UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ciencias y Humanidades

Desarrollo de un controlador lógico programable (PLC) habilitado para la conexión con Internet, con el manejo de los protocolos TCP/IP y el estándar Ethernet.

Trabajo de Graduación presentado por Carlos Alberto Esquit Hernández para optar al grado académico de Licenciado en Ingeniería Electrónica.

Guatemala

2003

Desarrollo de un controlador lógico programable (PLC) habilitado para la conexión con Internet, con el manejo de los protocolos TCP/IP y el estándar Ethernet.

CONTENIDO

LISTA DE TABLAS

Vo.Bo.:

(f) _____
Ing. Gonzalo Palaréa Murga

Asesor

Tribunal:

(f) _____
Dr. Ing. Manuel A. López V.

(f) _____
Ing. Gabriel Rouanet Mora

(f) _____
Ing. Gonzalo Palaréa Murga

Fecha de aprobación: 10 de Julio de 2003

PREFACIO

La documentación que aquí se presenta es el resultado de un largo proceso de investigación, diseño y desarrollo del proyecto que de aquí en adelante será llamado *iPLC*. Este proceso fue extenso ya que el autor requirió estudiar diversos temas que desconocía por completo, y profundizar en otros de los cuales tan solo poseía conocimientos muy básicos. Algunos de estos temas son muy extensos, por lo que no pueden ser tratados en detalle en este trabajo. Los detalles de estos temas son importantes para el autor, quien necesitó de ellos para realizar el diseño y construcción del proyecto que en este documento se describe. Sin embargo, esos detalles son prescindibles para la presentación del proyecto hacia el lector. Por ello, se decidió incluir únicamente las nociones básicas de estos temas, presentándolos en el capítulo III, para que el lector adquiera las ideas y conceptos básicos que le permitan comprender el proyecto realizado.

En el capítulo II se presenta el planteamiento del proyecto, desde el punto de vista del autor, indicando los objetivos propios y los procedimientos de trabajo a ejecutar para el desarrollo del mismo.

Con el propósito de ofrecer al lector una visión completa del proyecto, y a la vez evitar que el contenido resulte muy enredado, se hizo una distinción entre los diferentes tipos de subsistemas que posee el proyecto. La idea es que el lector pueda concentrarse en un subsistema a la vez, y de esa forma, al haber comprendido cada uno de ellos, le resulte fácil comprender el proyecto como un todo. Es así, que en el capítulo IV se discute el sistema de hardware, en el capítulo V se trata acerca del sistema de software de bajo nivel, y en el capítulo VI se presenta el sistema de software de alto nivel.

En este trabajo se debe asumir que el lector posee conocimientos acerca de la programación de PLC's, pues resulta muy difícil incluir un tutor acerca de esto debido a la extensión que implicaría. El lenguaje de programación del iPLC es muy similar al lenguaje STL utilizado en los PLC's Siemens, por ello, en el capítulo VII se presentan las características fundamentales del lenguaje del iPLC, que son indispensables para que el usuario pueda programarlo adecuadamente.

Finalmente, en el capítulo VIII se muestran tablas comparativas entre las características básicas de los sistemas S7-200 de Siemens y del iPLC, de forma que se puedan evaluar los resultados.

Los diagramas esquemáticos de todos los circuitos del iPLC, y el código fuente de todos los programas, han sido incluidos al final, en los apéndices de este trabajo, de tal forma que puedan servir como referencia completa para el lector que desee profundizar en los detalles de la elaboración del iPLC.

LISTA DE FIGURAS

continuación de la lista de figuras...

RESUMEN

El proyecto consiste en el desarrollo de un PLC IP, que posee la capacidad para manejar la familia de protocolos TCP/IP y que puede ser conectado directamente en una red Ethernet. Para el desarrollo del PLC se tomó como modelo a la serie S7-200 de la empresa alemana Siemens. El PLC está diseñado para la compatibilidad con módulos de expansión, que le permiten expandir sus capacidades de entrada / salida y le agregan funciones especiales. Además de diseñar e implementar la CPU central (PLC), se diseñaron dos módulos de expansión, uno digital con ocho entradas y ocho salidas, y uno analógico con dos entradas y dos salidas.

El iPLC fue habilitado para la conexión con Internet de forma exitosa. Esta habilitación fue realizada por medio de la tecnología ofrecida en el módulo SitePlayer, producido por Netmedia Inc. El entorno de programación y monitorización del PLC IP desarrollado en este proyecto se encuentra en un servidor HTTP incrustado en el sistema del PLC, por lo que estas tareas (programación y monitorización) pueden ser realizadas de forma remota, utilizando un explorador de Internet. Todo lo que el iPLC necesita para ser programado y monitorizado desde cualquier parte del mundo es una red Ethernet y una dirección IP pública.

Se desarrolló un lenguaje de programación casi idéntico al lenguaje STL (Statement List) de la serie S7-200 de los PLC's Siemens. Se implementaron 106 instrucciones que ofrecen la funcionalidad básica de la serie S7-200, con algunas características inferiores y otras superiores. El hardware del iPLC presenta mayor capacidad de almacenamiento, expansión y cálculo matemático que la serie S7-200 de Siemens. La principal debilidad del iPLC es su velocidad de procesamiento.

I. INTRODUCCIÓN

En la actualidad, la electrónica digital juega un papel importante, ya que todos los sistemas y dispositivos tienden a ser digitalizados, presentando interfases digitales para el usuario / operador, basadas en el desarrollo de sistemas microprocesados o microcontrolados. Los microcontroladores son dispositivos populares alrededor de todo el mundo, son utilizados en muchas aplicaciones de electrónica, y permiten simplificar el diseño de los sistemas electrónicos. La simplificación está basada en la programación de la microelectrónica del dispositivo (electrónica interna en el microcontrolador). Otras ventajas son la reducción de costos y la reducción en el tamaño del sistema electrónico completo.

Los microcontroladores son utilizados en todo tipo de soluciones electrónicas, desde juguetes, aparatos para el hogar y la oficina, hasta dispositivos y sistemas en las plantas industriales. Considerando ese amplio campo de aplicación para los microcontroladores, algo de utilidad en la actualidad es la habilitación de la Internet directamente en los sistemas microcontrolados. Esta habilitación se refiere a que los microcontroladores mismos poseen la capacidad de conectarse a una red Ethernet y manejan la familia de protocolos de la Internet. Con ello se evita el uso de una computadora y se logra que los sistemas microcontrolados formen parte de la Internet de una forma simple e independiente.

Empresas como Microchip Technology Inc., Yipee Inc., ambas de los Estados Unidos de América, Iosoft Ltd., del Reino Unido, así como Embedded Ethernet de Noruega, entre otras, están trabajando en el desarrollo de soluciones al "problema" de la conexión a la Internet para sistemas microcontrolados. Las primeras soluciones propuestas por esas empresas consisten en un programa que maneja la familia de protocolos de la Internet (protocolos TCP/IP) y está diseñado para que sea integrado con el programa principal del microcontrolador de la aplicación específica. Ese programa es la solución que se conoce con el nombre de "stack de TCP/IP". El trabajo de esas empresas continúa, con el propósito de obtener mejores soluciones, que sean más

eficientes y permitan la habilitación de la Internet de una forma más sencilla. Las nuevas soluciones incorporan la capacidad para interactuar directamente con variables internas en la memoria local del microcontrolador de la aplicación específica. Así, las variables del sistema microcontrolado son accesibles desde una computadora remota. Esas nuevas soluciones se conocen con el nombre de "stacks de TCP/IP con procesadores de objetos" [1].

La habilitación de la Internet en los microcontroladores es de utilidad para el desarrollo de todos los aparatos electrónicos, ya que ello permite conectar directamente cualquier equipo a la Internet. Esto hace que dicha red sea de utilidad no solo para aparatos electrónicos como las computadoras y los teléfonos celulares, sino para cualquier aparato que posea un microcontrolador. Esta habilitación hace posible operar y monitorizar remotamente los aparatos del hogar, la oficina y la industria. Por ejemplo, en la industria esta habilitación es de utilidad para realizar mediciones remotas directamente del equipo instalado en una fábrica.

Los controladores lógicos programables (PLC's) son dispositivos importantes en los sistemas de automatización actuales. Las industrias tienden a modernizarse, e introducen sistemas automáticos en las líneas de producción. La habilitación de la Internet en los PLC's es importante para la industria, ya que permite realizar mediciones, lecturas y ajustes remotos en los parámetros y variables de un proceso de producción. De esta manera, utilizando un explorador de Internet se puede consultar, ajustar y controlar a dispositivos y máquinas instalados en una fábrica, desde cualquier parte del mundo. Por ello, se considera que el desarrollo de un PLC IP, que se pueda programar y monitorizar desde cualquier parte del mundo, a través de un explorador de Internet, es un proyecto de utilidad, proyección y vanguardia.

[1] Si desea obtener mayor información acerca de algunas empresas que desarrollan soluciones para la conectividad con Internet en los sistemas microcontrolados, consulte los siguientes sitios de Internet:
HTTP://www.yipeeinc.com, HTTP://www.microchip.com, HTTP://www.embeddedethernet.com
HTTP://www.livedevices.com, HTTP://www.iosoft.com, HTTP://www.cmx.com
HTTP://www.emware.com, HTTP://www.SitePlayer.com

II. PLANTEAMIENTO DEL PROYECTO

A. Objetivos para el proyecto desarrollado

Con la implementación de este proyecto, el autor pretende satisfacer los siguientes objetivos específicos:

- Estudiar las características generales de los controladores lógicos programables.
- Diseñar e implementar un controlador lógico programable que posea las funciones básicas de los PLC's de la serie S7-200 de la empresa alemana Siemens.
- Investigar las tecnologías existentes para la habilitación de la Internet en sistemas microcontrolados.
- Investigar lenguajes de programación para la Internet.
- Habilitar la conectividad con Internet en un sistema microcontrolado.
- Diseñar un sistema que permita programar un PLC de forma remota, a través de un explorador de Internet.
- Diseñar un sistema que permita acceder a variables y parámetros de un PLC a través de un explorador de Internet.

B. Método de trabajo

Para crear una división general del desarrollo de todo el proyecto, considérense siete fases de trabajo definidas de la siguiente forma:

1. **Fase de documentación.** En esta fase del proceso se investigan los elementos "claves" del proyecto de trabajo de graduación, como lo son, los controladores lógicos programables, las tecnologías para el manejo de la familia de protocolos TCP/IP, tecnologías para la interacción de sistemas microcontrolados con redes Ethernet y dispositivos especializados que pueden ser de utilidad para la implementación de funciones en el PLC.

2. Fase de pruebas preliminares. En esta fase se realizan las pruebas y experimentos necesarios para aprender los detalles en cuanto al funcionamiento y manejo de los elementos estudiados en la *fase de documentación*, y que se consideren necesarios para el futuro diseño del sistema.

3. Fase de diseño. De acuerdo al conocimiento adquirido en las dos fases anteriores, en esta fase se diseñan los circuitos electrónicos y el sistema de software necesarios para la implementación del controlador lógico programable habilitado para la conexión con Internet. También se diseña el sistema de software que programa y monitoriza a dicho PLC.

