  m_N = CC[5];
  m_Z = CC[6];
  m_C = CC[7];
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  a2 = a2 + 1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_AAh = contenido;
    m_AAb = ConvierteHexadecimalaBinario((const char*)
contenido);
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  a2 = a2 + 1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_INDXh = contenido;
  m_INDXb = ConvierteHexadecimalaBinario((const char*)
contenido);
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  a2 = a2 + 1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_cp = contenido;
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  a2 = a2 + 1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_cp + = contenido;
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');

  m_SP = "0";
  m_SP + = a2.hexadecimal;

  if(pInstruccion[indice].PC < m_cp)
    while(pInstruccion[indice].PC != m_cp)
      indice + + ;
  if(pInstruccion[indice].PC > m_cp)
  while(pInstruccion[indice].PC != m_cp)
    indice--;

  CalculaSegundos();
}

void CxDialog::RTS(int tipins)
{
  int index;
  CString contenido;
  CString dir;
  CString dirSalto;
  char aux[3];
  CHexadecimal a1("");

  dir = "0";
  dir + = m_SP;
  contenido = ObtenIndice(dir,index);
  dirSalto = m_mem[index];
  dirSalto + = m_mem[index + 1];
  if(m_mem[2] != '\r')
  {
    dirSalto + = m_mem[index + 5];
    dirSalto + = m_mem[index + 6];
  }
  else
  {
    m_mem.SetAt(index + 10,'0');
    m_mem.SetAt(index + 11,'0');
  }
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  if(m_mem[2] != '\r')
  {
    m_mem.SetAt(index + 5,'0');
    m_mem.SetAt(index + 6,'0');
  }
  else
  {
    m_mem.SetAt(index + 10,'0');
    m_mem.SetAt(index + 11,'0');
  }
  aux[0] = dir[2];
  aux[1] = dir[3];
  aux[2] = '\0';
  a1.Iniciadato(aux);
  a1 = a1 + 2;
  m_SP = "0";
  m_SP + = a1.hexadecimal;

    while(pInstruccion[indice].PC != dirSalto)
      indice + + ;


    while(pInstruccion[indice].PC != dirSalto)
      indice--;

m_ciclos + = 4;
```

```
    CalculaSegundos();
}

void CxDialog::SBC(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;
  int cmp2;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
    }
  }
  else
    operando[0] = '\0';

  switch(tipins)
  {
    case 4:{
            contenido = operando;
            m_ciclos + = 2;
            break;
           }
    case 5:{
            dir = "00";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
           }
    case 6:{
            dir = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 7:{
            dir = "00";
            dir + = m_INDXh;
            contenido = ObtenIndice(dir);
            m_ciclos + = 4;
            break;
           }
    case 8:{
            a1.Iniciadato((const char *)m_INDXh);
            a2.Iniciadato(operando);
            a1 = a1 - a2;
            dir = "00";
            dir + = a1.hexadecimal;
            contenido = ObtenIndice(dir);
            m_ciclos + = 5;
            break;
           }
    case 9:{
            A = ConvierteHexadecimalaEntero(operando);
            strcpy(aux,(const char*) m_INDXh);
            B = ConvierteHexadecimalaEntero(aux);
            A = A + B;
            _strupr(_itoa(A,operando,16));
            dir = "0";
            dir + = operando;
            contenido = ObtenIndice(dir);
            m_ciclos + = 6;
            break;
           }
  }
  a1.Iniciadato((const char *)contenido);
  a2.Iniciadato((const char *)m_AAh);
  cmp2 = a1.binario[0];
  a1 = a1-a2;
  a1 = a1-(m_C[0]-48);
  m_N = a1.binario[0];
        if((m_AAb[0]-48)   &&   cmp2   ||   cmp2   &&
(a1.binario[0]-48) || (a1.binario[0]-48) && (m_AAb[0]-48))
    m_C = '1';
  else
    m_C = '0';
  m_AAh = a1.hexadecimal;
  m_AAb = a1.binario;
  if(m_AAb = = "00000000")
    m_Z = '1';
  else
    m_Z = '0';

  CalculaSegundos();
  indice + +;
}

void CxDialog::SEC(int tipins)
{
  m_C = '1';
  m_ciclos + = 2;
  CalculaSegundos();

  indice + +;
}

void CxDialog::SEI(int tipins)
{
  m_I = '1';
  m_ciclos + = 2;
  CalculaSegundos();

  indice + +;
}

void CxDialog::STA(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("");
  CHexadecimal a2("");
  char aux[3];
  char operando[5];
  int A;
  int B;
  int index;

  if(CO.GetLength() > = 4)
  {
    operando[0] = CO[2];
    operando[1] = CO[3];
    operando[2] = '\0';

    if(CO.GetLength() = = 6)
    {
      operando[2] = CO[4];
      operando[3] = CO[5];
      operando[4] = '\0';
```

```
    }
  }
  else
    operando[0] = '\0';

  switch(tipins)
  {
     case 5:{
              dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 5;
              break;
            }
     case 6:{
              dir = operando;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
            }
     case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 5;
              break;
            }
     case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir + = a1.hexadecimal;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 6;
              break;
            }
     case 9:{
              A = ConvierteHexadecimalaEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalaEntero(aux);
              A = A + B;
              _itoa(A,operando,16);
              dir = "0";
              dir + = operando;
              contenido = ObtenIndice(dir, index);
              m_ciclos + = 7;
              break;
            }
  }
  m_mem.SetAt(index,m_AAh[0]);
  m_mem.SetAt(index + 1,m_AAh[1]);
  m_N = m_AAb[0];
  if(m_AAb == "00000000")
     m_Z = '1';
  else
     m_Z = '0';

  CalculaSegundos();
  IntroduceMemoria(dir,m_AAh,m_AAb);
  indice + + ;
}

void CxDialog::STOP(int tipins)
{
  indice + + ;
  MessageBox("BRN","Informacion",MB_OK);
}

void CxDialog::STX(int tipins)
{
  CString dir;
  CString contenido;
  CHexadecimal a1("--");
  CHexadecimal a2("--");
  char aux[3];
  char operando[5];
  int A;
  int B;
  int index;

  if(CO.GetLength() > = 4)
  {
     operando[0] = CO[2];
     operando[1] = CO[3];
     operando[2] = '\0';

     if(CO.GetLength() = = 6)
     {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
     }
  }
  else
     operando[0] = '\0';

  switch(tipins)
  {
     case 5:{
              dir = "00";
              dir + = operando;
              contenido = ObtenIndice(dir,index);
              m_ciclos + = 4;
              break;
            }
     case 6:{
              dir = operando;
              contenido = ObtenIndice(dir,index);
              m_ciclos + = 5;
              break;
            }
     case 7:{
              dir = "00";
              dir + = m_INDXh;
              contenido = ObtenIndice(dir,index);
              m_ciclos + = 4;
              break;
            }
     case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = a1.hexadecimal;
              contenido = ObtenIndice(dir,index);
              m_ciclos + = 5;
              break;
            }
     case 9:{
              A = ConvierteHexadecimalaEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalaEntero(aux);
              A = A + B;
              _itoa(A,operando,16);
              dir = "0";
              dir + = operando;
              contenido = ObtenIndice(dir,index);
              m_ciclos + = 6;
              break;
            }
  }
  m_mem.SetAt(index,m_INDXh[0]);
  m_mem.SetAt(index + 1,m_INDXh[1]);
```

```
    m_N = m_INDXb[0];
    if(m_INDXb == "00000000")
      m_Z = '1';
    else
      m_Z = '0';
    CalculaSegundos();
    IntroducesMemoria(dir,m_INDXh,m_INDXb);
    indice++;
}

void CxDialog::SUB(int tipins)
{
    CString dir;
    CString contenido;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char aux[3];
    char operando[5];
    int A;
    int B;
    int cmp2;

    if(CO.GetLength() > 4)
    {
      operando[0] = CO[2];
      operando[1] = CO[3];
      operando[2] = '\0';

      if(CO.GetLength() == 6)
      {
        operando[2] = CO[4];
        operando[3] = CO[5];
        operando[4] = '\0';
      }
    }
    else
      operando[0] = '\0';

    switch(tipins)
    {
      case 4:{
              contenido = operando;
              m_ciclos += 2;
              break;
             }
      case 5:{
              dir = "00";
              dir += operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 4;
              break;
             }
      case 6:{
              dir = operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 5;
              break;
             }
      case 7:{
              dir = "00";
              dir += m_INDXh;
              contenido = ObtenIndice(dir);
              m_ciclos += 4;
              break;
             }
      case 8:{
              a1.Iniciadato((const char*)m_INDXh);
              a2.Iniciadato(operando);
              a1 = a1 + a2;
              dir = "00";
              dir += a1.hexadecimal;
              contenido = ObtenIndice(dir);
              m_ciclos += 5;
              break;
             }
      case 9:{
              A = ConvierteHexadecimalaEntero(operando);
              strcpy(aux,(const char*) m_INDXh);
              B = ConvierteHexadecimalaEntero(aux);
              A = A + B;
              _strupr(_itoa(A,operando,16));
              dir = "00";
              dir += operando;
              contenido = ObtenIndice(dir);
              m_ciclos += 6;
              break;
             }
    }
    a1.Iniciadato((const char *)contenido);
    a2.Iniciadato((const char *)m_AAh);
    cmp2 = a1.binario[0];
    a1 = a1-a2;
    m_N = a1.binario[0];
          if((m_AAb[0]-48)   &&   cmp2   ||   cmp2   &&
(a1.binario[0]-48) || (a1.binario[0]-48) && (m_AAb[0]-48))
      m_C = '1';
    else
      m_C = '0';
    m_AAh = a1.hexadecimal;
    m_AAb = a1.binario;
    if(m_AAb == "00000000")
      m_Z = '1';
    else
      m_Z = '0';

    CalculaSegundos();
    indice++;
}

void CxDialog::SWI(int tipins)
{
    int a1;
    int index;
    char progcount[5];
    CString incPC;
    CString CC;
    CString CChex;
    CString dirint;
    CHexadecimal a2("");
    CString dir;
    CString contenido;
    char aux[5];

    a1 = ConvierteHexadecimalaEntero((const char *)m_cp);
    a1 = a1 + 1;
    _strupr( _itoa(a1,progcount,16));
    if(strlen(progcount) == 1)
      incPC = "000";
    if(strlen(progcount) == 2)
      incPC = "00";
    if(strlen(progcount) == 3)
      incPC = "0";
    incPC += progcount;
    CC = "111";
    CC += m_H;
    CC += m_I;
    CC += m_N;
    CC += m_Z;
    CC += m_C;
    CChex = ConvierteBinarioaHexadecimal((const
char*)CC);
    aux[0] = m_SP[1];
    aux[1] = m_SP[2];
    m_ciclos += 5;
    break;
  }
  case 9:{
    A = ConvierteHexadecimalaEntero(operando);
    strcpy(aux,(const char*) m_INDXh);
    B = ConvierteHexadecimalaEntero(aux);
    A = A + B;
    _strupr(_itoa(A,operando,16));
    dir = "00";
    dir += operando;
    contenido = ObtenIndice(dir);
    m_ciclos += 6;
    break;
  }
}
a1.Iniciadato((const char *)contenido);
a2.Iniciadato((const char *)m_AAh);
cmp2 = a1.binario[0];
a1 = a1-a2;
m_N = a1.binario[0];
      if((m_AAb[0]-48)   &&   cmp2   ||   cmp2   &&
(a1.binario[0]-48) || (a1.binario[0]-48) && (m_AAb[0]-48))
  m_C = '1';
else
  m_C = '0';
m_AAh = a1.hexadecimal;
m_AAb = a1.binario;
if(m_AAb == "00000000")
  m_Z = '1';
else
  m_Z = '0';

CalculaSegundos();
indice++;
}

void CxDialog::SWI(int tipins)
{
  int a1;
  int index;
  char progcount[5];
  CString incPC;
  CString CC;
  CString CChex;
  CString dirint;
  CHexadecimal a2("");
  CString dir;
  CString contenido;
  char aux[5];

  a1 = ConvierteHexadecimalaEntero((const char *)m_cp);
  a1 = a1 + 1;
  _strupr( _itoa(a1,progcount,16));
  if(strlen(progcount) == 1)
    incPC = "000";
  if(strlen(progcount) == 2)
    incPC = "00";
  if(strlen(progcount) == 3)
    incPC = "0";
  incPC += progcount;
  CC = "111";
  CC += m_H;
  CC += m_I;
  CC += m_N;
  CC += m_Z;
  CC += m_C;
  CChex = ConvierteBinarioaHexadecimal((const
char*)CC);
  aux[0] = m_SP[1];
  aux[1] = m_SP[2];
```