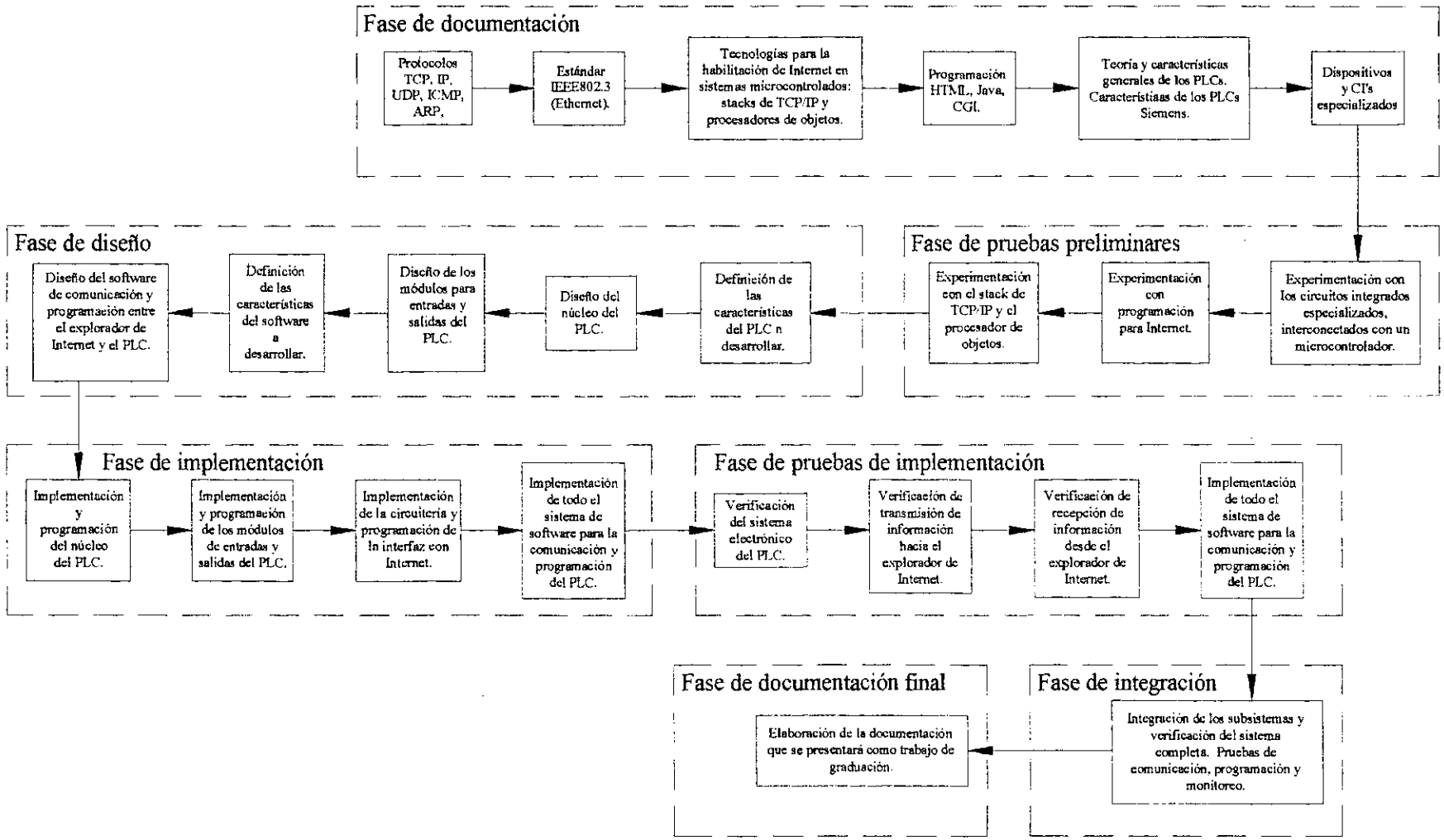
4. Fase de implementación. Esta es la fase en la cual toda la documentación obtenida, las ideas y diseños realizados se combinan para crear un sistema electrónico real. En esta fase se arman los circuitos electrónicos y se realiza toda la programación necesaria para hacer que los subsistemas requeridos por el proyecto de trabajo de graduación queden completamente implementados.

5. Fase de pruebas de implementación. Aquí se realizan las pruebas necesarias para determinar si cada subsistema funciona adecuadamente, de ser así, se procede con la fase siguiente, de lo contrario se utiliza una fase auxiliar para la solución de los problemas encontrados, o para un rediseño si se considera necesario.

6. Fase de integración. En esta fase se "monta" el sistema completo, a partir de los subsistemas ya implementados y verificados. Se realizarán las pruebas necesarias para la verificación del funcionamiento del proyecto de trabajo de graduación completo y se trabaja sobre los detalles que requieran mejoras.

7. Fase de documentación final. En esta fase se elabora la documentación del proyecto de graduación realizado, esta es la documentación que se presenta en los capítulos siguientes.

Figura 2.1: "Principales tareas a realizar"



III. TEMAS FUNDAMENTALES

A. Controladores lógicos programables

Un PLC (de las siglas en inglés *Programmable Logic Controller*) es un dispositivo que fue inventado para sustituir la lógica de relevadores electromecánicos que se utilizaba para el control de las máquinas en las plantas industriales. Los PLC's son dispositivos orientados a entradas y salidas, que basándose en el estado de sus entradas, y por medio de un programa cargado en su memoria, toma decisiones y finalmente actualiza sus salidas. El proceso de la planta industrial está representado dentro del PLC por medio de todas sus entradas y salidas instaladas en el sistema. Estas entradas y salidas, son conectadas a diferentes tipos de elementos, como interruptores, sensores, válvulas electrónicas, motores, pistones, etcétera, que cumplen funciones de detectores o actuadores dentro del proceso de producción. De esta forma, los PLC's tienen la función de controladores centrales en un proceso de producción automatizado, en el cual, todos los dispositivos de acción son controlados por el PLC, según la información ofrecida por los detectores, y de acuerdo a las órdenes establecidas por el programa de usuario, que es cargado en la memoria del PLC previo a su instalación en la planta de producción.

1. Componentes de un PLC.

a. El microprocesador. Este componente se encarga de la ejecución de las instrucciones del programa de usuario, y del procesamiento de todos los datos dentro del sistema, incluyendo a las entradas, salidas, y a todas las variables de usuario, procesadas durante la ejecución del programa.

b. Relevadores de entrada. Existen físicamente en el PLC, y son los que se conectan con los dispositivos externos como interruptores o sensores. También son conocidos como "contactores". Pueden estar implementados con relevadores electromecánicos, transistores BJTs, MOSFETs o tiristores, lo cual debe ser tomado en cuenta según la aplicación.

c. Relevadores de salida. También conectan al PLC con el exterior de éste. También son conocidos como "bobinas". En estos relevadores se conectan los actuadores del proceso automático, como por ejemplo solenoides, luces, bombas, etcétera. Dependiendo de la aplicación, pueden estar implementados con transistores, relevadores, o triacs.

d. Relevadores de utilidades. Estos relevadores no existen fisicamente, en cambio, son implementados en software dentro del sistema del PLC. Sirven para realizar tareas específicas como por ejemplo inicializar variables. Algunos están siempre activados, otros están siempre desactivados, otros se encuentran activos únicamente durante un tiempo predeterminado o bajo ciertas condiciones.

e. Entradas / salidas especiales. Además de las entradas o salidas de relevadores, el PLC puede poseer entradas de diferentes tipos, como por ejemplo, entradas / salidas analógicas, salidas PWM, salidas de impulsos, etcétera. Estas entradas / salidas especiales, pueden estar implementadas en el PLC propiamente, aunque generalmente se encuentran en módulos de expansión, que son conectados al PLC para expandir las capacidades de éste.

f. Memoria de usuario. Los PLC's deben poseer memoria disponible para las variables del programa de usuario. En esta memoria se almacenan datos en general.

g. Contadores. Los contadores están implementados en software, son utilizados para contar pulsos en las entradas del PLC, o en las variables internas, en el estado de éstas. También pueden existir contadores rápidos, generalmente implementados en hardware, diseñados específicamente para realizar el conteo de pulsos rápidos que no podrían ser detectados correctamente con los contadores en software.

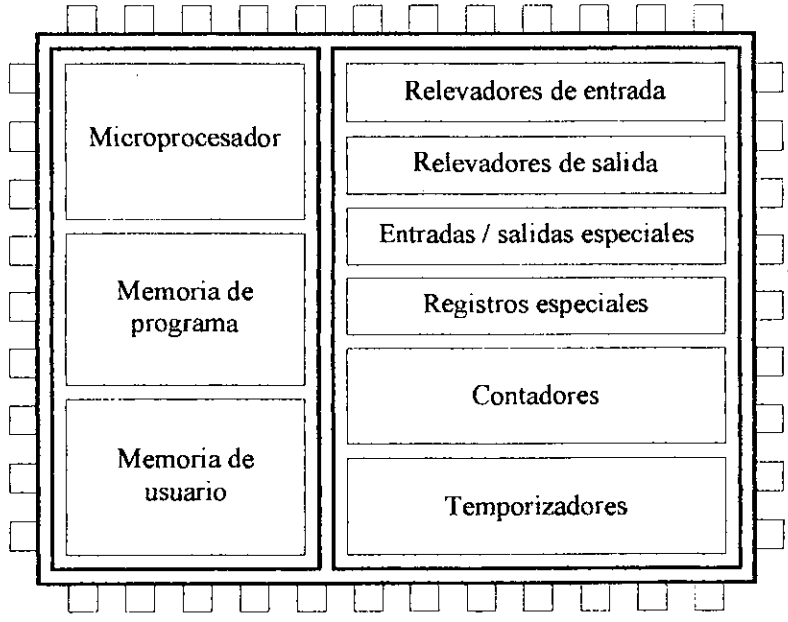
h. Temporizadores. Estos dispositivos permiten temporizar diversas acciones que se ejecutan dentro del PLC. Son utilizados ampliamente, pues son una herramienta poderosa para la organización y sincronización de las tareas dentro de un

proceso de producción. Debido a la gran utilidad de los temporizadores, generalmente los PLC's poseen una cantidad elevada de ellos, razón por la cual son implementados en software. Los temporizadores de un PLC pueden ser de diferentes resoluciones, típicamente de 1 mS, 10 mS y 100 mS.

i. Registros especiales. Son registros reservados para funciones especiales implementadas en un PLC, como por ejemplo, un reloj de tiempo real incorporado. En estos casos, el PLC posee un conjunto de registros que operan directamente con el dispositivo especial, ya sea para lectura o escritura, según sea el caso.

j. Memoria de programa. Esta memoria es de tipo no volátil. En esta memoria se encuentra almacenado el programa cargado al PLC.

Figura 3.1: "Componentes de un PLC"



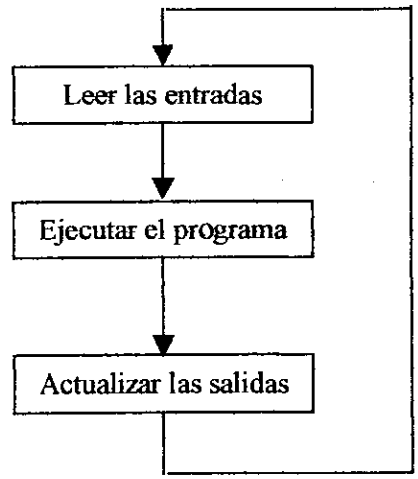
2. Funcionamiento de los PLC's. Generalmente los PLC's ofrecen dos modos de operación, uno de ellos llamado *modo detenido*, en el cual el PLC únicamente ejecuta las funciones básicas, y no realiza las tareas para las cuales fue programado. El

segundo modo de operación se llama *modo de ejecución*, en el cual, el PLC realiza todas las tareas posibles, entre ellas, la ejecución del programa de usuario, tarea que no es ejecutada en el *modo detenido*, y que resulta ser la diferencia más importante entre estos dos modos de operación.

El funcionamiento de los PLC's se basa en un esquema cíclico. El PLC ejecuta una lista de tareas, de forma secuencial y cíclica, un número indefinido de veces. La lista de tareas depende del modo de operación activo, pero en ambos modos de operación (detenido y de ejecución), el PLC siempre se mantiene en un ciclo de tareas. La velocidad a la que se realizan las tareas, que a su vez implica el tiempo que dura la ejecución de un ciclo completo, depende de la velocidad general del PLC. En esta velocidad influyen diversos parámetros, como la velocidad de ejecución de las instrucciones del PLC, el tiempo de acceso a entradas / salidas, el tiempo de comunicación con módulos de expansión y el tiempo de comunicación con periféricos, entre otras cosas.

En la figura 3.2 se muestra un modelo para el ciclo fundamental de los PLC's en el modo de ejecución, típicamente la diferencia con el modo detenido es que en este último no se realiza la fase de ejecución del programa de usuario.

Figura 3.2: "Modelo del ciclo fundamental de un PLC"



Generalmente, en la fase de lectura de las entradas, los PLC's únicamente leen las entradas físicas del tipo digital, y en la fase de escritura a las salidas, únicamente escriben a las salidas físicas del tipo digital. Es decir, que únicamente las entradas / salidas digitales instaladas en el sistema son automáticamente accedidas durante una fase específica dentro del ciclo del PLC. Todos los demás tipos de entradas / salidas son accedidas en el momento de ser requeridas como parámetros de las instrucciones del programa de usuario, al momento de la ejecución de las mismas, cuando el PLC se encuentre en el *modo de ejecución.*

Durante la fase de lectura de las entradas digitales, los PLC's leen el estado de las entradas físicas, y generan los denominados *registros de imagen del proceso* para las entradas, que consisten en los valores de todas las entradas digitales instaladas en el sistema. Luego, durante la fase de ejecución del programa, cuando las instrucciones requieren como parámetro el valor de estas entradas, el PLC no accede a las entradas físicas sino a los registros de imagen previamente generados. Cuando se ejecuta alguna instrucción que debe modificar alguna salida digital, ésta no es modificada directamente, puesto que el nuevo valor es escrito en el *registro de imagen* para las salidas digitales, que luego, durante la fase de actualización de las salidas, es trasladado a las salidas digitales físicas instaladas en el PLC.

3. El tiempo de respuesta. Este parámetro es función de diversas características internas del PLC, tanto intrínsecas a la arquitectura de éste, como a otras debidas al programa de usuario cargado en la memoria del PLC, o a las condiciones específicas de operación. Es importante debido a que determina el tiempo mínimo en que puede responder un PLC bajo ciertas circunstancias específicas, de tal forma que indica la máxima velocidad en que pueden ocurrir los eventos externos, del proceso de producción, para que éstos sean detectados y procesados correctamente por el PLC.

4. Las instrucciones y el lenguaje de los PLC's. Existen diversos lenguajes de programación que dependen del fabricante del PLC. Típicamente los fabricantes de PLC's mantienen un desarrollo continuo para las herramientas de automatización, entre

ellas se encuentra el lenguaje de programación, del cual surgen distintas versiones. Existen lenguajes tipo texto y tipo gráfico, dentro de los lenguajes tipo texto se encuentran muchos con similitudes a los lenguajes ensambladores de los microprocesadores, por otra parte, los lenguajes gráficos son cada vez más abstractos y de más alto nivel, aunque por lo general, esta abstracción tiene un precio: la pérdida de flexibilidad bajo ciertas circunstancias, es por ello, que en algunos casos, se prefiere utilizar los lenguajes tipo texto, y no los gráficos.

El conjunto de instrucciones típicas de un PLC incluye diversas instrucciones con muchas funciones, entre ellas, cargar relevadores de entrada (contactores), realizar operaciones booleanas, activar relevadores de salida (bobinas), mover datos entre localidades de memoria, realizar comparaciones de magnitud, manipular la pila lógica del PLC, operar con los contadores, activar / desactivar los temporizadores, incrementar / decrementar localidades de memoria, convertir entre diversos tipos de datos, rotar o desplazar bits, funciones matemáticas, y funciones especiales, según las capacidades del PLC. Las funciones matemáticas de los PLC's simples se limitan a sumar, restar, multiplicar y dividir números enteros, sin embargo, existen PLC's dotados con gran potencial matemático, que puede extenderse a muchas otras funciones matemáticas, incluyendo funciones trigonométricas, y con el manejo de números de punto flotante.

5. Las entradas y salidas de los PLC's. Los PLC's son dispositivos desarrollados para aplicaciones industriales, es por ello que deben estar protegidos contra diversas circunstancias. Una de las protecciones es tomada en cuenta en el diseño de las entradas y salidas digitales, ya sean incorporadas en el PLC, o en módulos de expansión. Todos los PLC's comerciales ofrecen aislamiento eléctrico entre los puntos de entradas / salidas y la lógica interna del sistema. Esto se logra utilizando un acoplamiento óptico con los puntos de conexión para los dispositivos de campo (detectores y actuadores). En el caso de puntos de entrada / salida implementados con relevadores, el aislamiento es el ofrecido por el propio elemento, sin embargo, cuando se utilizan transistores o triacs, estos van acoplados ópticamente con la electrónica interna, ya sea utilizando fototransistores o fototiristores, según sea requerido por la aplicación específica.

Los PLC's son dispositivos digitales. Los módulos de entradas / salidas son diseñados para aplicaciones específicas, según se requiera trabajar con voltajes de corriente continua o corriente alterna, con voltajes nominales típicos dentro del rango de 24 V a 220V. En todos los casos, los voltajes de entrada deben ser transformados en una señal de corriente continua, compatible con lógica TTL, para que puedan ser procesadas con la lógica del PLC. Los circuitos que realizan esta transformación son implementados como parte de los módulos que proveen los puntos de entradas / salidas.

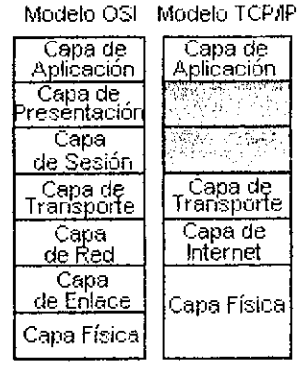
B. El modelo OSI

Un grupo llamado *OSI* (Interconexión de sistemas abiertos) creó un modelo que explica cómo debe trabajar una red. A este modelo se le llama *modelo de las siete capas OSI*, y es la base firme de la teoría de la conexión de redes.

El modelo OSI define en siete capas los protocolos de comunicación. Cada uno de los niveles tiene funciones definidas, que se relacionan con las funciones de las capas siguientes. Los niveles inferiores se encargan de acceder al medio, mientras que los superiores definen la forma en que las aplicaciones acceden a los protocolos de comunicación.

El OSI fue desarrollado como modelo de referencia, para la conexión de los sistemas abiertos. No es una arquitectura de red, pues no define qué aplicaciones ni protocolos usar, sino dice qué hace cada capa. El modelo OSI dio origen al modelo TCP/IP que se usa en la Internet.

Figura 3.3: "Modelo OSI y modelo TCP/IP"



1. Capa física. Se ocupa de la transmisión de los bits por el canal de comunicación. Esta es la encargada de garantizar que si un extremo envía un bit (con valor 0 o 1), éste llegue al otro extremo de la misma manera.

2. Capa de enlace. La función de esta capa es transformar un medio de transmisión común, en una línea sin errores de transmisión para la capa de red. Fracciona la entrada en tramas de datos y las transmite en forma secuencial. Establece los límites de la trama.

Cuando una trama es totalmente destruida por una ráfaga de ruido, la capa de enlace de la computadora emisora se encarga de retransmitirla. También se encarga de resolver la duplicidad de tramas, debido a que se puede destruir el acuse de recibo de la misma.

3. Capa de red. Se ocupa de controlar las operaciones de las subredes, resuelve cómo enviar los paquetes del origen al destino. Controla la congestión en la red ocasionada por la presencia de muchos paquetes. Esta capa resuelve los problemas de comunicación, que resultan de unir redes heterogéneas, que manejan diferentes protocolos y tienen formas diferentes de direccionamientos.

4. Capa de transporte. La función de esta capa es aceptar los datos de la capa de sesión, dividirlos si es necesario y pasarlos a la capa de red y asegurarse que lleguen correctamente al destino. Esta capa crea una conexión de red, distinta para cada conexión de transporte solicitada por la capa de sesión. Si el caudal es grande puede realizar más de una conexión para mejorarlo. La capa de transporte se encarga de establecer y liberar conexiones en la red.

La conexión más conocida es el canal punto a punto sin error, en el cual se entregan los mensajes en el mismo orden que fueron enviados. Otra forma del servicio de transporte es el envío de mensajes aislados, que no garantizan el orden de difusión, ni la distribución de mensajes a destinos múltiples. El famoso protocolo TCP utilizado en la Internet pertenece a esta capa.

5. Capa de sesión. Permite que usuarios en distintas computadoras establezcan una sesión entre ellos, a través de la misma se puede llevar a cabo un transporte de datos, tal como lo hace la capa de transporte. La mejora de los servicios le permite al usuario acceder a un sistema de tiempo compartido a distancia o transferir un archivo. Entre los servicios de esta capa se encuentran: control de diálogo, administración de testigo y sincronización.

a. Control de diálogo. Las sesiones permiten que el tráfico se realice en ambas direcciones o en una sola en un momento dado, cuando se realiza en un solo sentido, esta capa ayudará en el seguimiento de quién tiene el turno.

b. Administración de testigo. Esto es para que en algunos protocolos los dos extremos no intenten transmitir al mismo tiempo, de esta forma sólo lo hace el que posee el testigo (token).

c. Sincronización. Proporciona la inserción de puntos de verificación para el control de flujo. Si dos computadoras desean transmitir un archivo que lleva dos horas, y al cabo de una hora se interrumpen las conexiones de red, la transmisión se debe desarrollar nuevamente desde el principio, con el servicio que brinda esta capa sólo se transmite lo posterior al punto de verificación.

6. Capa de presentación. Esta capa no cumple el mismo tipo de funciones que las anteriores, las cuales se encargaban de la transmisión fiable de los bits, sino que se ocupa de la sintaxis y la semántica de la información.

7. Capa de aplicación. Esta capa proporciona acceso al entorno OSI para los usuarios, y ofrece servicios de información distribuida. Contiene una gran variedad de protocolos que son usados frecuentemente. Contiene los programas de los usuarios (aplicaciones). Dos programas bastante conocidos son la transferencia de archivos (FTP) y el acceso de archivos remotos (TELNET).