```
    aux[2] = '\0';
    a2.Iniciadato(aux);
    a2 = a2-1;
    dir = "00";
    dir + = a2.hexadecimal;
    contenido = ObtenIndice(dir,index);
    m_mem.SetAt(index,incPC[2]);
    m_mem.SetAt(index + 1,incPC[3]);
    aZ = a2-1;
    dir = "00";
    dir + = a2.hexadecimal;
    contenido = ObtenIndice(dir,index);
    m_mem.SetAt(index,incPC[0]);
    m_mem.SetAt(index + 1,incPC[1]);
    aZ = a2-1;
    dir = "00";
    dir + = a2.hexadecimal;
    contenido = ObtenIndice(dir,index);
    m_mem.SetAt(index,m_INDXh[0]);
    m_mem.SetAt(index + 1,m_INDXh[1]);
    aZ = a2-1;
    dir = "00";
    dir + = a2.hexadecimal;
    contenido = ObtenIndice(dir,index);
    m_mem.SetAt(index,m_AAh[0]);
    m_mem.SetAt(index + 1,m_AAh[1]);
    aZ = a2-1;
    dir = "00";
    dir + = a2.hexadecimal;
    contenido = ObtenIndice(dir,index);
    m_mem.SetAt(index,CChex[0]);
    m_mem.SetAt(index + 1,CChex[1]);
    m_I = "1";

    m_SP = "0";
    m_SP + = a2.hexadecimal;
    if(m_procesador = = "P3")
    {
       dirInt = "07F8";
       contenido = ObtenIndice(dirInt);
       dirInt = "07F9";
       contenido + = ObtenIndice(dirInt);
    }
    else
    {
       dirInt = "0FF8";
       contenido = ObtenIndice(dirInt);
       dirInt = "0FF9";
       contenido + = ObtenIndice(dirInt);
    }
    if( contenido != "0000" )
    {



        while(pInstruccion[indice].PC != contenido)
            indice--;
    }
    m_ciclos + = 11;
    CalculaSegundos();
}

void CxDialog::TAX(int tipins)
{
    m_INDXh = m_AAh;
    m_INDXb = m_AAb;
    m_ciclos + = 2;

    CalculaSegundos();
    indice + +;
}

void CxDialog::TST(int tipins)
{
    CString dir;
    CString contenido;
    CString bin;
    CString hex;
    CHexadecimal a1("");
    CHexadecimal a2("");
    char operando[5];
    int index;

    if(CO.GetLength() > = 4)
    {
       operando[0] = CO[2];
       operando[1] = CO[3];
       operando[2] = '\0';

       if(CO.GetLength() = = 6)
       {
          operando[2] = CO[4];
          operando[3] = CO[5];
          operando[4] = '\0';
       }
    }
    else
       operando[0] = '\0';

    switch(tipins)
    {
       case 2:{
                contenido = m_AAh;
                m_ciclos + = 4;
                break;
              }
       case 3:{
                contenido = m_INDXh;
                m_ciclos + = 4;
                break;
              }
       case 5:{
                dir = operando;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 6;
                break;
              }
       case 7:{
                dir = "00";
                dir + = m_INDXh;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 6;
                break;
              }
       case 8:{
                a1.Iniciadato((const char*)m_INDXh);
                a2.Iniciadato(operando);
                a1 = a1 + a2;
                dir = "00";
                dir + = a1.hexadecimal;
                contenido = ObtenIndice(dir, index);
                m_ciclos + = 7;
                break;
              }
    }

    a1.Iniciadato((const char*)contenido);
    a2.Iniciadato("00");

    a2 = a1-a2;

    m_N = a1.binario[0];
```

```
    if(a1.binario == "00000000")
      m_Z = '1';
    else
      m_Z = '0';

    CalculaSegundos();
    indice++;
}

void CxDialog::TXA(int tipins)
{
    m_AAh = m_INDXb;
    m_AAb = m_INDXb;
    m_ciclos += 2;

    CalculaSegundos();
    indice++;
}

void CxDialog::WAIT(int tipins)
{
  indice++;
MessageBox("BRN","Informacion",MB_OK);
}

void CxDialog::especial()
{
    int ope,res,n;

    char aux[3];

    if(strcmp((const char *)Nmo,"END") == 0)
      END();
    else
      if(strcmp((const char *)Nmo,"EQU") == 0)
        EQU();
      else
        if(strcmp((const char *)Nmo,"ORG") == 0)
          ORG();
        else
        {
          if(Nmo! = "")
            {
              aux[0] = CO[0];
              aux[1] = CO[1];
              aux[2] = '\0';
              ope = atoi(aux);
              res = ope % 2;
              n = ope / 2 ;
              switch(res)
              {
                case 1: BRCLR(n); break;
                case 0: BRSET(n); break;
                case 10: BSET(n); break;
                case 11: BCLR(n); break;
              }
            }
            else
              indice++;
          }
}

// Directivas
void CxDialog::DB()
{
  indice++;
}

void CxDialog::DW()
{
  indice++;
}

void CxDialog::END()
{
  if (MessageBox("Fin del programa\r\n¿Quieres volverlo
a ejecutar?","Pregunta",MB_YESNO|
      MB_ICONQUESTION) == IDYES)
    inicioDatos();
}

void CxDialog::EQU()
{
  indice++;
}

void CxDialog::ORG()
{
  indice++;
}

void CxDialog::BRCLR(int n)
{
  CString dir;
  CString dirSalto;
  char operando[3];
  char rel[3];
  char aux[5];
  CString contenido;
  CString bin;
  int a1;
  int a2;
  int pc;

  m_ciclos += 10;
  CalculaSegundos();

  operando[0] = CO[2];
  operando[1] = CO[3];
  operando[2] = '\0';
  rel[0] = CO[4];
  rel[1] = CO[5];
  rel[2] = '\0';
  dir = "00";
  dir += operando;
  contenido = ObtenIndice(dir);
                bin = ConvierteHexadecimalaBinario((const
char *)contenido);
  a1 = ConvierteHexadecimalaEntero(rel);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*)m_cp);
  m_C = bin[7-n];
  if(bin[7-n] == "0")
  {
    if(strcmp(operando,"80") >= 0)
    {
      a2 = a2-a1;
      pc = pc-a2+3;
      _strupr( _itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(p->instruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = a1+3;
      _strupr( _itoa(pc,aux,16));
```

```
        if(strlen(aux) == 1)
          dirSalto = "000";
        if(strlen(aux) == 2)
          dirSalto = "00";
        if(strlen(aux) == 3)
          dirSalto = "0";
        dirSalto += aux;
        while(pInstruccion[indice].PC != dirSalto)
          indice++;
    }
  }
  else
    indice++;
}
void CxDialog::BRSET(int n)
{
  CString dir;
  char operando[3];
  char rel[3];
  CString contenido;
  CString bin;
  int a1;
  int a2;
  int pc;
  char aux[5];
  CString dirSalto;

  m_ciclos += 10;
  CalculaSegundos();

  operando[0] = CO[2];
  operando[1] = CO[3];
  operando[2] = '\0';
  rel[0] = CO[4];
  rel[1] = CO[5];
  rel[2] = '\0';
  dir = "00";
  dir += operando;
  contenido = ObtenIndice(dir);
              bin = ConvierteHexadecimalaBinario((const
char*)contenido);
  a1 = ConvierteHexadecimalaEntero(rel);
  a2 = 256;
  pc = ConvierteHexadecimalaEntero((const char*)m_cp);
  m_C = bin[7-n];
  if(bin[7-n] != '0')
  {
    if(strcmp(operando,"80") > = 0)
    {
      a2 = a2-a1;
      pc = pc-a2+3;
      _strupr( _itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice--;
    }
    if(strcmp(operando,"7E") < 0)
    {
      pc = a1 + 3;
      _strupr( _itoa(pc,aux,16));
      if(strlen(aux) == 1)
        dirSalto = "000";
      if(strlen(aux) == 2)
        dirSalto = "00";
      if(strlen(aux) == 3)
        dirSalto = "0";
      dirSalto += aux;
      while(pInstruccion[indice].PC != dirSalto)
        indice++;
    }
  }
  else
    indice++;
}
void CxDialog::BSET(int n)
{
  CString dir;
  char operando[5];
  CString contenido;
  int index;
  CString bin;
  CString hex;

  m_ciclos += 7;
  CalculaSegundos();

  operando[0] = CO[2];
  operando[1] = CO[3];
  operando[2] = '\0';
  dir = "00";
  dir += operando;
  contenido = ObtenIndice(dir,index);
              bin = ConvierteHexadecimalaBinario((const
char*)contenido);
  bin.SetAt(7-n,'1');
  hex = ConvierteBinarioaHexadecimal((const char*)bin);
  m_mem.SetAt(index,hex[0]);
  m_mem.SetAt(index+1,hex[1]);
  indice++;

}
void CxDialog::BCLR(int n)
{
  CString dir;
  char operando[5];
  CString contenido;
  int index;
  CString bin;
  CString hex;

  m_ciclos += 7;
  CalculaSegundos();

  operando[0] = CO[2];
  operando[1] = CO[3];
  operando[2] = '\0';
  dir = "00";
  dir += operando;
  contenido = ObtenIndice(dir,index);
              bin = ConvierteHexadecimalaBinario((const
char*)contenido);
  bin.SetAt(7-n,'0');
  hex = ConvierteBinarioaHexadecimal((const char*)bin);
  m_mem.SetAt(index,hex[0]);
  m_mem.SetAt(index+1,hex[1]);
  indice++;
}

void CxDialog::EjecutaTimer()
{
  int a1;
  int index;
  char progcount[5];
  CString incPC;
  CString CC;
  CString CChex;
```

```
CString dirInt;
CHexadecimal a2("");
CString dir;
CString contenido;
char aux[5]; .

if(m_cp> = Org1)
{
a1 = ConvierteHexadecimalaEntero((const char *)m_cp);
if(CO.GetLength() = = 4)
  a1 = a1 + 2;
if(CO.GetLength() = = 6)
  a1 = a1 + 3;
  strupr( itoa(a1,progcount,16));
if(strlen(progcount) = = 1)
  incPC = "000";
if(strlen(progcount) = = 2)
  incPC = "00";
if(strlen(progcount) = = 3)
  incPC = "0";
incPC + = progcount;
CC = "111";
CCC + = m_H;
CCC + = m_I;
CCC + = m_N;
CCC + = m_Z;
CCC + = m_C;
            CChex = ConvierteBinarioaHexadecimal((const
char*)CC);
  aux[0] = m_SP[1];
  aux[1] = m_SP[2];
  aux[2] = '\0';
  a2.Iniciadato(aux);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,incPC[2]);
  m_mem.SetAt(index + 1,incPC[3]);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,incPC[0]);
  m_mem.SetAt(index + 1,incPC[1]);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,m_INDXh[0]);
  m_mem.SetAt(index + 1,m_INDXh[1]);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,m_AAh[0]);
  m_mem.SetAt(index + 1,m_AAh[1]);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,CChex[0]);
  m_mem.SetAt(index + 1,CChex[1]);
  m_I = "1";

  m_SP = "0";
  m_SP + = a2.hexadecimal;
  if(m_procesador = = "P3")
  {
    dirInt = "07F8";
    contenido = ObtenIndice(dirInt);
    dirInt = "07F9";
    contenido + = ObtenIndice(dirInt);
  }
  else
  {
    dirInt = "0FF8";
    contenido = ObtenIndice(dirInt);
    dirInt = "0FF9";
    contenido + = ObtenIndice(dirInt);
  }

  if( contenido I = "0000" )
  {
    if(pInstruccion[indice].PC < contenido)
       while(pInstruccion[indice].PC I = contenido)
          indice + +;
    if(pInstruccion[indice].PC > contenido)
       while(pInstruccion[indice].PC I = contenido)
          indice--;
  }

  m_ciclos + = 11;
  CalculaSegundos();
  }
}
void CxDialog::EjecutaIntExt()
{
  int a1;
  int index;
  char progcount[5];
  CString incPC;
  CString CC;
  CString CChex;
  CString dirInt;
  CHexadecimal a2("");
  CString dir;
  CString contenido;
  char aux[5];

  if(m_cp> = Org1)
  {
  a1 = ConvierteHexadecimalaEntero((const char *)m_cp);
  a1 = a1 + 1;
   strupr( itoa(a1,progcount,16));
  if(strlen(progcount) = = 1)
     incPC = "000";
  if(strlen(progcount) = = 2)
     incPC = "00";
  if(strlen(progcount) = = 3)
     incPC = "0";
  incPC + = progcount;
  CC = "111";
  CCC + = m_H;
  CCC + = m_I;
  CCC + = m_N;
  CCC + = m_Z;
  CCC + = m_C;
            CChex = ConvierteBinarioaHexadecimal((const
char*)CC);
  aux[0] = m_SP[1];
  aux[1] = m_SP[2];
  aux[2] = '\0';
  a2.Iniciadato(aux);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,incPC[2]);
  m_mem.SetAt(index + 1,incPC[3]);
  a2 = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
```