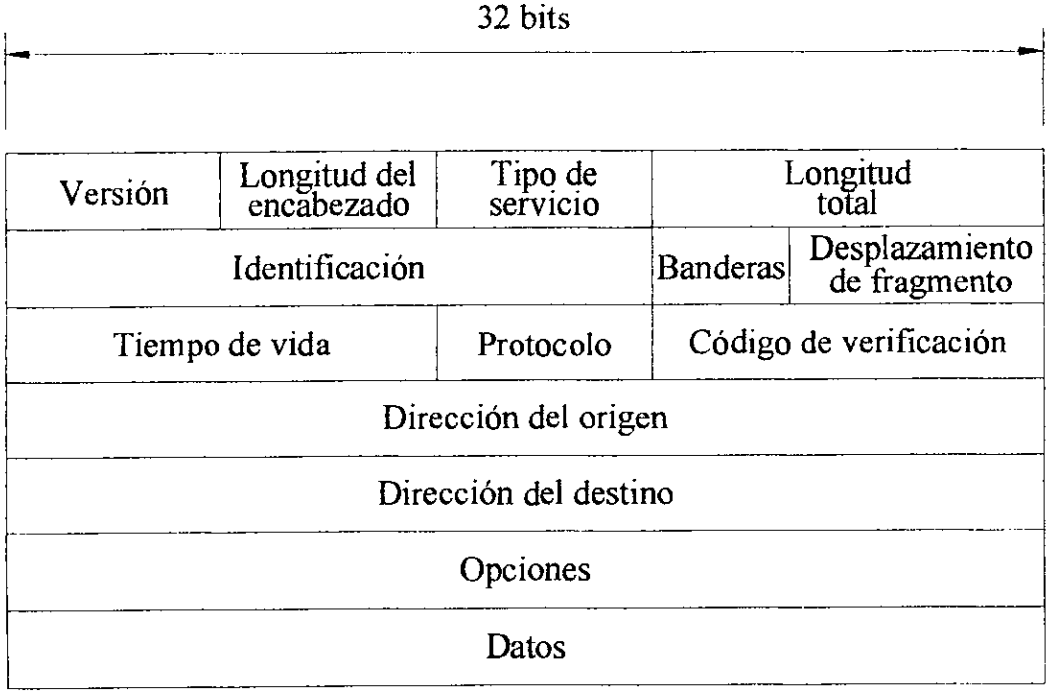
C. Los protocolos de la Internet

La familia de protocolos de la Internet es el conjunto de protocolos de sistema abierto (sin propietario) más popular alrededor de todo el mundo. Esta familia consiste en un conjunto de protocolos de comunicación, de los cuales los dos más conocidos son el *protocolo de control de transmisión* (TCP) y el *protocolo de Internet* (IP).

1. El protocolo de Internet (IP). La unidad de información que maneja este protocolo es llamada *paquete*. Este es un protocolo de red que contiene información de direccionamiento y alguna información de control que permite el enrutamiento (distribución) de los paquetes de datos a través de las redes de computadoras.

a. El formato de un paquete IP. El paquete IP consta de diversos campos, en los que se incluye la información necesaria para la realización y control básico del enrutamiento a efectuar. La figura 3.4 muestra el formato de un paquete IP.

Figura 3.4: "Formato de un paquete IP"



1) Versión: Indica la versión del protocolo IP actualmente en uso.
2) Longitud del encabezado IP: Indica la longitud del encabezado, en palabras de 32 bits.
3) Tipo de servicio: Especifica cómo debe ser utilizada la información del datagrama actual, y asigna niveles de importancia.
4) Longitud total: Especifica la longitud total del paquete IP, en bytes, tomando en cuenta la longitud del encabezado y de los datos.
5) Identificación: Es un número entero que identifica al datagrama actual. Es utilizado para unir en orden a todos los datagramas que se transmiten.
6) Banderas: Consisten en tres bits que indican opciones de fragmentación para el paquete actual.
7) Desplazamiento de fragmento: Indica la posición del fragmento de datos actual, que tenía en el datagrama original, relativa al inicio de éste.
8) Tiempo de vida: Es un contador que gradualmente decrementa su valor, hasta llegar a cero, momento en el cual se descarta el paquete. Esto evita que los paquetes no sean distribuidos debido a la ocurrencia de un ciclo infinito durante su enrutamiento.
9) Protocolo: Indica qué tipo de protocolo de nivel superior recibe los paquetes entrantes, luego de que se complete el procesamiento IP.
10) Código de verificación del encabezado: Es un valor utilizado para verificar la integridad de la información en el encabezado.
11) Dirección de origen: Especifica la dirección del nodo remitente.
12) Dirección destino: Especifica la dirección del nodo destino.
13) Opciones: Permite que el IP sea compatible con diversas opciones, como por ejemplo seguridad.

b. Direccionamiento IP. El direccionamiento se lleva a cabo por medio de las llamadas *direcciones IP*. Estas direcciones poseen un formato básico y pueden ser subdivididas para la creación de subredes lógicas. En una red TCP/IP se asigna a cada

cliente una dirección lógica única, de 32 bits. Esta dirección se divide en dos partes principales: El número de red y el número de cliente. El número de red identifica a una red específica, y si ésta va a formar parte de la Internet, este número debe ser asignado por el Centro de Información de Redes de Internet (InterNIC).

Los proveedores de servicios de Internet pueden obtener bloques de direcciones de Internet, asignadas por el InterNIC, y distribuirlas por sí mismas. El número de cliente identifica a un cliente específico dentro de una red. Este número es asignado por el administrador de la red local.

La dirección IP está agrupada en cuatro segmentos de 8 bits cada uno, separados por un punto y representados en formato decimal. Cada grupo de 8 bits es llamado *octeto*. El valor mínimo para cada octeto es 0, y el valor máximo es 255.

c. Direccionamiento de subredes. Las redes IP pueden ser divididas en redes más pequeñas, llamadas *subredes*. La utilización de subredes provee diversos beneficios al administrador de la red. Entre los beneficios se encuentran la mayor flexibilidad y eficiencia en el uso de las direcciones de red. Las subredes son administradas localmente, de tal forma que el "mundo exterior" ve a una organización como una única red, y no posee conocimiento de los detalles acerca de la estructura interna de ésta. La dirección de subred es creada a partir del préstamo de bits tomados del campo *cliente*, estos bits se transforman en el campo de subred. El número de bits prestados es variable, y se encuentra determinado por la *máscara de subred*. La máscara de subred posee unos en todos los bits que identifican al número de red y subred, y ceros en todos los demás bits, que identifican al número de cliente dentro de la subred.

2. El protocolo de resolución de direcciones (ARP). Para que se pueda establecer comunicación entre dos máquinas conectadas a una red, es necesario que cada una de ellas conozca la dirección física de la otra. A esta dirección física se le llama *dirección MAC*, y corresponde a un número único, de 48 bits, asignado a cada circuito integrado controlador de red, y que es almacenado en la memoria de éste, durante su

pueden ser retransmitidos. El mecanismo de TCP permite que los dispositivos puedan tratar con paquetes dañados, perdidos, duplicados o retrasados. Un mecanismo de tiempo límite permite que los dispositivos detecten los paquetes perdidos y soliciten su retransmisión.

TCP ofrece un control de flujo eficiente en cuanto a que cuando se realiza la notificación de un segmento de bytes recibidos, el proceso TCP receptor indica al proceso TCP emisor, el número de secuencia más alto que puede recibir sin que se provoque un desborde en sus registros de almacenamiento temporal.

La multiplexación se refiere a que con una sola conexión, pueden establecerse diversas transacciones, de forma simultánea, sobre protocolos superiores.

a. Establecimiento de la conexión TCP. Para la utilización de servicios de transporte confiables, un cliente TCP debe establecer una sesión orientada a conexión con otro cliente TCP. El establecimiento de esta conexión es llevado a cabo por medio de un mecanismo especial. Primero, ambos extremos de la conexión se sincronizan por medio de la selección de sus números iniciales para la secuenciación de los bytes a transmitir. Esto garantiza que ambos lados estén listos para la transmisión de datos. La selección del número de secuenciación es de forma aleatoria, cada cliente selecciona uno, con el cual rastreará los bytes que transmita / reciba. Luego de la selección de los números de secuenciación, el mecanismo procede de la siguiente forma: El primer cliente (cliente A) inicia una conexión por medio del envío de un paquete con el número de secuenciación inicial seleccionado (X), y un bit de bandera denominado "SYN", colocado en uno para indicar la solicitud de conexión. El segundo cliente (cliente B) recibe la señalización SYN y almacena el número de secuenciación X, y responde por medio de una notificación de recibo, con el valor ACK=X+1, incluyendo además, su propio número de secuenciación inicial (Y). Luego el cliente A realiza la notificación de recibo de todos los bytes enviados por el cliente B, indicando el número del próximo byte a recibir, por medio de la notificación con el valor ACK= Y+1. Con esto la conexión se encuentra establecida y la transferencia de datos puede iniciar.

fabricación, previo a su instalación en una tarjeta de red. Para que una máquina adquiera la dirección física de otra, se utiliza el envío de mensajes con el protocolo de resolución de direcciones, denominados *ARPs*, en los cuales, se envía la solicitud de la dirección física requerida, correspondiente con una dirección IP particular. Luego de recibir la respuesta conteniendo la dirección MAC del dispositivo con quien se desea establecer comunicación, los dispositivos IP crean una tabla en memoria, conteniendo las parejas de direcciones IP y MAC, de tal forma que no se requiera reenviar ARPs la próxima vez que se desee establecer comunicación con la misma máquina. Si una máquina no responde durante un tiempo definido, su pareja IP – MAC es borrada de la tabla.

3. **Protocolo de mensajes de control de Internet (ICMP).** Este protocolo provee paquetes de mensajes acerca de los errores, e información adicional, durante el procesamiento de paquetes IP. Estos mensajes son enviados hacia el remitente de los paquetes que sufrieron errores. Los mensajes generados son conocidos como *Mensajes ICMP*.

4. **Protocolo de control de transmisión (TCP).** Este protocolo ofrece la transmisión confiable de datos en un ambiente IP. Entre los servicios que provee este protocolo se encuentran la transferencia de cadenas de datos, control de flujo eficiente, confiabilidad, operación full-duplex, y multiplexación.

Con la transferencia de cadenas de datos, el protocolo TCP entrega cadenas no estructuradas de bytes identificados por números secuenciales. Con este servicio, las aplicaciones no necesitan partir en bloques los datos que enviarán al protocolo TCP. En lugar de ello, el protocolo TCP agrupa los bytes en segmentos, y los envía al protocolo IP para su enrutamiento.

El protocolo TCP basa la confiabilidad de la transmisión de los paquetes en el establecimiento de una conexión punto a punto. TCP utiliza un sistema de numeración secuencial para los bytes que transmite, y una notificación de recibo para ellos, de tal forma que luego de un tiempo determinado, los bytes no notificados como recibidos

b. Notificación positiva de recibo y retransmisión (PAR). Un protocolo de transporte simple puede utilizar una técnica de control de flujo en la cual el transmisor envía un paquete, inicializa un temporizador y espera una notificación de recibo antes de enviar el próximo paquete. Si la notificación de recibo no llega luego de cierto tiempo, el temporizador expira y el paquete es retransmitido. Una técnica como ésta es llamada *notificación positiva de recibo y retransmisión (Positive acknowledgement and retransmisión, PAR).* Esta técnica implica un uso ineficiente del ancho de banda, puesto que el cliente debe esperar la notificación de recibo para poder enviar un nuevo paquete, y solo un paquete puede ser enviado a la vez.

c. La ventana deslizante TCP. Esta es una técnica de transporte con la cual se logra hacer más eficiente el uso del ancho de banda en la red, comparado con la técnica PAR. Esto se logra habilitando a los clientes para que envíen múltiples bytes o paquetes antes de esperar alguna notificación de recibo. A este conjunto de múltiples bytes se le denomina *ventana deslizante.*

Puesto que TCP provee una conexión con transporte de cadenas de bytes, los tamaños de las ventanas también son expresados en bytes. El tamaño de una ventana indica al transmisor el número autorizado de bytes de datos para enviar, antes de esperar por una señal de recibo. Los tamaños iniciales de las ventanas son indicados durante la negociación de la conexión, y estos pueden variar durante la transferencia de datos, con lo cual se implementa el control de flujo deseado. Por ejemplo, una ventana con tamaño igual a cero indica que no se deben transmitir bytes.

d. Formato de un paquete TCP. En la figura 3.5 se presenta el formato de campos que componen un paquete TCP.

1) Puerto origen y puerto destino: Identifican los puntos en los cuales los procesos superiores al protocolo TCP, reciben los servicios de éste.
2) Número de secuenciación: Usualmente especifica el número asignado al primer byte de datos en el mensaje actual. Durante la fase de

establecimiento de la conexión, este campo es utilizado para especificar el número de secuenciación inicial.

3) Número de notificación de recibo: Contiene el número de secuenciación del siguiente byte de datos que el transmisor del paquete actual espera recibir.
4) Desplazamiento de datos: Indica la cantidad de palabras de 32 bits contenidas en el encabezado TCP.
5) Reservado: Se mantiene reservado para uso futuro.
6) Banderas: Ofrecen información de control, incluyendo los bits SYN, ACK, utilizados para el establecimiento de la conexión, y el bit FIN, utilizado para la finalización de la misma.
7) Ventana: Especifica el tamaño de la ventana receptora para el transmisor del paquete actual.
8) Código de verificación: Utilizado para verificar la integridad de los campos del encabezado TCP.
9) Puntero de urgencia: Apunta al primer byte de datos urgente en el paquete.
10) Opciones: Especifica varias opciones de configuración TCP.

Figura 3.5: "Formato de un paquete TCP"

<table>
<tr><td colspan="3">Puerto origen</td><td>Puerto origen</td></tr>
<tr><td colspan="4">Número de secuencia</td></tr>
<tr><td colspan="4">Número de notificación de recibo</td></tr>
<tr><td>Desplazamiento de datos</td><td>Reservado</td><td>Banderas</td><td>Ventana</td></tr>
<tr><td colspan="3">Código de verificación</td><td>Puntero de urgencia</td></tr>
<tr><td colspan="4">Opciones</td></tr>
<tr><td colspan="4">Datos</td></tr>
</table>

5. Protocolo de datagramas de usuario. Este es un protocolo no orientado a conexión. Básicamente es una interfaz entre IP y procesos superiores. A diferencia de TCP, UDP no ofrece confiabilidad, control de flujo o corrección de errores. Debido a la simplicidad del UDP, sus encabezados poseen menos bytes, y consume menos recursos de red que TCP. UDP es útil en situaciones donde no se requiere control de errores y de flujo, debido a que estas características sean ofrecidas por algún otro protocolo superior.

Este es el protocolo de transporte utilizado en diversos protocolos de aplicación conocidos, como DNS (Domain Name System), NFS (Network File System), SNMP (Simple Network Management Protocol) y TFTP (Trivial File Transfer Protocol). El encabezado de los paquetes UDP posee cuatro campos, tal como se muestra en la figura 3.6.

Figura 3.6: "Encabezado de un paquete UDP"

Puerto origen	Puerto destino
Longitud	Código de verificación

Los campos para puertos de origen y destino poseen los números de 16 bits para los puertos del protocolo UDP. Estos puertos son utilizados para demultiplexar los datagramas recibidos por los protocolos de aplicación. El campo de longitud especifica el tamaño de todo el paquete UDP. El último campo es utilizado para verificar la integridad del paquete UDP (encabezado y datos).

D. La tecnología Ethernet

En términos generales, *Ethernet* es un sistema para el transporte digital de datos a través de sistemas de cómputo local. Ethernet es una tecnología de transmisión de datos de alta velocidad que fue inventada en 1973. En 1980, Digital Equipment Corporation

(DEC), Intel y Xerox, desarrollaron el hardware para Ethernet a 3 Mbps, el cual ganó gran aceptación en el mundo de la computación.

Ethernet es compatible con una topología de bus y usa un canal compartido de comunicaciones, manejado por acceso múltiple por percepción de portadora con detección de colisiones (CSMA/CD). CSMA/CD no tiene manejo de control central de acceso al canal. Si una estación desea transmitir a otro lado, tiene que esperar hasta que pueda adquirir el canal. Una vez adquirido este canal, la estación transmite a través de éste.

Para adquirir el canal, la estación determina si la red esta ocupada (usa detección de portadora) y espera a transmitir el paquete hasta que la red se haya liberado. Cuando esto es detectado, la estación empieza inmediatamente a transmitir. Durante la transmisión, la estación está atenta a posibles colisiones (por ejemplo, alguna otra estación que quiera transmitir). Una colisión podrá ocurrir en un corto intervalo de tiempo al inicio de la transmisión, si no ocurren colisiones durante este corto intervalo de tiempo, la estación ha adquirido el canal y continua transmitiendo. Si ocurre una colisión, cada una de estas transmisiones se hace posteriormente.

La implementación Ethernet está definida como un estándar del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y es llamada IEEE 802.3.

Ethernet se divide a sí misma en dos componentes o capas. La capa fisica y la capa de enlace de datos. La capa física describe las características físicas de la red y el hardware usado. Esta capa incluye: topología, hardware de transmisión, equipo utilizado, etc.

La capa de enlace de datos es la responsable del intercambio de datos entre un host cualquiera y la red Ethernet a la que está conectado, permitiendo la correcta comunicación y trabajo conjunto entre las capas superiores (Red, transporte y aplicación) y el medio físico de transporte de datos. Su principal objetivo es proporcionar una

comunicación eficiente, libre de errores, entre dos máquinas adyacentes, pertenecientes a la misma red / subred Ethernet.

Todas las topologías básicas de Ethernet se describen en el estándar 802.3 del IEEE. Los miembros principales de esta familia se listan a continuación:

- **10Base2.** Realiza las conexiones utilizando cable coaxial. La longitud máxima de un segmento 10base2 es de 185 metros.
- **10Base5.** Éstas utilizan un cable más grueso que 10base2. La longitud máxima de un segmento 10base5 es de 500 metros.
- **10Base-T.** Utiliza cable tipo UTP, que consiste en cuatro pares trenzados sin blindaje. Únicamente se utilizan dos pares para la transmisión y recepción de datos. En 10Base-T la longitud máxima desde un concentrador hasta la estación de trabajo es de 100 metros.

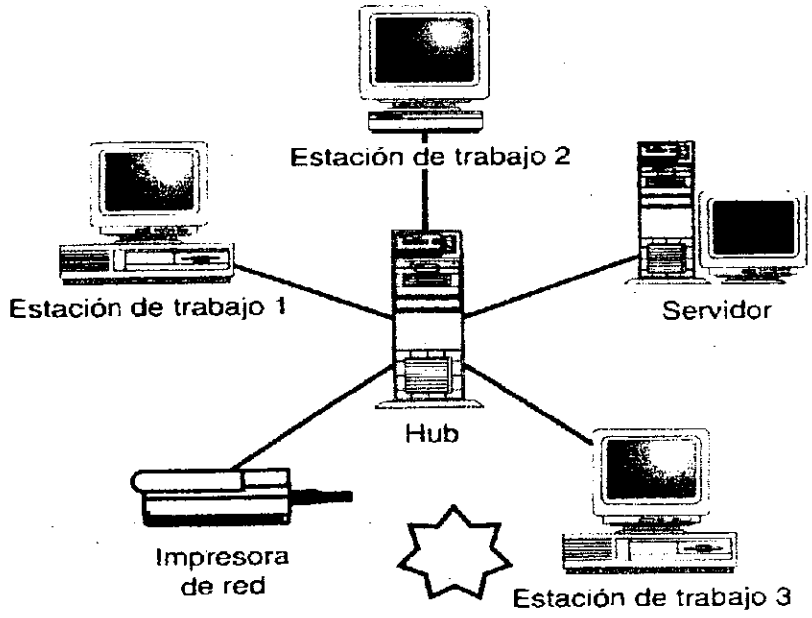
La tecnología Ethernet ha seguido evolucionando, incluyendo redes más rápidas y medios de transmisión basados en fibra óptica. Los nuevos miembros de la familia Ethernet se describen en el estándar 802.3u del IEEE e incluye los siguientes:

- **100Base-T.** También conocido como *Fast Ethernet*, en el que los datos viajan a 100 millones de bits por segundo. Se utiliza el mismo cable que en 10Base-T. La longitud máxima del cable entre el concentrador y la estación de trabajo es de 20 metros.
- **100Base-FX.** Equivale a 100Base-T, pero ésta no utiliza pares trenzados de cobre, en cambio, utiliza fibras ópticas.
- **100Base-T4.** Equivale a 100Base-T, pero ésta utiliza los cuatro pares trenzados.

La tarjeta de red es un elemento importante en los sistemas Ethernet, pues ésta permite la conexión de una computadora con la red. Básicamente, la tarjeta de red Ethernet se compone de un circuito integrado llamado *controlador Ethernet*, el cual se

encarga del manejo de todas las tramas Ethernet. Además posee un transformador que acopla impedancias, y un receptáculo RJ-45 para la conexión del cable.

Figura 3.7: "Topología de una red Ethernet"



E. Tecnologías para la habilitación de la Internet en los sistemas microcontrolados

Para que un sistema microcontrolado pueda conectarse con la Internet, existen básicamente dos esquemas:

En el primero se utiliza una computadora como puerta de enlace para el sistema microcontrolado, en este esquema, la computadora que posee conexión con la Internet funciona como una interfaz entre la Internet y el sistema microcontrolado, el cual se encuentra conectado de alguna forma con la computadora, por medio de un puerto, una tarjeta u otro equipo especial adicional. En este esquema existen diversas desventajas,

entre ellas se encuentran el costo del sistema completo, la poca flexibilidad, la dependencia con respecto al sistema computarizado de interfaz, las dimensiones físicas y la complejidad del sistema completo.

En el segundo esquema se trata un concepto más sofisticado, la *habilitación de la Internet en los sistemas microcontrolados.* Aquí la idea fundamental es implementar todos los protocolos de red necesarios para la conexión con Internet, dentro del mismo sistema microcontrolado, desarrollando lo que se conoce como un *sistema incrustado* (Embedded System).

En la actualidad existen diversas soluciones para la implementación de un sistema incrustado, desarrolladas por diversas empresas alrededor de todo el mundo, entre ellas Microchip Inc., Embedded Ethernet, Yipee Inc., Iosoft Ltd., LiveDevices, NetMedia Inc., EmWare y CMX. Todas estas soluciones tienen en común la utilización de un *stack de TCP/IP.*

El stack de TCP/IP es la implementación del conjunto fundamental de los protocolos de la Internet, en un segmento de código ensamblador, o algún código de alto nivel, como el lenguaje C, y que ha sido desarrollado para su inclusión dentro del código fuente de cualquier aplicación específica de un sistema microcontrolado. El stack de TCP/IP funciona como un subprograma dentro del sistema microcontrolado completo.