```
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,incPC[0]);
  m_mem.SetAt(index + 1,incPC[1]);
  aZ = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,m_INDXh[0]);
  m_mem.SetAt(index + T,m_INDXh[1]);
  aZ = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,m_AAh[0]);
  m_mem.SetAt(index + T,m_AAh[1]);
  aZ = a2-1;
  dir = "00";
  dir + = a2.hexadecimal;
  contenido = ObtenIndice(dir,index);
  m_mem.SetAt(index,CChex[0]);
  m_mem.SetAt(index + T,CChex[1]);
  m_I = "1";

  m_SP = "0";
  m_SP + = a2.hexadecimal;
  if(m_procesador = = "P3")
  {
    dirInt = "07FA";
    contenido = ObtenIndice(dirInt);
    dirInt = "07FB";
    contenido + = ObtenIndice(dirInt);
  }
  else
  {
    dirInt = "0FFA";
    contenido = ObtenIndice(dirInt);
    dirInt = "0FFB";
    contenido + = ObtenIndice(dirInt);
  }

  if( contenido ! = "0000" )
  {
    if(pInstruccion[indice].PC < contenido)
      while(pInstruccion[indice].PC ! = contenido)
        indice + + ;
    if(pInstruccion[indice].PC > contenido)
      while(pInstruccion[indice].PC ! = contenido)
        indice--;
  }
  m_ciclos + = 11;
  CalculaSegundos();
  }
}
```

```
// Programa: funcion.h
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 24 noviembre 1996
// Contiene: Programa que contiene la cabecera de las
funciones miembro de los nemonicos a ser ejecutados
// por el simulador

#include <afxwin.h>
```

```
// Programa: hexa.cpp
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 24 noviembre 1996
// Contiene: Programa que contiene la funciones miembro
de la clase Hexadecimal que permite
//  el  manejo  de  operadores  sobrecargados  para
operaciones en binario y hexadecimal

#include <afx.h>
#include "hexa.h"
#include <math.h>
#include "xdialog.h"
#include "convert.h"

CHexadecimal  CHexadecimal::operator + (CHexadecimal
&rArg)
{
  CHexadecimal total("");
  char auxbin[10] = "\0";
  int b1;
  int b2;
  int tot;
  int tambin;

  b1 = ConvierteHexadecimalaEntero(hexadecimal);
  b2 = ConvierteHexadecimalaEntero(rArg.hexadecimal);
  tot = b1 + b2;
  itoa(tot,auxbin,2);
  if(strlen(auxbin) < 8)
  {
    tambin = 8-strlen(auxbin);
    for (int i = 0 ; i<tambin ;i + + )
    {
      total.binario[i] = '0';
    }
    total.binario[tambin] = '\0';
  }
  else
  if(strlen(auxbin) = = 8)
  {
    strcpy(total.binario,"\0");
  }
  else
  {
    total.c = auxbin[0];
    tambin = strlen(auxbin);
    for (int i = 1 ; i<tambin ;i + + )
    {
      auxbin[i-1] = auxbin[i];
    }
    auxbin[tambin-1] = '\0';
    strcpy(total.binario,"\0");
  }
  strcat(total.binario,auxbin);

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
  return total;
}

CHexadecimal        CHexadecimal::operator = (const
CHexadecimal rArg)
{
  strcpy(hexadecimal,rArg.hexadecimal);
  strcpy(binario,rArg.binario);
  c = rArg.c;
  return *this;
}

CHexadecimal     CHexadecimal::operator-(CHexadecimal
rArg)
{
  CHexadecimal total("");
  char auxbin[10] = "\0";
  int b1;
  int b2;
  int tot;
```

```
int tambin;
CHexadecimal aux1("");
CHexadecimal aux2("");
CHexadecimal total1("");
CHexadecimal c("");

b1 = ConvierteHexadecimalaEntero(hexadecimal);
b2 = ConvierteHexadecimalaEntero(rArg.hexadecimal);
tot = b1-b2;
if (tot > 0)
{
   itoa(tot,auxbin,2);
   if(strlen(auxbin) < 8)
   {
      tambin = 8-strlen(auxbin);
      for (int i = 0 ; i < tambin ;i + + )
      {
         total.binario[i] = '0';
      }
      total.binario[tambin] = '\0';
   }
   else
   if(strlen(auxbin) = = 8)
   {
      strcpy(total.binario,"\0");
   }
   else
   {
      total.c = auxbin[0];
      tambin = strlen(auxbin);
      for (int i = 1 ; i < tambin ;i + + )
      {
         auxbin[i-1] = auxbin[i];
      }
      auxbin[tambin-1] = '\0';
      strcpy(total.binario,"\0");
   }
   strcat(total.binario,auxbin);

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
   return total;
}
else
{
   total.c = '1';
   aux1.Iniciadato(hexadecimal);
   aux2.Iniciadato(rArg.hexadecimal);
   c = -aux2;
   total1 = c + 1;
   aux2 = total1 + aux1;
   if(strlen(aux2.binario) < 8)
   {
      tambin = 8-strlen(aux2.binario);
      for (int i = 0 ; i < tambin ;i + + )
      {
         total.binario[i] = '0';
      }
      aux2.binario[tambin-1] = '\0';
   }
   else
   if(strlen(aux2.binario) = = 8)
   {
      strcpy(total.binario,"\0");
   }
   else
   {
      tambin = strlen(auxbin);
      for (int i = 1 ; i < tambin ;i + + )
      {
         aux2.binario[i-1] = aux2.binario[i];
      }
      strcpy(aux2.binario,"\0");
   }
   strcat(total.binario,aux2.binario);
   strcpy(total.hexadecimal,aux2.hexadecimal);
   return total;
}
}

CHexadecimal CHexadecimal::operator + (int rA)
{
   CHexadecimal total("");
   char auxbin[10] = "\0";
   int b1;
   int tot;
   int tambin;

   b1 = ConvierteHexadecimalaEntero(hexadecimal);
   tot = b1 + rA;
   itoa(tot,auxbin,2);
   if(strlen(auxbin) < 8)
   {
      tambin = 8-strlen(auxbin);
      for (int i = 0 ; i < tambin ;i + + )
      {
         total.binario[i] = '0';
      }
      total.binario[tambin] = '\0';
   }
   else
   if(strlen(auxbin) = = 8)
   {
      strcpy(total.binario,"\0");
   }
   else
   {
      total.c = auxbin[0];
      tambin = strlen(auxbin);
      for (int i = 1 ; i < tambin ;i + + )
      {
         auxbin[i-1] = auxbin[i];
      }
      auxbin[tambin-1] = '\0';
      strcpy(total.binario,"\0");
   }
   strcat(total.binario,auxbin);

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
   return total;
}

CHexadecimal CHexadecimal::operator-(int rA)
{
   CHexadecimal total("");
   char auxbin[10] = "\0";
   int b1;
   int tot;
   int tambin;
   CHexadecimal aux1("");
   CHexadecimal aux2("");
   CHexadecimal total1("");
   CHexadecimal c("");

   b1 = ConvierteHexadecimalaEntero(hexadecimal);
   tot = b1-rA;
   if (tot > 0)
   {
      itoa(tot,auxbin,2);
      if(strlen(auxbin) < 8)
      {
         tambin = 8-strlen(auxbin);
         for (int i = 0 ; i < tambin ;i + + )
```

```
    {
      total.binario[i] = '0';
    }
    total.binario[tambin] = '\0';
  }
  else
  if(strlen(auxbin) == 8)
  {
    strcpy(total.binario,"\0");
  }
  else
  {
    total.c = auxbin[0];
    tambin = strlen(auxbin);
    for (int i = 1 ; i<tambin ;i++)
    {
      auxbin[i-1] = auxbin[i];
    }
    auxbin[tambin-1] = '\0';
    strcpy(total.binario,"\0");
  }
  strcat(total.binario,auxbin);

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
    return total;
  }
  return total;
}

CHexadecimal CHexadecimal::operator&&(CHexadecimal
rArg)
{
  CHexadecimal total("");
  CString bin;
  CString rA;

  bin = ConvierteHexadecimalaBinario(hexadecimal);
  rA = ConvierteHexadecimalaBinario(rArg.hexadecimal);

  for(int i = 0;i<8; i++)
  {
    if(bin[i] == '1' && rA[i] == '1')
      total.binario[i] = '1';
    else
      total.binario[i] = '0';
  }
  total.binario[8] = '\0';

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
  total.c = '0';
  return total;
}

CHexadecimal  CHexadecimal::operator||(CHexadecimal
rArg)
{
  CHexadecimal total("");
  CString bin;
  CString rA;

  bin = ConvierteHexadecimalaBinario(hexadecimal);
  rA = ConvierteHexadecimalaBinario(rArg.hexadecimal);

  for(int i = 0;i<8; i++)
  {
    if(bin[i] == '0' && rA[i] == '0')
      total.binario[i] = '0';
    else
      total.binario[i] = '1';
  }
  total.binario[8] = '\0';

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
  total.c = '0';
  return total;
}

CHexadecimal  CHexadecimal::operator^(CHexadecimal
rArg)
{
  CHexadecimal total("");
  CString bin;
  CString rA;

  bin = ConvierteHexadecimalaBinario(hexadecimal);
  rA = ConvierteHexadecimalaBinario(rArg.hexadecimal);

  for(int i = 0;i<8; i++)
  {
    if(bin[i] == rA[i])
      total.binario[i] = '0';
    else
      total.binario[i] = '1';
  }
  total.binario[8] = '\0';

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
  total.c = '0';
  return total;
}

CHexadecimal CHexadecimal::operator~(void)
{
  CString aux;
  CHexadecimal total;

  aux = ConvierteHexadecimalaBinario(hexadecimal);

  for(int i = 0;i<8; i++)
  {
    if(aux[i] == '1')
      total.binario[i] = '0';
    else
      total.binario[i] = '1';
  }
  total.binario[8] = '\0';

strcpy(total.hexadecimal,ConvierteBinarioaHexadecimal(t
otal.binario));
  total.c = '0';
  return total;
}
```

```
// Programa: hexa.h
// Elaborado por: Graciela Santillan Alcala

#include "conver.h"

class CHexadecimal: public CObject
{

public:
  char binario[9];
  char hexadecimal[3];
  char c;

  CHexadecimal();
```

```
CHexadecimal(const char *hex)
{
  CString aux;
  strcpy(hexadecimal,hex);
  aux = ConvierteHexadecimalaBinario(hexadecimal);
  strcpy(binario,(const char*)aux);
  c = '0';
};

CHexadecimal(CString hex)
{
  CString aux;
  strcpy(hexadecimal,(const char*) hex);
  aux = ConvierteHexadecimalaBinario(hexadecimal);
  strcpy(binario,(const char*)aux);
  c = '0';
};

CHexadecimal(CHexadecimal &b)
{
  CString aux;
  strcpy(binario,b.binario);
  aux = ConvierteBinarioaHexadecimal(binario);
  strcpy(hexadecimal,(const char*)aux);
  c = b.c;
}

void Iniciadato(const char *hex)
{
  CString aux;
  strcpy(hexadecimal,hex);
  aux = ConvierteHexadecimalaBinario(hexadecimal);
  strcpy(binario,(const char*)aux);
  c = '0';
}