Los stacks de TCP/IP pueden ser fusionados con el programa de la aplicación específica, en la memoria interna de un microcontrolador central, o puede implementarse como un programa utilitario, cargado en la memoria de un dispositivo adicional, que se comunica con el microcontrolador central por medio de un canal o bus de datos.

Los stacks de TCP/IP generalmente ofrecen rutinas para la recepción y transmisión de datos, que están diseñadas para ser utilizadas como subrutinas dentro de un programa principal, desarrollado como parte del sistema microcontrolado, para una aplicación específica.

Todos los parámetros de configuración, como las direcciones IP, MAC, etc., son manejados por medio de registros especiales, que deben ser definidos dentro del sistema de software del controlador central.

En general, existen muchos detalles acerca de la interacción entre los stacks de TCP/IP y el software de un sistema microcontrolado, sin embargo esos detalles se refieren a la programación y configuración del stack y la aplicación específica, y dependen de las herramientas que se utilicen para su desarrollo, como por ejemplo, la familia de microcontroladores, el lenguaje de desarrollo, los medios de comunicación del sistema (paralelo, RS-232, I^2C, SPI, etc.), entre otras cosas.

Lo último en estas tecnologías es el concepto de un servidor HTTP incorporado dentro del sistema microcontrolado. Esta tecnología implementa un servidor HTTP básico en un microcontrolador, de tal forma que el microcontrolador central de la aplicación específica pueda comunicarse directamente con el microcontrolador del servidor HTTP, el cual a su vez posee incorporado un stack de TCP/IP. Esto permite que el sistema microcontrolado se convierta en un servidor HTTP, conectado a la Internet, y que realiza alguna tarea específica, que es precisamente la función del sistema microcontrolado original. Dicho en otras palabras, con esta tecnología se puede "navegar" en el sistema microcontrolado de la aplicación específica.

F. Conceptos útiles de interacción con servidores HTTP.

1. Servidores HTTP. Se encargan de que los documentos de hipertexto estén disponibles para un usuario a través de una máquina cliente. Los servidores HTTP reciben las llamadas de los clientes y proporcionan la información deseada. Bajo ciertas peticiones, los servidores HTTP poseen la capacidad de ejecutar programas especiales, que interactúan con el cliente y con datos del propio servidor, con lo cual generan datos de forma dinámica, como por ejemplo, páginas HTML capaces de responder según las acciones de la persona en el lado del cliente.

2. CGI. Se denomina CGI (interfaz de pasarela común) al conjunto de normas que rigen la interacción entre los servidores HTTP y un conjunto de programas especiales que se ejecutan en el servidor, destinados a procesar información que envían los clientes y generar documentos de forma dinámica (generalmente HTML) a partir de esos datos. Los módulos CGI tienen muchas aplicaciones, una de las más utilizadas es la de recoger y procesar información que un cliente envía desde un formulario o enlaces HTML.

Los clientes utilizan los comandos GET y POST del protocolo HTTP para enviar información hacia un servidor. Cuando un cliente solicita un documento en cuya dirección se incluye el nombre de una aplicación CGI, éste hace una división de los parámetros contenidos en la dirección completa. De tal forma que todo lo que se encuentre después del símbolo "?", será interpretado como parámetros de entrada para la aplicación CGI, claro que con ciertos criterios de interpretación según la aplicación CGI. Al recibir los parámetros de entrada, la aplicación CGI los procesa y genera el resultado, que puede consistir en una página HTML, con lo cual se presenta un entorno dinámico, de interacción entre los datos del servidor, y los datos enviados por el cliente.

IV. SISTEMA DE HARDWARE DEL iPLC

Se deseaba desarrollar un sistema de hardware con una arquitectura cuyo diseño permitiera la interacción de un controlador central con diversos dispositivos especializados, y que ofreciera la capacidad de expansión. Es por ello que se realizó el diseño de una "tarjeta madre", la cual constituye a la CPU central del PLC, que posee buses de datos y de direcciones para comunicarse con diversos dispositivos periféricos, y un bus adicional especial, denominado *bus de expansión*, con el cual se logra que el sistema del iPLC sea expansible.

Se decidió diseñar e implementar dos módulos de expansión, uno de tipo digital, con ocho entradas y ocho salidas, y otro de tipo analógico, con dos entradas y una salida. La arquitectura de estos módulos está basada en la del *módulo de expansión modelo*, que consiste en un diseño generalizado, para que sea utilizado como el núcleo de cualquier módulo de expansión que se diseñe posteriormente, y que ofrezca cualquier tipo de servicios a la CPU central. Con el diseño de este módulo de expansión fundamental se logró desarrollar un sistema de PLC completo, en el que además de ofrecer la unidad central (PLC propiamente), también se encuentra ya definida la arquitectura de todos los dispositivos que se puedan conectar a ésta. La arquitectura del iPLC se basa en la centralización del control de las tareas, y la distribución de las tareas y funciones del sistema. Para la centralización del control se utilizó un microcontrolador PIC16F877 (controlador central), al cual se conectaron todos los dispositivos periféricos del sistema del iPLC, por medio de buses y canales de datos con diferentes protocolos. A continuación se expone con mayor detalle la arquitectura de la CPU central (tarjeta madre), y de los módulos de expansión.

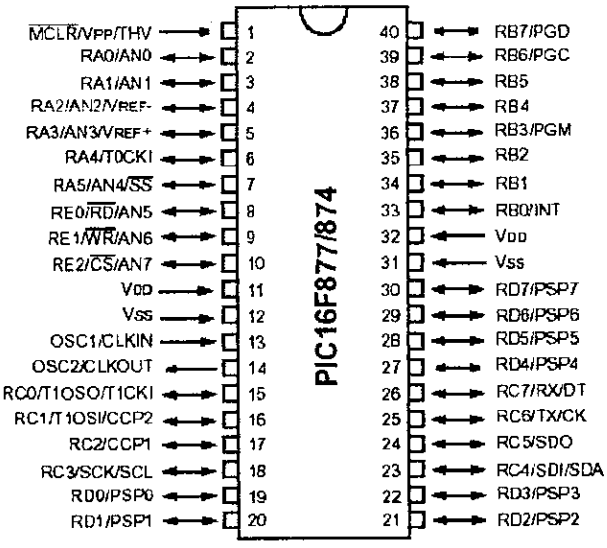
A. Controlador central

Para el desarrollo del "corazón" de la CPU central (PLC) se eligió utilizar un microcontrolador PIC16F877 (ver U1 en la figura A-1, en el apéndice A), al cual se le

cargó el software completo del PLC, funcionando así como el firmware del sistema. El microcontrolador central es el encargado de comunicarse con todos los periféricos de la tarjeta madre del PLC, y además realiza las tareas de comunicación con los módulos de expansión instalados (que previamente han sido detectados). Dentro de este microcontrolador central se encuentra toda la funcionalidad y estructura lógica del PLC.

Todos los periféricos de la tarjeta madre se encuentran mapeados a este controlador, que únicamente centraliza el control de las tareas, pero las delega en todos sus dispositivos esclavos. Para la operación del controlador central se utilizó un reloj con frecuencia de oscilación f_{OSC} = 20 MHz, que es la frecuencia máxima para el reloj que temporiza y sincroniza el funcionamiento interno del PIC16F877. Se eligió esa frecuencia máxima ya que con ello también se logra la máxima velocidad de operación del sistema completo, pues cuanto más rápida sea la operación del controlador central, más rápidos pueden ser los canales y buses de comunicación, así como también se hace más rápida la ejecución de todas las fases de operación del PLC (que se encuentran en este controlador, como parte del firmware del iPLC). Se utilizaron los periféricos internos en el controlador central (PIC16F877) para establecer la comunicación con todos los periféricos externos a éste, que consisten en los dispositivos especializados instalados en la tarjeta madre del sistema. Estos dispositivos se presentan en las secciones siguientes.

Figura 4.1: "Controlador central PIC16F877"

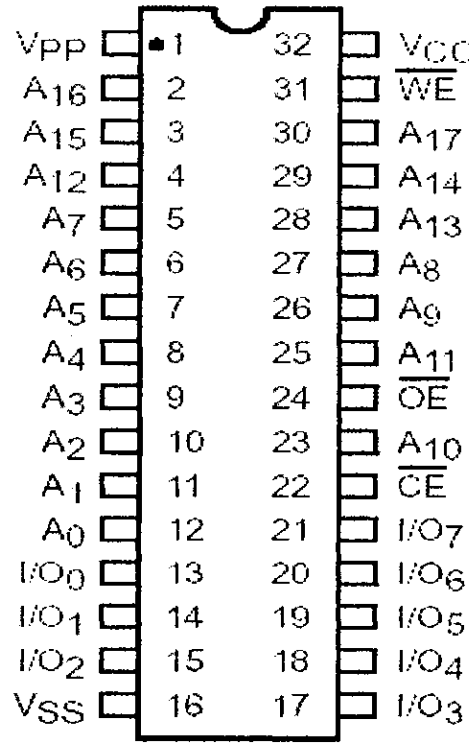


B. Memoria flash

El iPLC posee capacidad de direccionamiento para 64 K bytes de memoria de programa y una capacidad instalada de 256 K bytes prevista para ampliaciones futuras. El programa de usuario es almacenado en la memoria flash de circuito integrado CAT28F020 (U2 en la figura A-1 del apéndice A), que fue elegida básicamente por su capacidad de almacenamiento (256 K bytes), su velocidad (200nS para el ciclo de lectura), y su desempeño en cuanto a la cantidad de ciclos de escritura que soporta (un mínimo de 100,000).

Este dispositivo de memoria flash ofrece al sistema un medio de almacenamiento permanente para el programa de usuario cargado al PLC, con suficiente capacidad para los requerimientos típicos de los PLC's comerciales. En esta memoria se encuentran almacenados todos los códigos de programa, que son leídos por el controlador central para la ejecución del programa de usuario.

Figura 4.2: "Memoria flash CAT28F020"



El circuito integrado CAT28F020 posee los pines típicos de cualquier dispositivo paralelo: A0-A17 para el bus de direcciones, I/O0-I/O7 para el bus de datos, las señales de control de escritura (WE'), lectura (OE') y selección de dispositivo (CE'). El pin V_{PP} es utilizado como señal de control para habilitar la grabación de datos. Esta señal es activa en alto (12V).

C. Memoria RAM

Para el almacenamiento de las variables del sistema y de las variables del programa de usuario, la tarjeta madre posee la memoria RAM de circuito integrado CY7C199 (U3), con una capacidad de 32 K bytes y un tiempo de acceso de 15 nS. Esta memoria fue seccionada a nivel lógico para el mapeo de las diversas áreas de memoria que debe poseer el PLC. En total son once áreas de memoria.

1. Área de memoria de usuario (memoria V). A esta región le fue asignada una capacidad de 28 KB, pues es la región que el usuario utiliza para el almacenamiento de variables en sus programas.

2. Área de imágenes del proceso para las entradas digitales (DI). A esta región se le asignaron siete bytes para el almacenamiento de las entradas de siete módulos de expansión, de forma automática durante el inicio de cada ciclo del PLC.

3. Área de imágenes del proceso para las salidas digitales (DQ). A esta región se le asignaron siete bytes para el almacenamiento de los estados de las salidas de siete módulos de expansión, que se actualizan de forma automática durante la ejecución del programa de usuario en la fase correspondiente dentro de cada ciclo del PLC.