CHexadecimal& operator = (const CHexadecimal &rArg);
CHexadecimal operator + (CHexadecimal &rArg);
CHexadecimal operator + (int rA);
CHexadecimal operator-(int rA);
CHexadecimal operator-(CHexadecimal &rArg);
CHexadecimal operator&&(CHexadecimal &rArg);
CHexadecimal operator|| (CHexadecimal &rArg);
CHexadecimal operator^(CHexadecimal &rArg);
CHexadecimal operator ~ (void);
};
```

```
// Programa: INSTRU.H
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 11 noviembre 1996
// Contiene: Programa que contiene la declaracion de la
clase CInstruccion

class CInstruccion //: public CObject
{
//  DECLARE_SERIAL(CInstruccion)
public:
  CString PC;
  CString CodHex;
  CString Etiqueta;
  CString Nmonico;
  CString Operando;
  BOOL BEtiqueta;

  CInstruccion()
  {
    PC = "";
    CodHex = "";
    Etiqueta = "";
    Nmonico = "";
    Operando = "";
    BEtiqueta = FALSE;
  }

  BOOL Inicialnstruccion(CString pc, CString CH, CString
E, CString Nmo, CString Oprn, BOOL BE)
  {
    PC = pc;
    CodHex = CH;
    Etiqueta = E;
    Nmonico = Nmo;
    Operando = Oprn;
    BEtiqueta = BE;
    return TRUE;
  }
};
```

```
// Programa: SIM6805.CPP
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 7 noviembre 1996
// Contiene: Programa principal para la creacion y manejo
del simulador de los MCU's 68705P3,
// R3 y U3

#include <afxwin.h>
#include "sim6805.h"
#include "resource.h"
#include "xdialog.h"

class CWindowApp: public CWinApp
{
public:

  virtual BOOL InitInstance();
};

// expande la funcionalidad de CFrameWnd derivando la
clase CAppWindow
class CAppWindow: public CFrameWnd
{

public:

  CAppWindow();
  ~ CAppWindow();

protected:

  // declara la macro de mapeo de mensaje
  DECLARE_MESSAGE_MAP();
};

CAppWindow::CAppWindow()
{
  // carga el recurso de teclas aceleradoras
  LoadAccelTable("BUTTONS");
  // crea la ventana
  Create(NULL, "Simulador de los MCUs 68705P3, R3 y
U3",
  WS_OVERLAPPEDWINDOW | WS_MINIMIZE, rectDefault,
NULL, "MAINMENU");
};

CAppWindow::~CAppWindow()
{
}

BEGIN_MESSAGE_MAP(CAppWindow,CFrameWnd)
```

```
END_MESSAGE_MAP()

// construye el dato miembro m_pMaindWnd de CWindowApp
BOOL CWindowApp::InitInstance()
{
   CxDialog pDlg1;

   SetDialogBkColor(RGB(192,192,192));
   pDlg1.DoModal();
   return TRUE;
}

// el constructor de la aplicaci¾n inicializa y ejecuta la aplicaci¾n
CWindowApp NEAR WindowApp;

// Programa: xdialog.CPP
// Elaborado por: Graciela Santillan Alcala
// Fecha de ultima correccion: 22 diciembre 1996
// Contiene: Programa que contiene las funciones miembro de las clases definidas en
// xdialog.h
//

//#include "stdafx.h"
//#include "..h"
#include <afxwin.h>
#include "xdialog.h"
#include "sim8085.h"
#include <string.h>
#include "funcion.h"
#include <math.h>
#include "conver.h"
#include "hexa.h"
#include <stdio.h>


////////////////////////////////////////////////////////////////////////////
// CxDialog dialog

CxDialog::CxDialog() //CxDialog::CxDialog(CWnd* pParent /*=NULL*/)
        : CDialog(CxDialog::IDD, pParent)
CDialog(CxDialog::IDD, pParent)
{
        //{{AFX_DATA_INIT(CxDialog)
        m_archivo = "";
        m_C = "0";
        m_ciclos = 0;
        m_cp = "0000";
        m_DDRAb = "00000000";
        m_AAb = "00000000";
        m_AAh = "00";
        m_DDRAh = "00";
        m_DDRBb = "00000000";
        m_DDRBh = "00";
        m_DDRCb = "00000000";
        m_DDRCh = "00";
        m_directorio = "";
        m_H = "0";
        m_I = "1";
        m_INDXb = "00000000";
        m_INDXh = "00";
        m_N = "0";
        m_lineaprog = "";
        m_PORTAb = "00000000";
        m_PORTAh = "00";
        m_PORTBb = "00000000";
        m_PORTBh = "00";
        m_PORTCb = "00000000";
        m_PORTCh = "00";
        m_PORTDb = "00000000";
        m_PORTDh = "00";
        m_procesador = "";
        m_SP = "07FF";
        m_TCAh = "40";
        m_TCAb = "01000000";
        m_TDRb = "11111111";
        m_TDRh = "FF";
        m_Z = "0";
        m_pins = "";
        m_mem = "";
        m_Programa = "";
        m_segundos = "";
        m_free = "";
        m_timer = "0";
        m_INT = "";
        //}}AFX_DATA_INIT
        segundos = 0.0;
        m_ICount = 0L;
        m_IMaxCount = 1500L;
}

void CxDialog::IniciaMemoria()
{
   char memoria[5];
   char memaux[5];
   int bloqmem = 0;
   int cmp1;

   m_mem = "";

   if(m_procesador = = "P3")
      bloqmem = 128;
   else
      bloqmem = 256;
   for (int i = 0; i < bloqmem; i + +)
   {
      strupr( itoa(i*16,memoria,16));
      if (strlen(memoria) = = 1)
         strcpy(memaux,"000");
      else
         if (strlen(memoria) = = 2)
            strcpy(memaux,"00");
         else
            if (strlen(memoria) = = 3)
               strcpy(memaux,"0");
      strcat(memaux,memoria);
      cmp1 = strcmp(memaux,"0080");
      /*cmp2 = strcmp(memaux,"0F20");
      cmp3 = strcmp(memaux,"0770");*/
      if(cmp1 < 0 /*|| (cmp2 > 0 && bloqmem = = 256) ||
(cmp3 > 0 && bloqmem = = 128)*/)
      {
         m_mem + = memaux;
         m_mem + = ": 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00\r\n";
      }
   }

}

BOOL CxDialog::OnInitDialog()
{
 return TRUE;
}

void CxDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CxDialog)
```

```
        DDX_Control(pDX,        IDC_PASOAPASO,
m_pasoapaso);
        DDX_Control(pDX, IDC_SIMULA, m_simula);
        DDX_Control(pDX, IDC_RESET, m_reset);
        DDX_Text(pDX, IDC_ARCHIVO, m_archivo);
        DDV_MaxChars(pDX, m_archivo, 12);
        DDX_Text(pDX, IDC_C, m_C);
        DDV_MaxChars(pDX, m_C, 1);
        DDX_Text(pDX, IDC_CICLOS, m_ciclos);
        DDV_MinMaxLong(pDX, m_ciclos, 0, 1000000);
        DDX_Text(pDX, IDC_CP, m_cp);
        DDV_MaxChars(pDX, m_cp, 8);
        DDX_Text(pDX, IDC_DDRAB, m_DDRAb);
        DDV_MaxChars(pDX, m_DDRAb, 8);
        DDX_Text(pDX, IDC_ACUAB, m_AAb);
        DDV_MaxChars(pDX, m_AAb, 8);
        DDX_Text(pDX, IDC_ACUAH, m_AAh);
        DDV_MaxChars(pDX, m_AAh, 2);
        DDX_Text(pDX, IDC_DDRAH, m_DDRAh);
        DDV_MaxChars(pDX, m_DDRAh, 2);
        DDX_Text(pDX, IDC_DDRBB, m_DDRBb);
        DDV_MaxChars(pDX, m_DDRBb, 8);
        DDX_Text(pDX, IDC_DDRBH, m_DDRBh);
        DDV_MaxChars(pDX, m_DDRBh, 2);
        DDX_Text(pDX, IDC_DDRCB, m_DDRCb);
        DDV_MaxChars(pDX, m_DDRCb, 8);
        DDX_Text(pDX, IDC_DDRCH, m_DDRCh);
        DDV_MaxChars(pDX, m_DDRCh, 2);
        DDX_Text(pDX, IDC_DIRECTORIO, m_directorio);
        DDV_MaxChars(pDX, m_directorio, 30);
        DDX_Text(pDX, IDC_H, m_H);
        DDV_MaxChars(pDX, m_H, 1);
        DDX_Text(pDX, IDC_I, m_I);
        DDV_MaxChars(pDX, m_I, 1);
        DDX_Text(pDX, IDC_INDXB, m_INDXb);
        DDV_MaxChars(pDX, m_INDXb, 8);
        DDX_Text(pDX, IDC_INDXH, m_INDXh);
        DDV_MaxChars(pDX, m_INDXh, 2);
        DDX_Text(pDX, IDC_N, m_N);
        DDV_MaxChars(pDX, m_N, 1);
        DDX_Text(pDX, IDC_LINEAPROG, m_lineaprog);
        DDV_MaxChars(pDX, m_lineaprog, 70);
        DDX_Text(pDX, IDC_PORTAB, m_PORTAb);
        DDV_MaxChars(pDX, m_PORTAb, 8);
        DDX_Text(pDX, IDC_PORTAH, m_PORTAh);
        DDV_MaxChars(pDX, m_PORTAh, 2);
        DDX_Text(pDX, IDC_PORTBB, m_PORTBb);
        DDV_MaxChars(pDX, m_PORTBb, 8);
        DDX_Text(pDX, IDC_PORTBH, m_PORTBh);
        DDV_MaxChars(pDX, m_PORTBh, 2);
        DDX_Text(pDX, IDC_PORTCB, m_PORTCb);
        DDV_MaxChars(pDX, m_PORTCb, 8);
        DDX_Text(pDX, IDC_PORTCH, m_PORTCh);
        DDV_MaxChars(pDX, m_PORTCh, 2);
        DDX_Text(pDX, IDC_PORTDB, m_PORTDb);
        DDV_MaxChars(pDX, m_PORTDb, 8);
        DDX_Text(pDX, IDC_PORTDH, m_PORTDh);
        DDV_MaxChars(pDX, m_PORTDh, 2);
        DDX_Text(pDX, IDC_PROC, m_procesador);
        DDV_MaxChars(pDX, m_procesador, 2);
        DDX_Text(pDX, IDC_SP, m_SP);
        DDV_MaxChars(pDX, m_SP, 8);
        DDX_Text(pDX, IDC_TCRH, m_TCRh);
        DDV_MaxChars(pDX, m_TCRh, 2);
        DDX_Text(pDX, IDC_TCRB, m_TCRb);
        DDV_MaxChars(pDX, m_TCRb, 8);
        DDX_Text(pDX, IDC_TDRB, m_TDRb);
        DDV_MaxChars(pDX, m_TDRb, 8);
        DDX_Text(pDX, IDC_TDRH, m_TDRh);
        DDV_MaxChars(pDX, m_TDRh, 2);
        DDX_Text(pDX, IDC_Z, m_Z);
        DDV_MaxChars(pDX, m_Z, 1);
        DDX_Text(pDX, IDC_PINS, m_pins);
        DDV_MinMaxUInt(pDX, m_pins, 0, 50);
        DDX_Text(pDX, IDC_MEM, m_mem);
        DDX_Text(pDX, IDC_PROGRAMA, m_programa);
        DDX_Text(pDX, IDC_SEG, m_segundos);
        DDX_Text(pDX, IDC_FREC, m_frec);
        DDX_Text(pDX, IDC_TIMER, m_timer);
        DDV_MaxChars(pDX, m_timer, 1);
        DDX_Text(pDX, IDC_INT, m_INT);
        DDV_MaxChars(pDX, m_INT, 1);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CxDialog, CDialog)
        //{{AFX_MSG_MAP(CxDialog)
        ON_COMMAND(IDM_ABOUT,CMAbout)
        ON_COMMAND(IDM_DIALOG,CMDialog)
        ON_COMMAND(IDM_EXIT,OnExit)
        ON_WM_CLOSE()
        ON_BN_CLICKED(IDC_SALIR, OnClickedSalir)
        ON_BN_CLICKED(IDC_SIMULA,OnClickedSimula)
        ON_BN_CLICKED(IDC_RESET, OnClickedReset)
        ON_BN_CLICKED(IDC_PASOAPASO,
OnClickedPasoapaso)
        ON_EN_CHANGE(IDC_MEM, OnChangeMem)
        ON_EN_CHANGE(IDC_DDRAB,OnChangeDDRAb)
        ON_EN_CHANGE(IDC_DDRAH,OnChangeDDRAh)
        ON_EN_CHANGE(IDC_DDRBB, OnChangeDDRBb)
        ON_EN_CHANGE(IDC_DDRBH, OnChangeDDRBh)
        ON_EN_CHANGE(IDC_DDRCB, OnChangeDDRCb)
        ON_EN_CHANGE(IDC_DDRCH, OnChangeDDRCh)
        ON_EN_CHANGE(IDC_PORTAB,
OnChangePORTAb)
        ON_EN_CHANGE(IDC_PORTAH,
OnChangePORTAh)
        ON_EN_CHANGE(IDC_PORTBB,
OnChangePORTBb)
        ON_EN_CHANGE(IDC_PORTBH,
OnChangePORTBh)
        ON_EN_CHANGE(IDC_PORTCB,
OnChangePORTCb)
        ON_EN_CHANGE(IDC_PORTCH,
OnChangePORTCh)
        ON_EN_CHANGE(IDC_PORTDB,
OnChangePORTDb)
        ON_EN_CHANGE(IDC_PORTDH,
OnChangePORTDh)
        ON_EN_CHANGE(IDC_ACUAB, OnChangeAcuAb)
        ON_EN_CHANGE(IDC_ACUAH, OnChangeAcuAh)
        ON_EN_CHANGE(IDC_INDXB, OnChangeIndxb)
        ON_EN_CHANGE(IDC_INDXH, OnChangeIndxh)
        ON_WM_ABAUTTON(UP)
        ON_EN_CHANGE(IDC_CP, OnChangeCp)
        ON_EN_CHANGE(IDC_C, OnChangeC)
        ON_EN_CHANGE(IDC_H, OnChangeH)
        ON_EN_CHANGE(IDC_I, OnChangeI)
        ON_EN_CHANGE(IDC_N, OnChangeN)
        ON_EN_CHANGE(IDC_Z, OnChangeZ)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_PARA, OnClickedPara)
        ON_COMMAND(IDC_SALIR,OnClickedSalir)
        ON_COMMAND(IDC_SIMULA,OnClickedSimula)
        ON_COMMAND(IDC_RESET,OnClickedReset)
        ON_COMMAND(IDC_PASOAPASO,
OnClickedPasoapaso)
        ON_EN_CHANGE(IDC_TIMER, OnChangeTimer)
        ON_EN_CHANGE(IDC_INT, OnChangeInt)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////
```

```
// CxDialog message handlers

void CxDialog::CMAbout()
{
  CAboutDlg aboutDlg;
  aboutDlg.DoModal();
}

void CxDialog::CMDialog()
{
  char archivo[31];
  indice = 0;
  BOOL bactivo;
  int tam;
  int cmp1;
  CxResDlg dialogDlg;

  dialogDlg.DoModal();
  fval = 0.0;
  m_archivo = dialogDlg.m_arch;
  m_directorio = dialogDlg.m_direc;
  if !dialogDlg.m_radio = = 0)
  {
    m_procesador = "P3";
    m_pins = 28;
  }
  if (dialogDlg.m_radio = = 1)
  {
    m_procesador = "R3";
    m_pins = 40;
  }
  if (dialogDlg.m_radio = = 2)
  {
    m_procesador = "U3";
    m_pins = 40;
  }
  strcpy(archivo,(const char *)m_directorio);
  tam = strlen(archivo);
  if(archivo[tam] != '\\' && m_directorio != "c:\\")
    strcat(archivo,"\\");
  strcat(archivo,(const char *)m_archivo);
  cmp1 = strcmp(archivo,"c:\\*.prn");
  if(cmp1 != 0)
  {
    m_mem = "";
    IniciaMemoria();
    IniciaDatos();
    m_programa = "";
    bactivo = AbreArchivo(archivo);
    if(m_procesador! = "P3")
    {
      CambiaDirMORelNT();
    }
    ObtenIval();
  }
}

void CxDialog::CambiaDirMORelNT()
{
  int index;
  CString contenido;
  CString MOR;

  //obtengo lo que tenia el MOR
  MOR = ObtenIndice("0784",index);
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'0');
  contenido = ObtenIndice("07F0", index);
  index = index-6;
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'F');
  m_mem.SetAt(index + 2,'F');
  m_mem.SetAt(index + 3,'0');
  contenido = ObtenIndice("0780", index);
  index = index-6;
  m_mem.SetAt(index,'0');
  m_mem.SetAt(index + 1,'F');
  m_mem.SetAt(index + 2,'3');
  m_mem.SetAt(index + 3,'0');
  contenido = ObtenIndice("0F38",index);
  m_mem.SetAt(index,MOR[0]);
  m_mem.SetAt(index + 1,MOR[1]);
}

void CxDialog::ObtenIval()
{
  CString contenido;
  CString aux;
  CHexadecimal a("");
  double val;
  char periodo[20];
  unsigned long frec;

  if(m_procesador = = "P3")
    contenido = ObtenIndice("0784");
  else
    contenido = ObtenIndice("0F38");
  a.Iniciadato((const char *)contenido);
  if(a.binario[0] = = '0')
  {
    fval = 0.00000025;
    m_frec = "4000000 hz";
  }
  else
  {
    fval = 0.0000001;
    m_frec = "10000000 hz";
  }
  if(a.binario[2] = = 1)
  {
  }
  else
  {
  }
  aux = a.binario[5];
  aux + = a.binario[6];
  aux + = a.binario[7];
  if (aux = = "000")
  {
    fval = fval/1.0;
  }
  if (aux = = "001")
  {
    fval = fval/2.0;
  }
  if (aux = = "010")
  {
    fval = fval/4.0;
  }
  if (aux = = "011")
  {
    fval = fval/8.0;
  }
  if (aux = = "100")
  {
    fval = fval/16.0;
  }
  if (aux = = "101")
  {
    fval = fval/32.0;
  }
  if (aux = = "110")
  {
    fval = fval/64.0;
```

```
    }
    if (aux = ="111")
    {
        fval = fval/128.0;
    }
    val = (1/fval)/1000000;
    frec = (long)val;
    itoa(frec,periodo,10);
    m_frec = periodo;
    m_frec += " Mhz";
}

void CxDialog::OnClose()
{
        if (MessageBox("¿Quieres cerrar la aplicación?","Pregunta",MB_YESNO|
    MB_ICONQUESTION) = = IDYES)
    DestroyWindow();
}

void CxDialog::OnExit()
{ SendMessage(WM_CLOSE); }

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //
    //
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
                // NOTE: the ClassWizard will add DDX
and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // NOTE: the ClassWizard will add
message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg message handlers

/////////////////////////////////////////////////////////////////////////////
// CxResDlg dialog

CxResDlg::CxResDlg(){ //CWnd* pParent /* = NULL*/
    : CDialog(CxResDlg::IDD) //, pParent
{
        //{{AFX_DATA_INIT(CxResDlg)
        m_arch = "*.pin";
        m_direc = "c:\\";
        m_radio = 0;
        //}}AFX_DATA_INIT
}

void CxResDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CxResDlg)
        DDX_Text(pDX, ID_ARCHIVO_EDIT, m_arch);
        DDV_MaxChars(pDX, m_arch, 12);
        DDX_Text(pDX, ID_DIRECTORIO_EDIT, m_direc);
        DDV_MaxChars(pDX, m_direc, 30);
        DDX_Radio(pDX, ID_P3_RBT, m_radio);
        //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CxResDlg, CDialog)
        //{{AFX_MSG_MAP(CxResDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CxResDlg message handlers

void CxResDlg::OnCancel()
{
        // TODO: Add extra cleanup here
        //CDialog::OnCancel();
        if (MessageBox("¿Quieres cerrar la
ventana?","Pregunta",MB_YESNO|MB_ICONQUESTION)
 = = IDYES)
        DestroyWindow();
}

void CxResDlg::OnOK()
{
    CDialog::OnOK();
}

void CxDialog::OnClickedSalir()
{
        // TODO: Add your control notification handler
code here
        OnClose();
}

void CxDialog::ObtenSubcadena(char *C)
{

    int i = 0;
    int j = 0;
    int tam;

    char operando[10];
    char auxcad[70];
    char auxcad2[10];

    strcpy(auxcad,C);
    tam = strlen(auxcad);

    strcpy(operando,"\0");
    strcpy(auxcad2,"\0");
    while( i< tam && auxcad[i]!= 10 && auxcad[i]!= 13 &&
auxcad[i]!= 59)
    {
        int k = 0;
        while(auxcad[i] != ' ' && i<tam && auxcad[i] != 10
)
        {
            operando[k] = auxcad[i];
            k++;
            i++;
        }
        if( k!= 0)
        {
            strncat(auxcad2,operando,k);
            arreglo[j] = auxcad2;
            j++;
        }
        if ((i = = 6 || i = = 15) && auxcad[i] = = ' ')
        { i++; }
        strcpy(operando,"\0");
        strcpy(auxcad2,"\0");
```

```
    i++;
  }
}

BOOL CxDialog::AbreArchivo(const char* archivo)
{
  FILE *f;
  //int count = 0;
  int cont = 0;
  char buffer[70];
  CString s;
  BOOL Etiqueta;
  char scont[4];

  for (int i=0;i<70;i++)
    buffer[i] = ' ';

  strcpy(buffer,"\0");

  for(i=0 ; i<5;i++)
    arreglo[i] = "\0";

  EsOrg = TRUE;
  EsOrg1 = TRUE;

  count = 0;
  if( (f = fopen(archivo, "r" )) != NULL )
  {
    while(fgets(buffer,70,f)!=NULL || !feof(f))
    {
          if(buffer[1] >= '0' && buffer[1] <= '9' &&
buffer[0]!= 10 || buffer[1] == ' ')
        {
          ObtenSubcadena(buffer);
          itoa(cont+1,scont,10);
          s = scont;
          if (strlen(scont) == 1)
                s += " ";
              else
                s += " ";
          s += arreglo[2];
          if (arreglo[2] == "" || arreglo[2].GetLength() == 2
|| arreglo[2].GetLength() == 3)
                s += "\t\t";
              else
                s += "\t";
          s += arreglo[3];
          s += "\t";
          s += arreglo[4];
          m_programa += s;
          m_programa += "\r\n";
          if !strlen(arreglo[2])!= '0')
            Etiqueta = TRUE;
          else
            Etiqueta = FALSE;

plnstruccion[cont],IniciaInstruccion(arreglo[0],arreglo[1],
arreglo[2],arreglo[3],arreglo[4],Etiqueta);
          if(buffer[1] != ' ')
          {
            if(arreglo[3] == "ORG" )
            {
              Org = arreglo[0];
            }
            if(arreglo[3] == "ORG" && EsOrg1)
            {
              Org = arreglo[0];
              EsOrg1 = FALSE;
            }
            CargaProgramaAMemoria(arreglo[0],arreglo[1]);
            count++;
          }
          for (i=0; i<5;i++)
            arreglo[i] = "\0";
          cont++;
        }
      }
      fclose(f);
      return TRUE;
  }
  else
  {
      m_archivo = "";
      m_directorio = "";
      m_procesador = "";
                  MessageBox("No se pudo abrir el
archivo","Information",MB_OK);
      IniciaDatos();
      return FALSE;
  }
}


void CxDialog::CargaProgramaAMemoria(CString a0,
CString a1)
{
  int index;
  int indexaux;
  CString memaux;
  char aux2[5];

  if(!(a1 == ""))
  {
    index = BuscaDireccionMemoria(a0);
    if(a1.GetLength() >= 2)
    {
      m_mem.SetAt(index,a1[0]);
      m_mem.SetAt(index+1,a1[1]);
    }

    if(a1.GetLength() >= 4)
    {
      if(m_mem.GetAt(index+2) == '\r')
      {
        index = index+4;
        aux2[0] = a0[0];
        aux2[1] = a0[1];
        aux2[2] = a0[2];
        aux2[3] = 0;
        aux2[4] = '\0';
        indexaux = ConvierteHexadecimalaEntero(aux2);
        indexaux = indexaux+16;
        strupr( itoa(indexaux,aux2,16));
        if (strlen(aux2) == 1)
         memaux = "000";
        else
          if (strlen(aux2) == 2)
            memaux = "00";
          else
            if (strlen(aux2) == 3)
              memaux = "0";
        memaux += aux2;
        m_mem += memaux;
        m_mem += ": 00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00\r\n";
        aux2[0] = m_mem[index];
        aux2[1] = m_mem[index+1];
        aux2[2] = m_mem[index+2];
        aux2[3] = m_mem[index+3];

        indexant = index;
        index = index+6;
        m_mem.SetAt(index,a1[2]);
```

```
            m_mem.SetAt(index + 1,a1[3]);
            aux2[0] = m_mem[index];
            aux2[1] = m_mem[index + 1];
            aux2[2] = m_mem[index + 2];
            aux2[3] = m_mem[index + 3];