4. Área de entradas analógicas (memoria AI). A esta región le fueron asignados 240 bytes para el mapeo de hasta 16 entradas analógicas en cada uno de los 15 módulos de expansión. En total se puede direccionar hasta 240 entradas analógicas.

5. Área de salidas analógicas (memoria AQ). A esta región le fueron asignados 240 bytes para el mapeo de hasta 16 salidas analógicas en cada uno de los 15 módulos de expansión. En total se pueden direccionar hasta 240 salidas analógicas.

6. Área de marcas especiales (memoria SM). Se asignaron siete bytes en esta región para el almacenamiento de las siete marcas especiales del sistema del PLC. Estas marcas especiales son:

a. SM0. Byte con todos sus bits siempre iguales a uno (1). Se creó esta marca especial por compatibilidad con los PLC's de Siemens, y la estructura de su lenguaje de programación, pues esta marca especial resulta necesaria si se desean ejecutar instrucciones de forma incondicional.

b. SM1. Byte que posee todos sus bits iguales a uno (1) únicamente durante el primer ciclo del PLC. Marca especial útil para tareas de inicialización.

c. SM2. Byte que almacena el tiempo del último ciclo del PLC (en milisegundos).

d. SM3. Byte que almacena el tiempo del ciclo más corto que el PLC a ejecutado desde que entró en el *modo de ejecución* (tiempo en milisegundos).

e. SM4. Byte que almacena el tiempo del ciclo más largo que el PLC a ejecutado desde que entró en el *modo de ejecución* (tiempo en milisegundos).

f. SM5. Contiene el byte de estado del PLC, con dos bits para el modo de operación, y cuatro bits para los códigos de error.

g. SM6. Marca especial tipo byte que asume el valor complementario de su valor anterior al inicio de cada ciclo del PLC. Genera una señal cuadrada (en software),

que inicia con valor alto (255_d) para el primer ciclo, y asume un valor bajo (0) para el ciclo siguiente, complementándose continuamente según los ciclos ejecutados.

7. Área de contadores (memoria C). A esta región se le asignaron 80 bytes para mapear los 40 contadores que posee el sistema, de 16 bits cada uno de ellos.

8. Área de temporizadores (memoria T). A esta región se le asignaron 192 bytes para el mapeo de los 96 temporizadores que posee el sistema, de 16 bits cada uno de ellos.

9. Área de registros del reloj de tiempo real (memoria R). A esta región se le asignaron trece bytes para el mapeo de todos los registros del reloj de tiempo real utilizado. Esta región es escrita / leída cuando se realiza alguna instrucción con el reloj de tiempo real.

10. Área de la pila del contador de programa (PC Stack). A esta región le fueron asignados 32 bytes para el almacenamiento de 16 posiciones de memoria de programa, que se almacenan para poder retornar a una posición específica, dentro del código del programa de usuario, luego de haberse trasladado desde ésta por medio de la utilización de alguna de las instrucciones de control del programa (JMP, CALL). Es importante especificar que esta región de memoria es no accesible por el usuario, es decir, que no puede ser accedida con las instrucciones disponibles desde el programa de usuario.

11. Área de registros de flancos. A esta región le fueron asignados 512 bytes para el almacenamiento de los registros de memoria del estado de bits específicos dentro de la memoria de usuario, utilizados por diversas instrucciones de programa, para determinar cuando se provocan flancos ascendentes / descendentes en dichos bits.

Se mapearon 256 bytes para registros de flancos ascendentes y 256 bytes para registros de flancos descendentes. Esta región es no accesible por el usuario.

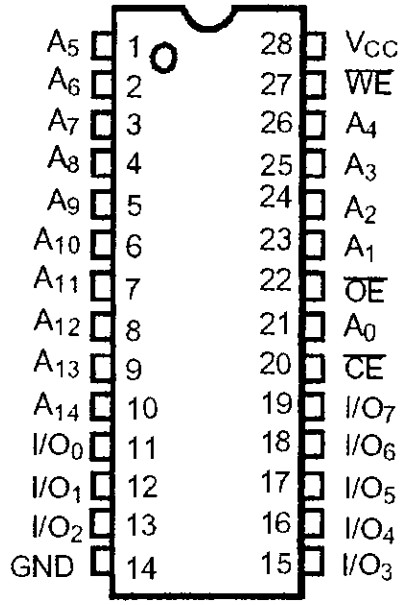
Tabla 4.1: "Mapeo de todos los tipos de memoria del sistema del PLC en el circuito integrado RAM externo CY7C199 de 32 K Bytes"

Localidad	Tipo de Memoria Mapeada
0x0020 0x0026	Imágenes del proceso de las entradas digitales (DI)
0x0027 0x002D	Imágenes del proceso de las salidas digitales (DQ)
0x006A 0x0070	Marcas especiales (SM)
0x00A0 0x00EF	Contadores (C)
0x0110 0x016F	Temporizadores (T)
0x0190 0x01EF	Temporizadores (T)

Localidad	Tipo de Memoria Mapeada
0x0210 0x02FF	Entradas analógicas (AI)
0x0310 0x03FF	Salidas analógicas (AQ)
0x0400 0x7DD2	Memoria de usuario (V)
0x7DD3 0x7DDF	Registros del reloj de tiempo real (R)
0x7DE0 0x7DFF	Stack del program counter
0x7E00 0x7EFF	Registros de flanco positivo
0x7F00 0x7FFF	Registros de flanco negativo

	Área de memoria NO utilizada

Figura 4.3: "Memoria RAM CY7C199"



D. Coprocesador matemático

El PLC fue dotado con capacidades matemáticas bastante poderosas dentro del campo de los controladores programables, pues a la CPU central le fue instalado un coprocesador matemático de 32 bits y manipulación de datos en punto flotante. El coprocesador matemático utilizado es el PAK-IX (U4), seleccionado debido a su versatilidad y a su interfaz serial que se ajustaba adecuadamente al esquema diseñado para el sistema completo del iPLC. En el capítulo VI encontrará el listado de las funciones matemáticas incorporadas dentro del conjunto de instrucciones del iPLC.

1. El formato interno del PAK-IX. Todas las operaciones matemáticas del PAK-IX son realizadas utilizando un formato propio, desarrollado por el fabricante. Ese formato es una variación del formato IEEE754, que consiste en un formato para la representación de números de punto flotante con 32 bits y que es utilizado por diversos compiladores, como por ejemplo, algunas versiones del compilador C. Sin embargo, no es necesario conocer el formato interno del PAK-IX ya que este posee funciones para

realizar conversiones entre diversos formatos. Se pueden convertir números enteros de 24 bits, o en formato IEE754, al formato interno del PAK-IX y viceversa. De tal forma que todos los valores numéricos que se envíen al PAK-IX pueden estar en formato de enteros de 24 bits o en formato IEEE754, pero es indispensable que antes de realizar operaciones matemáticas con ellos, se les transforme al formato interno.

Figura 4.4: "Formato de los datos IEEE754"

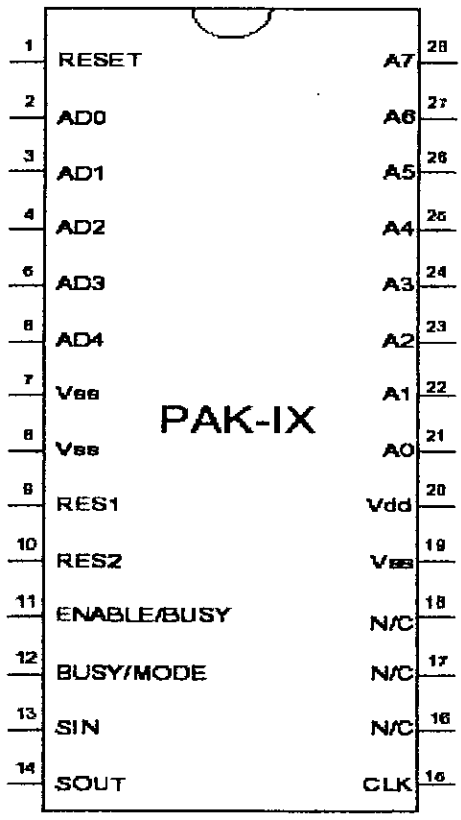
$$Valor = N = (-1)^S \times 2^{E-127} \times (1.M)$$

S (1 bit)	E (8 bits)	M (23 bits)

2. Señales del PAK-IX. En la figura 4.5 se muestran todos los pines del PAK-IX. A continuación se describe la función de cada uno de ellos.

a. **AD0-AD4.** Estos pines son entradas de canales analógicos de 10 bits.
b. **Reset.** Entrada para reinicializar el sistema interno del coprocesador matemático. Al aplicar una señal en bajo (0V), se inicializan los registros internos y el canal de comunicaciones.
c. **Vss.** Terminal de tierra para la alimentación.
d. **Res1/2.** Estas terminales se conectan a un resonador para producir la señal de reloj de temporización interna.
e. **Enable/Busy.** Si durante la inicialización esta entrada está conectada a un nivel alto (5V), este pin funciona como una entrada habilitadora para el coprocesador matemático. Un estado en alto habilita al coprocesador, mientras que un estado en bajo lo deshabilita.
f. **Sin y Sout.** Pines para la entrada / salida de datos respectivamente.
g. **Clk.** Entrada para la señal de reloj, proveniente del dispositivo maestro.
h. **N/C.** Estas terminales no deben conectarse. Deben quedar libres.
i. **Vdd.** Terminal de alimentación, debe conectarse a +5Vdc.
j. **A0-A7.** Puerto de entrada / salida de propósito general.

Figura 4.5: "Coprocesador matemático PAK-IX"



3. Comunicación. El PAK-IX utiliza un protocolo serial síncrono. La tasa de transferencia máxima es de 100 Kbps. Posee los pines Sin y Sout para la recepción y transmisión de datos respectivamente. Con esos pines puede comunicarse con un dispositivo maestro que no tenga la capacidad de utilizar un pin bidireccional. El pin Sout es de colector abierto, por lo que se puede conectar directamente con el pin Sin y comunicarse bidireccionalmente por medio de un sólo bit. En cualquiera de los dos casos, la comunicación siempre es half duplex, por lo que resulta más conveniente utilizar un solo bit bidireccional, si eso es posible para el dispositivo maestro que solicita operaciones matemáticas.

El primer bit que se recibe / transmite es el bit más significativo, y el muestreo de la línea de datos se realiza en el flanco positivo de cada pulso del reloj.

4. Instrucciones del PAK-IX. El coprocesador matemático utiliza dos registros principales denominados *X* y *Y* para la realización de sus funciones matemáticas. En estos registros se almacenan los argumentos de las funciones a calcular. Además, existen veinticuatro registros de propósito general, que pueden ser leídos / escritos por el dispositivo maestro. La tabla 4.2 muestra el conjunto de instrucciones del PAK-IX.