          }
          else
          {
            index = index + 5;
            m_mem.SetAt(index,a1[2]);
            m_mem.SetAt(index + 1,a1[3]);
            aux2[0] = m_mem[index];
            aux2[1] = m_mem[index + 1];
            aux2[2] = m_mem[index + 2];
            aux2[3] = m_mem[index + 3];

          }
        }
        if(a1.GetLength() == 6)
        {
          if(m_mem[index + 2] == '\r')
          {
            index = index + 4;
            aux2[0] = a0[0];
            aux2[1] = a0[1];
            aux2[2] = a0[2];
            aux2[3] = '0';
            aux2[4] = '\0';
            indexaux = ConvierteHexadecimalaEntero(aux2);
            indexaux = indexaux + 16;
            _itoa( indexaux,aux2,16);
           if (strlen(aux2) == 1)
            memaux = "000";
            else
              if (strlen(aux2) == 2)
              memaux = "00";
              else
                if (strlen(aux2) == 3)
                memaux = "0";
            memaux += aux2;
            m_mem += memaux;
             m_mem+ = ": 00   00   00   00   00   00   00
00   00   00   00   00   00   00   00\r\n";
            aux2[0] = m_mem[index];
            aux2[1] = m_mem[index + 1];
            aux2[2] = m_mem[index + 2];
            aux2[3] = m_mem[index + 3];
            indexant = index;
            index = index + 6;
            m_mem.SetAt(index,a1[4]);
            m_mem.SetAt(index + 1,a1[5]);
            aux2[0] = m_mem[index];
            aux2[1] = m_mem[index + 1];
            aux2[2] = m_mem[index + 2];
            aux2[3] = m_mem[index + 3];
          }
          else
          {
            index = index + 5;
            m_mem.SetAt(index,a1[4]);
            m_mem.SetAt(index + 1,a1[5]);
            aux2[0] = m_mem[index];
            aux2[1] = m_mem[index + 1];
            aux2[2] = m_mem[index + 2];
            aux2[3] = m_mem[index + 3];
          }
        }
      }
    }
    void CxDialog::OnClickedSimula()
    {
          CString c1("0000");
          CString s1;
          char sindice[4];
          MSG Message;

        m_simula.EnableWindow(FALSE);
       UpdateData(TRUE);

         if (m_archivo != "")
         {

    for(m_lCount = 0L;m_lCount < m_lMaxCount;m_lCount + + )
        {
                PC = pInstruccion[indice].PC;
                CO = pInstruccion[indice].CodHex;
                E = pInstruccion[indice].Etiqueta;
                Nmo = pInstruccion[indice].Nmonico;
                Op = pInstruccion[indice].Operando;
                Et = pInstruccion[indice].BEtiqueta;
                itoa(indice + 1,sindice,10);
          if (PC[1] == ' ')
              {
                if (PC == " " + " ")
                {
                  s1 = sindice;
                  if (strlen(sindice) == 1)
                s1 + = ":  ";
                  else
                    s1 + = ": ";
                  s1 + = E;
                    if (E == "" || E.GetLength() == 2 ||
E.GetLength() == 3)
                      s1 + = "\t\t";
                    else
                s1 + = "\t";
                  s1 + = Nmo;
                  s1 + = "\t";
                  s1 + = Op;
                  m_lineaprog = s1;
                }
                else
                {
                  s1 = sindice;
                  if (strlen(sindice) == 1)
                    s1 + = ":  ";
                  else
                    s1 + = ": ";
                  s1 + = E;
                    if (E == "" || E.GetLength() == 2 ||
E.GetLength() == 3)
                s1 = "\t\t";
                    else
                      s1 + = "\t";
                    s1 + = Nmo;
                    s1 + = "\t";
                    s1 + = Op;
                    m_lineaprog = s1;
                    m_cp = PC;
              }
                }
                EjecutaInstruccion();
    }
              m_simula.EnableWindow(TRUE);
          }
    }
    void CxDialog::OnClickedReset()
    {
      char archivo[30];
            // TODO: Add your control notification handler
```

```
code here
        if (m_archivo != "")
        {
          m_mem = "";
          m_programa = "";
          IniBalMemoria();
          IniciaDatos(1);
          strcpy(archivo,m_directorio);
          strcat(archivo,"\\");
          strcat(archivo,m_archivo);
          AbreArchivo(archivo);
          UpdateData(FALSE);
        }
}

void CxDialog::notImplemented()
{
        MessageBox("This feature is not
implemented","Information",MB_OK);
}

void CxDialog::IniciaDatos()
{
    char auxmem[3];

        m_C = "0";
        m_ciclos = 0;
        m_cp = "0000";
        m_DDRAb = "00000000";
        m_AAb = "00000000";
        m_AAh = "00";
        m_DDRAh = "00";
        m_DDRBb = "00000000";
        m_DDRBh = "00";
        m_DDRCb = "00000000";
        m_DDRCh = "00";
        m_H = "0";
        m_I = "1";
        m_INDXb = "00000000";
        m_INDXh = "00";
        m_N = "0";
        m_lineaprog = "";
        m_PORTAb = "00000000";
        m_PORTAh = "00";
        m_PORTBb = "00000000";
        m_PORTBh = "00";
        m_PORTCb = "00000000";
        m_PORTCh = "00";
        m_PORTDb = "00000000";
        m_PORTDh = "00";
        m_SP = "07F";
        m_TCRh = "40";
        m_TCRb = "01000000";
        m_TDRb = "11111111";
        m_TDRh = "FF";
        m_Z = "0";
        m_INT = "0";
        m_timer = "0";
        m_segundos = "";
        segundos = 0.0;
        indice = 0;
        bandera = 0;

        strcpy(auxmem,(const char*)m_TDRh);
        m_mem.SetAt(49,auxmem[0]);
        m_mem.SetAt(47,auxmem[1]);

        strcpy(auxmem,(const char*)m_TCRh);
        m_mem.SetAt(51,auxmem[0]);
        m_mem.SetAt(52,auxmem[1]);
}
void CxDialog::OnClickedPasoapaso()
{
        // TODO: Add your control notification handler
code here
        // TODO: Add your control notification
handler code here
    CString s1;
        char sindice[4];

        if (m_archivo != "")
        {
    UpdateData(TRUE);
          PC = pInstruccion[indice].PC;
          CO = pInstruccion[indice].CodHex;
          E = pInstruccion[indice].Etiqueta;
          Nmo = pInstruccion[indice].Nmonico;
          Op = pInstruccion[indice].Operando;
          Et = pInstruccion[indice].BEtiqueta;
          m_lineaprog = "";
          _itoa(indice + 1,sindice,10);

      if (PC != "")
          {
            if (PC == "0")
            {
              s1 = sindice;
              if (strlen(sindice) == 1)
                s1 += ":  ";
              else
                s1 += ": ";
              s1 += E;
                if (E == "" || E.GetLength() == 2 ||
E.GetLength() == 3 || E.GetLength() == 3)
                  s1 += "\t\t";
              else
                s1 += "\t";
              s1 += Nmo;
              s1 += "\t";
              s1 += Op;
              m_lineaprog = s1;
            }
            else
            {
              s1 = sindice;
              if (strlen(sindice) == 1)
                s1 += ":  ";
              else
                s1 += ": ";
              s1 += E;
                if (E == "" || E.GetLength() == 2 ||
E.GetLength() == 3)
                  s1 += "\t\t";
              else
                s1 += "\t";
              s1 += Nmo;
              s1 += "\t";
              s1 += Op;
              m_lineaprog = s1;
              m_cp = PC;
            }
            UpdateData(FALSE);
            EjecutaInstruccion();
            UpdateData(FALSE);
          }
    }
}
void CxDialog::EjecutaInstruccion()
{
```

```
char aux[6];

strcpy(aux,(const char *) CO);
m_lineaprog = m_lineaprog;
if(strcmp((const char *)Nmo,"DW") == 0)
  DW();
else
{
  if(strcmp((const char *)Nmo,"DB") == 0)
    DB();
  switch(aux[1])
  {
    case '9':{
               switch(aux[0])
               {
                 case 'A': ADC(4); break;
                 case 'B': ADC(5); break;
                 case 'C': ADC(6); break;
                 case 'F': ADC(7); break;
                 case 'E': ADC(8); break;
                 case 'D': ADC(9); break;
                 case '2': BHCS(1); break;
                 case '9': SEC(0); break;
                 case '4': ROL(2); break;
                 case '5': ROL(3); break;
                 case '3': ROL(5); break;
                 case '7': ROL(7); break;
                 case '6': ROL(8); break;
                 default: especial(); break;
               }
             }break;
    case '8':{
               switch(aux[0])
               {
                 case '4': ASLoLSL(2); break;
                 case '5': ASLoLSL(3); break;
                 case '3': ASLoLSL(5); break;
                 case '7': ASLoLSL(7); break;
                 case '6': ASLoLSL(8); break;
                 case 'A': EOR(4); break;
                 case 'B': EOR(5); break;
                 case 'C': EOR(6); break;
                 case 'F': EOR(7); break;
                 case 'E': EOR(8); break;
                 case 'D': EOR(9); break;
                 case '2': BHCC(1); break;
                 case '9': CLC(0); break;
                 default: especial(); break;
               }
             }break;
    case 'B':{
               switch(aux[0])
               {
                 case 'A': ADD(4); break;
                 case 'B': ADD(5); break;
                 case 'C': ADD(6); break;
                 case 'F': ADD(7); break;
                 case 'E': ADD(8); break;
                 case 'D': ADD(9); break;
                 case '2': BMI(1); break;
                 case '9': SEI(0); break;
                 default: especial(); break;
               }
             }break;
    case '7':{
               switch(aux[0])
               {
                 case 'B': STA(5); break;
                 case 'C': STA(6); break;
                 case 'F': STA(7); break;
                 case 'E': STA(8); break;
                 case 'D': STA(9); break;
                 case '4': ASR(2); break;
                 case '5': ASR(3); break;
                 case '3': ASR(5); break;
                 case '7': ASR(7); break;
                 case '6': ASR(8); break;
                 case '2': BEQ(1); break;
                 case '9': TAX(0); break;
                 default: especial(); break;
               }
             }break;
    case '4':{
               switch(aux[0])
               {
                 case 'A': AND(4); break;
                 case 'B': AND(5); break;
                 case 'C': AND(6); break;
                 case 'F': AND(7); break;
                 case 'E': AND(8); break;
                 case 'D': AND(9); break;
                 case '4': LSR(2); break;
                 case '5': LSR(3); break;
                 case '3': LSR(5); break;
                 case '7': LSR(7); break;
                 case '6': LSR(8); break;
                 case '2': BCCoBHS(1); break;
                 default: especial(); break;
               }
             }break;
    case 'F':{
               switch(aux[0])
               {
                 case 'B': STX(5); break;
                 case 'C': STX(6); break;
                 case 'F': STX(7); break;
                 case 'E': STX(8); break;
                 case 'D': STX(9); break;
                 case '4': CLR(2); break;
                 case '5': CLR(3); break;
                 case '3': CLR(5); break;
                 case '7': CLR(7); break;
                 case '6': CLR(8); break;
                 case '2': BIH(1); break;
                 case '9': TXA(0); break;
                 case '8': WAIT(0); break;
                 default: especial(); break;
               }
             }break;
    case '5':{
               switch(aux[0])
               {
                 case 'A': BIT(4); break;
                 case 'B': BIT(5); break;
                 case 'C': BIT(6); break;
                 case 'F': BIT(7); break;
                 case 'E': BIT(8); break;
                 case 'D': BIT(9); break;
                 case '2': BLOoBCS(1); break;
                 default: especial(); break;
               }
             }break;
    case '1':{
               switch(aux[0])
               {
                 case 'A': CMP(4); break;
                 case 'B': CMP(5); break;
                 case 'C': CMP(6); break;
                 case 'F': CMP(7); break;
                 case 'E': CMP(8); break;
                 case 'D': CMP(9); break;
```

```
                case '2': BRN(1); break;
                case '9': RTS(0); break;

                default: especial(); break;
            }
        }break;
case '3':{
        switch(aux[0])
        {
                case 'A': CPX(4); break;
                case 'B': CPX(5); break;
                case 'C': CPX(6); break;
                case 'F': CPX(7); break;
                case 'E': CPX(8); break;
                case 'D': CPX(9); break;
                case '4': COM(2); break;
                case '5': COM(3); break;
                case '3': COM(5); break;
                case '7': COM(7); break;
                case '6': COM(8); break;
                case '2': BLS(1); break;
                case '8': SWI(0); break;
                default: especial(); break;
        }
    }break;
case '2':{
        switch(aux[0])
        {
                case 'A': SBC(4); break;
                case 'B': SBC(5); break;
                case 'C': SBC(6); break;
                case 'F': SBC(7); break;
                case 'E': SBC(8); break;
                case 'D': SBC(9); break;
                case '2': BHI(1); break;
                default: especial(); break;
        }
    }break;
case '0':{
        switch(aux[0])
        {
                case 'A': SUB(4); break;
                case 'B': SUB(5); break;
                case 'C': SUB(6); break;
                case 'F': SUB(7); break;
                case 'E': SUB(8); break;
                case 'D': SUB(9); break;
                case '4': NEG(2); break;
                case '5': NEG(3); break;
                case '3': NEG(5); break;
                case '7': NEG(7); break;
                case '6': NEG(8); break;
                case '2': BRA(1); break;
                case '8': RTI(0); break;
                default: especial(); break;
        }
    }break;
case 'D':{
        switch(aux[0])
        {
                case 'A': BSR(1); break;
                case 'B': JSR(5); break;
                case 'C': JSR(6); break;
                case 'F': JSR(7); break;
                case 'E': JSR(8); break;
                case 'D': JSR(9); break;
                case '4': TST(2); break;
                case '5': TST(3); break;
                case '3': TST(5); break;
                case '7': TST(7); break;
                case '6': TST(8); break;
                case '2': BMS(1); break;
                case '9': NOP(0); break;
                default: especial(); break;
        }
    }break;
case '6':{
        switch(aux[0])
        {
                case 'A': LDA(4); break;
                case 'B': LDA(5); break;
                case 'C': LDA(6); break;
                case 'F': LDA(7); break;
                case 'E': LDA(8); break;
                case 'D': LDA(9); break;
                case '4': ROR(2); break;
                case '5': ROR(3); break;
                case '3': ROR(5); break;
                case '7': ROR(7); break;
                case '6': ROR(8); break;
                case '2': BNE(1); break;
                default: especial(); break;
        }
    }break;
case 'E':{
        switch(aux[0])
        {
                case 'A': LDX(4); break;
                case 'B': LDX(5); break;
                case 'C': LDX(6); break;
                case 'F': LDX(7); break;
                case 'E': LDX(8); break;
                case 'D': LDX(9); break;
                case '2': BIL(1); break;
                case '8': STOP(0); break;
                default: especial(); break;
        }
    }break;
case 'C':{
        switch(aux[0])
        {
                case 'B': JMP(5); break;
                case 'C': JMP(6); break;
                case 'F': JMP(7); break;
                case 'E': JMP(8); break;
                case 'D': JMP(9); break;
                case '4': INC(2); break;
                case '5': INC(3); break;
                case '3': INC(5); break;
                case '7': INC(7); break;
                case '6': INC(8); break;
                case '9': RSP(0); break;
                case '2': BMC(1); break;
                default: especial(); break;
        }
    }break
case 'A':{
        switch(aux[0])
        {
                case 'A': ORA(4); break;
                case 'B': ORA(5); break;
                case 'C': ORA(6); break;
                case 'F': ORA(7); break;
                case 'E': ORA(8); break;
                case 'D': ORA(9); break;
                case '4': DEC(2); break;
                case '5': DEC(3); break;
                case '3': DEC(5); break;
                case '7': DEC(7); break;
                case '6': DEC(8); break;
                case '9': CLI(0); break;
```

```
                    case '2': BPL(0); break;
                    default: especial(); break;
                }
            }break;

        default: {
                especial(); break;
            }
    }
}

void CxDialog::OnChangeMem()
{
    // TODO: Add your control notification handler code
here
    char aux[3];
    CString binario;

        bandera + + ;
        if(bandera = = 2)
        {
          aux[0] = m_mem[6];
          aux[1] = m_mem[7];
          aux[2] = '\0';
          m_PORTAh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_PORTAb = binario;
          ////////////////////////////////////
          aux[0] = m_mem[11];
          aux[1] = m_mem[12];
          aux[2] = '\0';
          m_PORTBh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_PORTBb = binario;
          ////////////////////////////////////
          if(m_procesador = = "P3")
            aux[0] = 'F';
          else
            aux[0] = m_mem[16];
          aux[1] = m_mem[17];
          aux[2] = '\0';
          m_PORTCh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_PORTCb = binario;
          ////////////////////////////////////
            aux[0] = m_mem[21];
            aux[1] = m_mem[22];
            aux[2] = '\0';
            m_PORTDh = aux;
            binario = ConvierteHexadecimalaBinario(aux);
            m_PORTDb = binario;
        ////////////////////////////////////////
        aux[0] = m_mem[26];
        aux[1] = m_mem[27];
        aux[2] = '\0';
        m_DDRAh = aux;
        binario = ConvierteHexadecimalaBinario(aux);
        m_DDRAb = binario;
        ////////////////////////////////////////
        aux[0] = m_mem[31];
        aux[1] = m_mem[32];
        aux[2] = '\0';
        m_DDRBh = aux;
        binario = ConvierteHexadecimalaBinario(aux);
        m_DDRBb = binario;
        ////////////////////////////////////////
        aux[0] = m_mem[36];
        aux[1] = m_mem[37];
        aux[2] = '\0';
          m_DDRCh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_DDRCb = binario;
          ////////////////////////////////////////
          aux[0] = m_mem[46];
          aux[1] = m_mem[47];
          aux[2] = '\0';
          m_TDRh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_TDRb = binario;
          ////////////////////////////////////////
          aux[0] = m_mem[51];
          aux[1] = m_mem[52];
          aux[2] = '\0';
          m_TCRh = aux;
          binario = ConvierteHexadecimalaBinario(aux);
          m_TCRb = binario;
          UpdateData(FALSE);
          bandera = 0;
        }
}

void CxDialog::OnChangeDDRAb()
{
        // TODO: Add your control notification handler
code here
    char auxmem[9];
    CString hexadecimal;

        bandera + + ;
        if(bandera = = 2)
        {
          strcpy(auxmem,(const char*) m_DDRAb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
          m_mem.SetAt(26,hexadecimal[0]);
          m_mem.SetAt(27,hexadecimal[1]);
          m_DDRAh = hexadecimal;
          bandera = 0;
        }
}

void CxDialog::OnChangeDDRAh()
{
        // TODO: Add your control notification handler
code here
    char auxmem[3];
    CString binario;

        bandera + + ;
        if(bandera = = 2)
        {
          strcpy(auxmem,(const char*)m_DDRAh);

binario = ConvierteHexadecimalaBinario(auxmem);
          m_mem.SetAt(26,auxmem[0]);
          m_mem.SetAt(27,auxmem[1]);
          m_DDRAb = binario;
          bandera = 0;
        }
}

void CxDialog::OnChangeDDRBb()
{
        // TODO: Add your control notification handler
code here
    char auxmem[9];
    CString hexadecimal;

        bandera + + ;
        if(bandera = = 2)
        {
```

```
            UpdateData(TRUE);
            strcpy(auxmem,(const char*)m_DDRBb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
            m_mem.SetAt(31,hexadecimal[0]);
            m_mem.SetAt(32,hexadecimal[1]);
            m_DDRBh = hexadecimal;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangeDDRBh()
{
        // TODO: Add your control notification handler
code here
    char auxmem[3];
    CString binario;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_DDRBh);

binario = ConvierteHexadecimalaBinario(auxmem);
            m_mem.SetAt(31,auxmem[0]);
            m_mem.SetAt(32,auxmem[1]);
            m_DDRBb = binario;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangeDDRCb()
{
        // TODO: Add your control notification handler
code here
    char auxmem[9];
    CString hexadecimal;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_DDRCb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
            m_mem.SetAt(36,hexadecimal[0]);
            m_mem.SetAt(37,hexadecimal[1]);
            m_DDRCh = hexadecimal;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangeDDRCh()
{
        // TODO: Add your control notification handler
code here
    char auxmem[3];
    CString binario;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_DDRCh);

binario = ConvierteHexadecimalaBinario(auxmem);
            m_mem.SetAt(36,auxmem[0]);
            m_mem.SetAt(37,auxmem[1]);
            m_DDRCb = binario;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangePORTAb()
{
        // TODO: Add your control notification handler
code here
    char auxmem[9];
    CString hexadecimal;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTAb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
            m_mem.SetAt(6,hexadecimal[0]);
            m_mem.SetAt(7,hexadecimal[1]);
            m_PORTAh = hexadecimal;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangePORTAh()
{
        // TODO: Add your control notification handler
code here
    char auxmem[3];
    CString binario;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTAh);

binario = ConvierteHexadecimalaBinario(auxmem);
            m_mem.SetAt(6,auxmem[0]);
            m_mem.SetAt(7,auxmem[1]);
            m_PORTAb = binario;
            UpdateData(FALSE);
            bandera = 0;
        }
}

void CxDialog::OnChangePORTBb()
{
        // TODO: Add your control notification handler
code here
    char auxmem[9];
    CString hexadecimal;

        bandera + + ;
        if(bandera % = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTBb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
            m_mem.SetAt(11,hexadecimal[0]);
            m_mem.SetAt(12,hexadecimal[1]);
            m_PORTBh = hexadecimal;
            UpdateData(FALSE);
            bandera = 0;
        }
```

```
}
void CxDialog::OnChangePORTBh()
{
        // TODO: Add your control notification handler
code here
   char auxmem[3];
   CString binario;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTBh);

binario = ConvierteHexadecimalaBinario(auxmem);
          m_mem.SetAt(11,auxmem[0]);
          m_mem.SetAt(12,auxmem[1]);
          m_PORTBb = binario;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangePORTCb()
{
        // TODO: Add your control notification handler
code here
   char auxmem[9];
   CString hexadecimal;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTCb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
          m_mem.SetAt(16,hexadecimal[0]);
          m_mem.SetAt(17,hexadecimal[1]);
          m_PORTCh = hexadecimal;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangePORTCh()
{
        // TODO: Add your control notification handler
code here
   char auxmem[3];
   CString binario;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTCh);

binario = ConvierteHexadecimalaBinario(auxmem);
          m_mem.SetAt(16,auxmem[0]);
          m_mem.SetAt(17,auxmem[1]);
          m_PORTCb = binario;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangePORTDb()
{
        // TODO: Add your control notification handler
code here
   char auxmem[9];
   CString hexadecimal;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTDb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
          m_mem.SetAt(21,hexadecimal[0]);
          m_mem.SetAt(22,hexadecimal[1]);
          m_PORTDh = hexadecimal;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangePORTDh()
{
        // TODO: Add your control notification handler
code here
   char auxmem[3];
   CString binario;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_PORTDh);

binario = ConvierteHexadecimalaBinario(auxmem);
          m_mem.SetAt(21,auxmem[0]);
          m_mem.SetAt(22,auxmem[1]);
          m_PORTDb = binario;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangeAcuabl()
{
        // TODO: Add your control notification handler
code here
        char auxmem[9];
   CString hexadecimal;

        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          strcpy(auxmem,(const char*)m_AAb);

hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
          m_AAh = hexadecimal;
          UpdateData(FALSE);
          bandera = 0;
        }

}
void CxDialog::OnChangeAcuahl()
{
        // TODO: Add your control notification handler
code here
        char auxmem[3];
   CString binario;
```

```
            bandera = + + ;
            if(bandera = = 2)
            {
              UpdateData(TRUE);
              strcpy(auxmem,(const char*)m_AAh);
binario = ConvierteHexadecimalaBinario(auxmem);
              m_AAb = binario;
              UpdateData(FALSE);
              bandera = 0;
            }
}

void CxDialog::OnChangeIndxb()
{
            // TODO: Add your control notification handler
code here
            char auxmem[9];
    CString hexadecimal;

            bandera = + + ;
            if(bandera = = 2)
            {
              UpdateData(TRUE);
              strcpy(auxmem,(const char*)m_INDXb);
hexadecimal = ConvierteBinarioaHexadecimal(auxmem);
              m_INDXh = hexadecimal;
              UpdateData(FALSE);
              bandera = 0;
            }
}

void CxDialog::OnChangeIndxh()
{
            // TODO: Add your control notification handler
code here
            char auxmem[3];
    CString binario;

            bandera = + + ;
            if(bandera = = 2)
            {
              UpdateData(TRUE);
              strcpy(auxmem,(const char*)m_INDXh);
binario = ConvierteHexadecimalaBinario(auxmem);
              m_INDXb = binario;
              UpdateData(FALSE);
              bandera = 0;
            }
}

int CxDialog::BuscaDireccionMemoria(CString a0)
{
    CString aux;
    char aux2[5];
    int index = 0;
    int auxindex = 0;

    aux = a0[0];
    aux + = a0[1];
    aux + = a0[2];
    aux + = "0";
    strcpy(aux2,(const char *)aux);
    if( (aux = = Org && EsOrg))
    {
      m_mem + = aux;
      m_mem + = ": 00   00   00   00   00   00   00   00
00   00   00   00   00   00   00   00\r\n";
      index = ConvierteHexadecimalaEntero(aux2);
      index = index/16;
      index = index*85;
      Indexant = index;
      EsOrg = FALSE;
    }
    aux2[0] = m_mem[Indexant];
    aux2[1] = m_mem[Indexant + 1];
    aux2[2] = m_mem[Indexant + 2];
    aux2[3] = m_mem[Indexant + 3];
                        if(m_mem[Indexant] = = aux[0]          &&
m_mem[Indexant + 1] = = aux[1]          &&
m_mem[Indexant + 2] = = aux[2]          &&
m_mem[Indexant + 3] = = aux[3])
    {
      aux2[0] = m_mem[Indexant];
      aux2[1] = m_mem[Indexant + 1];
      aux2[2] = m_mem[Indexant + 2];
      aux2[3] = m_mem[Indexant + 3];
      index = Indexant;
    }
    else
    {
      m_mem + = aux;
      m_mem + = ": 00   00   00   00   00   00   00   00
00   00   00   00   00   00   00   00\r\n";
      index = Indexant + 85;
      Indexant = index;
      aux + = m_mem[index-4];
      aux + = m_mem[index-3];
      aux + = m_mem[index-2];
      aux + = m_mem[index-1];
      aux + = m_mem[index];
      aux + = m_mem[index + 1];
      aux + = m_mem[index + 2];
      aux + = m_mem[index + 3];
    }
                        if(aux[0] = = m_mem[index]          &&
aux[1] = = m_mem[index + 1]          &&
aux[2] = = m_mem[index + 2]          &&
aux[3] = = m_mem[index + 3])
    {
      switch(a0[3])
      {
        case'0': index = index + 6; break;
        case'1': index = index + 11; break;
        case'2': index = index + 16; break;
        case'3': index = index + 21; break;
        case'4': index = index + 26; break;
        case'5': index = index + 31; break;
        case'6': index = index + 36; break;
        case'7': index = index + 41; break;
        case'8': index = index + 46; break;
        case'9': index = index + 51; break;
        case'A': index = index + 56; break;
        case'B': index = index + 61; break;
        case'C': index = index + 66; break;
        case'D': index = index + 71; break;
        case'E': index = index + 76; break;
        case'F': index = index + 81; break;
      }
    }
    aux2[0] = m_mem[index];
    aux2[1] = m_mem[index + 1];
    aux2[2] = m_mem[index + 2];
    aux2[3] = m_mem[index + 3];

    return index;
}

CString CxDialog::ObtenIndice(CString dir)
{
    CString aux = "\0";
    CString aux2 = "";
```

```
CHexadecimal a1("");
int index = 0;

index = 0;
aux + = dir[0];
aux + = dir[1];
aux + = dir[2];
aux + = "0";

aux2 = m_mem[index];
aux2 + = m_mem[index + 1];
aux2 + = m_mem[index + 2];
aux2 + = m_mem[index + 3];
aux2 + = '\0';

while(aux1 = aux2)
{
   index = index + 85;
   aux2 = m_mem[index];
   aux2 + = m_mem[index + 1];
   aux2 + = m_mem[index + 2];
   aux2 + = m_mem[index + 3];
   aux2 + = '\0';

}
if(index > m_mem.GetLength())
{
                MessageBox("Error  de  asignaci¾n  de
memoria","Information",MB_OK);
   aux = "";
   return aux;
}
else
switch(dir[3])
{
   case'0': index = index + 6; break;
   case'1': index = index + 11; break;
   case'2': index = index + 16; break;
   case'3': index = index + 21; break;
   case'4': index = index + 26; break;
   case'5': index = index + 31; break;
   case'6': index = index + 36; break;
   case'7': index = index + 41; break;
   case'8': index = index + 46; break;
   case'9': index = index + 51; break;
   case'A': index = index + 56; break;
   case'B': index = index + 61; break;
   case'C': index = index + 66; break;
   case'D': index = index + 71; break;
   case'E': index = index + 76; break;
   case'F': index = index + 81; break;
}
aux = m_mem[index];
aux + = m_mem[index + 1];
return aux;
}

CString CxDialog::ObtenIndice(CString dir, int &index)
{
   CString aux = "\0";
   CString aux2 = "";
   CHexadecimal a1("");

   index = 0;
   aux + = dir[0];
   aux + = dir[1];
   aux + = dir[2];
   aux + = "0";

   aux2 = m_mem[index];
   aux2 + = m_mem[index + 1];
   aux2 + = m_mem[index + 2];
   aux2 + = m_mem[index + 3];
   aux2 + = '\0';

   while(aux1 = aux2)
   {
      index = index + 85;
      aux2 = m_mem[index];
      aux2 + = m_mem[index + 1];
      aux2 + = m_mem[index + 2];
      aux2 + = m_mem[index + 3];
      aux2 + = '\0';

   }
   if(index > m_mem.GetLength())
   {
                MessageBox("Error  de  asignaci¾n  de
memoria","Information",MB_OK);
      aux = "";
      return aux;
   }
   else
   switch(dir[3])
   {
      case'0': index = index + 6; break;
      case'1': index = index + 11; break;
      case'2': index = index + 16; break;
      case'3': index = index + 21; break;
      case'4': index = index + 26; break;
      case'5': index = index + 31; break;
      case'6': index = index + 36; break;
      case'7': index = index + 41; break;
      case'8': index = index + 46; break;
      case'9': index = index + 51; break;
      case'A': index = index + 56; break;
      case'B': index = index + 61; break;
      case'C': index = index + 66; break;
      case'D': index = index + 71; break;
      case'E': index = index + 76; break;
      case'F': index = index + 81; break;
   }
   aux = m_mem[index];
   aux + = m_mem[index + 1];
   return aux;
}
void CxDialog::OnChangeCp()
{
      // TODO: Add your control notification handler
code here
   CString binario;

         bandera + + ;
         if(bandera = = 2)
         {
            UpdateData(TRUE);
      if(m_cp < pInstruccion[indice].PC)
         {
            while(m_cp! = pInstruccion[indice].PC && indice
> = 0)
            indice--;
         }
      if(m_cp > pInstruccion[indice].PC)
         {
                  while(m_cp! = pInstruccion[indice].PC &&
indice < = count)
            indice + + ;
         }
      if(m_cp = = pInstruccion[indice].PC)
         OnClickedPasopaso();
      bandera = 0;
         }
}
```

```
void CxDialog::OnChangeC()
{
        // TODO: Add your control notification handler
code here
        bandera + + ;
        if(bandera = = 2)
          {
    UpdateData(TRUE);
          bandera = 0;
        }
}
void CxDialog::OnChangeH()
{
        // TODO: Add your control notification handler
code here
        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          bandera = 0;
        }
}
void CxDialog::OnChangeI()
{
        // TODO: Add your control notification handler
code here
        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          bandera = 0;
        }
}
void CxDialog::OnChangeN()
{
        // TODO: Add your control notification handler
code here
        UpdateData(TRUE);
}
void CxDialog::OnChangeZ()
{
        // TODO: Add your control notification handler
code here
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          bandera = 0;
        }
}
void CxDialog::OnTimer(UINT nIDEvent)
{
        // TODO: Add your message handler code here
and/or call default
        UpdateData(FALSE);
        CDialog::OnTimer(nIDEvent);
}
void CxDialog::OnClickedPara()
{
        // TODO: Add your control notification handler
code here
        if(m_lCount = = 0L)
          CDialog::OnCancel();
        else
          m_lCount = m_lMaxCount;
}
void CxDialog::OnChangeTimer()
{
        // TODO: Add your control notification handler
code here
        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          if(m_timer = = '1')
          {
            EjecutaTimer();
            EjecutaInstruccion();
          }
    bandera = 0;
  }
}
void CxDialog::OnChangeInt()
{
        // TODO: Add your control notification handler
code here
        bandera + + ;
        if(bandera = = 2)
        {
          UpdateData(TRUE);
          bandera = 0;
        }
}
```

Glosario

Glosario

ACCA	Acumulador
An	Bit de estado. Bit n de ACCA
C	Bit 0 del CCR. Indica acarreo/préstamo
CCR	Registro de codigo de condición
CD	Convertidor digital
CMOS	Tecnología Complementaria Metal-Óxido-Semiconductor
dd	Operando relativo
DDR	Registro de dirección de datos
DLL	Bibliotecas de enlace dinámico
DR	Operando con modo de direccionamiento directo
H	Bit 4 de CCR. Indica medio acarreo
I	Bit 3 del CCR. Para interrupción enmascarable
INDX	Registro índice X
M	Contenido de memoria
MCU	Microcontrolador
MDI	Interfaz de documentos múltiples
MFC	Biblioteca de clases de Microsoft Fundation Class
Mn	Bit n de un elemento de memoria

Mnemónico	Nombre de la instrucción a ejecutarse en lenguaje ensamblador
MOR	Registro de opción enmascarable
N	Bit 2 del CCR. Indicador de negativo
PC	Contador del programa
POO	Programación Orientada a Objetos
RAM	Memoria de acceso aleatorio
Rel	Cantidad de bytes a saltar en una instrucción de tipo de direccionamiento relativo
Reset	Reinicializador
Rn	Bit n del resultado de una operación en binario
ROM	Memoria de sólo lectura
SP	Apuntador a la pila (stack pointer)
SWI	Interrupción por software
TCR	Registro de control del timer
Timer	Temporizador
TTL	Lógica Transistor-Transistor
Xn	Bit n del INDX
Z	Bit 1 del CCR. Indicador de cero

Bibliografía

Bibliografía

Libros

Keil, Heinrich; "Microcomputadores", Marcombo, España, 1988.

Kruglinski, David; "Progrese con Visual C++", McGrawHill, España, 1994.

Martín, James; "Fourt Generation Languages", Volumen 1, Prentice Hall, E.E.U.U., 1985.

Microsoft; "Microsoft Visual C++, Funciones y aplicaciones", McGraw-Hill, España, 1994.

Motorola; "M6805 HMOS/146805 CMOS Family Users Manual", Prentice-Hall, E.E.U.U., 1983.

Motorola; "Motorola Semiconductor, Technical Data, Prentice-Hall, E.E.U.U.

Perry, Greg; "Aprendiendo Programación Orientada a Objetos con Turbo C++ en 21 días", Prentice Hall, México, 1995.

Shammas, Namir; "Aprendiendo Visual C++ 2 en 21 días", Prentice Hall, México, 1996.

Schildt, Herbert; "Programación en C y C++ en Windows 95", McGrawHill, México, 1995.

Sorenson, Paul/Tremblay, Jean; "The theory and practice of compiler writing", McGrawHill, E.E.U.U.

Watson, Des; "High-Level Languages and their Compilers, Adisson Wesley, Great Britian, 1989.

Internet:

Kruglinski, David; v-davidk@microsoft.com.

Microsoft; http:/microsoft.com/visualc

Revistas:

"Business Week", Octubre 1995, pags. 60 y 62.

"Computer World", Vol. 29 no. 27, Julio 1995, pag. 65.

"PC Magazine", Vol. 13 no. 6, Marzo 1994, pags. 188 y 212.

"Personal Computing México", Informe especial, Mayo 1995, pag. 36