Tabla 4.2: "Conjunto de instrucciones del PAK-IX"

Código	Nombre	Descripción
0x01	LoadX	Almacena un dato en el registro X
0x02	LoadY	Almacena un dato en el registro Y
0x03	ReadX	Lee el valor del registro X
0x04	Swap	Intercambia el contenido de los registros X y Y
0x05	Digit	Determina el signo del valor almacenado en X
0x06	IEEEcvt	Convierte X desde el formato interno a IEEE754
0x07	Float	Convierte un entero de 24 bits al formato interno
0x08	Commck	Retorna un código de verificación para la comunicación
0x09	Fpcvt	Convierte X desde IEEE754 al formato interno
0x0A	Chs	X = -X
0x0B	Int	Convierte X en un entero de 24 bits
0x0C	Mul	X = X*Y
0x0D	Div	X = X / Y
0x0E	Sub	X = X – Y
0x0F	Add	X = X + Y
0x10	Opt	Configura las opciones de operación
0x11	Abs	X = |X|
0x12	Sto	Reg i = X
0x13	Rcl	X = Reg i.
0x14	DirA	Configura los bits del puerto A (entrada o salida)
0x15	RioA	Lee el puerto A
0x16	WioA	Escribe al puerto A
0x17	XtoY	Y = X
0x18	YtoX	X = Y
0x19	Sqrt	X = √X
0x1A	Log	X = Ln (X)
0x1B	Log10	X = Log (X)
0x1C	Exp	X = e ^ X

Continuación de la tabla 4.2.

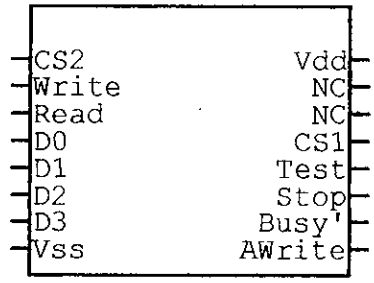
Código	Nombre	Descripción
0x1D	Exp10	X = 10 ^ X
0x1E	Pow	X = X ^ Y
0x1F	Root	$X = \sqrt[Y]{X}$
0x20	Recip	X = 1 / X
0x21/ 0x24	Sin/Asin	X = Sin (X) / X = Asin (X)
0x22/ 0x25	Cos/Acos	X = Cos (X) / X = Acos (X)
0x23/ 0x26	Tan/Atan	X = Tan (X) / X = Atan (X)
0x27	Poly	Calcula un polinomio utilizando los registros internos

E. Reloj de tiempo real

El PLC tiene instalado el reloj de tiempo real RTC58321B (U9), que posee las mismas capacidades que los RTCs incorporados en las computadoras personales. Ofrece la fecha y hora actuales, presentando el año, mes, semana del mes, día, horas, minutos y segundos, todo esto en un conjunto de trece registros de lectura / escritura accesibles desde el programa de usuario, por medio de las instrucciones correspondientes (ver el capítulo VII). Con este reloj de tiempo real, el PLC adquiere la capacidad de utilizar instrucciones que tomen en cuenta la hora y fecha actuales, y de esta forma, actuar según haya sido programado.

1. Características del circuito integrado. En la figura 4.6 se muestran todos los pines del RTC58321B, así como las señales correspondientes a cada uno de ellos. Los pines D0-D3 corresponden al bus de datos del RTC. La señal de control 'Write' se utiliza para escribir datos en los registros internos. La señal de control 'Read' se utiliza para leer datos de los registros internos. La señal de control 'AWrite' es utilizada para escribir direcciones, en otras palabras, sirve para seleccionar el registro al cual se desea leer o escribir. Las señales 'CS1' y 'CS2' son utilizadas para habilitar / deshabilitar al dispositivo.

Figura 4.6: "Reloj de tiempo real RTC58321B"



El RTC58321B posee un oscilador de cuarzo integrado, por lo que no requiere de componentes externos para su funcionamiento. La comunicación entre el RTC y el dispositivo maestro es llevada a cabo por medio del bus bidireccional de 4 bits, y las tres señales de control mencionadas anteriormente. Todas las operaciones de lectura y escritura son llevadas a cabo por medio de los trece registros internos asignados para el almacenamiento de la hora y fecha actual. La tabla 4.3 muestra el conjunto de registros internos del RTC58321B, utilizados en el sistema del iPLC. El RTC58321 posee algunos registros adicionales que no fueron utilizados (no se necesitaban).

Tabla 4.3: "Registros internos del RTC58321B"

Registro	D3	D2	D1	D0	Descripción
0x00	S3	S2	S1	S0	Unidades de segundos (0 a 9)
0x01	S7	S6	S5	S4	Decenas de segundos (0 a 5)
0x02	M3	M2	M1	M0	Unidades de minutos (0 a 9)
0x03	M7	M6	M5	M4	Decenas de minutos (0 a 5)
0x04	H3	H2	H1	H0	Unidades de horas (0 a 9)
0x05	24/12	PM/AM	H5	H4	Decenas de horas (0 a 2 ó 0 a 1) y formato del horario
0x06	--	SM2	SM1	SM0	Número de semana del mes actual (0 a 6)
0x07	D3	D2	D1	D0	Unidades del día
0x08	AB1	AB0	D5	D4	Decenas del día y configuración para los años bisiestos
0x09	M3	M2	M1	M0	Unidades de mes
0x0A	--	--	--	M4	Decena del mes
0x0B	A3	A2	A1	A0	Unidades del año (0 a 9)
0x0C	A7	A6	A5	A4	Decenas del año (0 a 9)

Los dos bits más significativos del registro 0x05 son utilizados para seleccionar el formato de horario en que debe operar el RTC. Para seleccionar un formato de 24 horas, el bit D3 debe hacerse igual a uno (1). Para seleccionar un horario vespertino, el bit d2 debe hacerse igual a uno (1). Los estados en bajo (cero lógico) seleccionan las opciones alternas presentadas en la tabla 4.3.

El reloj de tiempo real es alimentado por baterías recargables. Estas baterías se recargan siempre que el iPLC esté alimentado. Experimentalmente se determinó que con tres horas de operación del iPLC, la carga adquirida por las baterías es suficiente para mantener en operación al RTC58321B durante dieciséis días. Se considera que esto es suficiente si se toma en cuenta que bajo condiciones de operación normales, un PLC instalado en una planta trabaja muchas horas continuas, y no se detiene por mucho tiempo. Se considera que el tiempo de operación del RTC58321B aún puede incrementarse notablemente a medida que se incremente el tiempo de carga.

F. Coprocesador SitePlayer

Para la compatibilidad con la familia de protocolos TCP/IP y el servicio de páginas HTML, se desarrolló una interfaz entre el sistema iPLC y el módulo SitePlayer (U8), el cual posee un stack de TCP/IP y un servidor HTTP integrado. Este módulo posee un controlador de Ethernet y un microcontrolador 89C51RD2, producido por Philips, en el cual se almacenaron las páginas del entorno de programación / monitorización del sistema completo (en memoria flash interna).

1. Descripción del coprocesador SitePlayer. El módulo SitePlayer es un producto diseñado para la habilitación de la Internet en un sistema microcontrolado. En sus reducidas dimensiones (menos de 1 pulgada cuadrada) incluye un stack de TCP/IP, un servidor HTTP, un controlador de Ethernet 10baseT, memoria para almacenamiento de páginas HTML y un procesador de objetos. SitePlayer está diseñado para funcionar

como un módulo esclavo, incrustado en un sistema microcontrolado, donde el microcontrolador central funciona como el dispositivo maestro.

SitePlayer maneja los protocolos TCP/IP y los paquetes Ethernet de forma independiente, sin requerir interacción con el dispositivo maestro. La comunicación entre SitePlayer y el dispositivo maestro es realizada por medio de *objetos* enviados a través de un canal serial asíncrono. Los *objetos* son localidades de memoria en el microcontrolador del módulo SitePlayer, a las cuales se les asigna un nombre y un tipo. Para acceder a un *objeto*, simplemente se requiere transmitir el comando específico, para lectura o escritura, indicando la dirección del *objeto* correspondiente, luego se realiza la transferencia de datos.

Las páginas HTML del servidor son almacenadas en la memoria flash del módulo SitePlayer. En el código HTML de cada página, los *objetos* son accedidos por medio de la utilización del símbolo especial "^". Todos los elementos o texto en general, dentro del código HTML de la página, a cuyos nombres se antepone el símbolo "^", son transformados en *objetos*. Cuando el explorador de Internet carga una página HTML con *objetos*, ofrecida por el servidor HTTP del módulo SitePlayer, el servidor no envía el texto correspondiente con el nombre del *objeto*, sino que envía el valor del *objeto* con ese nombre. Es decir, que en el explorador de Internet no se despliega el texto encontrado como nombre del *objeto*, en cambio, se despliega el contenido de la localidad de memoria RAM en la cual está mapeado el *objeto* con ese nombre. Esa localidad, previamente definida como un *objeto*, es directamente accedida por el microcontrolador central, de tal forma que el valor del elemento *objeto* en la página HTTP es dinámico, y muestra los valores que le son escritos desde el sistema microcontrolado.

Este sistema de *objetos* permite que SitePlayer sea un servidor HTTP en el cual las páginas servidas por él presenten un contenido dinámico, que es modificado por medio de comandos enviados desde un microcontrolador. SitePlayer maneja los protocolos ARP, ICMP, IP, DHCP, TCP y UDP. Además ofrece 48K bytes para el almacenamiento de páginas HTML, y 768 bytes de memoria RAM para el mapeo de *objetos*. La tasa de

transferencia de su canal asíncrono puede ser configurado para operar desde 300 bps hasta 115.2 Kbps.

2. Los bloques funcionales. El hardware del módulo SitePlayer posee dos dispositivos: El circuito integrado controlador de Ethernet y el microcontrolador 89C51RD2 que posee todo el firmware del módulo. En ese firmware existen diversos bloques de programa, cada uno con una función específica. Tanto el hardware como el firmware pueden verse como un todo, con diversos bloques funcionales, que constituyen al módulo en sí. La figura 4.7 muestra la estructura funcional del módulo SitePlayer. Algunos de esos bloques son dispositivos de hardware, otros son componentes de software, y otros simplemente son regiones de memoria disponible para el almacenamiento de datos.

Figura 4.7: "Esquema de los bloques funcionales del módulo SitePlayer"

<table>
<tr><td rowspan="2">Interfaz serial asíncrona</td><td>Procesador de objetos

Interactúa con el puerto serial y provee interacción con los objetos</td><td colspan="2">Servidor HTTP

Sirve páginas HTML con características dinámicas. También procesa datos de entrada desde exploradores de internet</td><td>Procesador del protocolo Ethernet

Interactúa con el controlador Ethernet</td><td>Controlador Ethernet</td></tr>
<tr><td colspan="2">RAM para objetos

Ofrece 768 bytes para definición de objetos</td><td colspan="2">Memoria flash

Ofrece 48K bytes para almacenar páginas HTML e información de configuración</td><td>Puerto de entrada / salida

Ofrece un puerto de 8 bits para uso general</td></tr>
</table>

- La interfaz Ethernet consiste en el circuito integrado controlador de Ethernet.
- El bloque del procesador Ethernet y TCP/IP consiste en el stack de TCP/IP y el software que sirve de interfaz con el controlador de Ethernet.
- El puerto de entrada / salida consiste en 8 bits mapeados en un puerto del microcontrolador 89C51RD2, este puerto es de uso general, en aplicaciones sencillas puede evitar el uso de dispositivos de entrada / salida adicionales.

- El bloque de páginas HTML consiste en una región de memoria flash de 48 K bytes, reservada en la memoria interna del microcontrolador 89C51RD2 para el almacenamiento de las páginas que ofrece el servidor HTTP y los datos de configuración del módulo.
- El bloque del servidor HTTP consiste en un segmento de programa que implementa la funcionalidad de un servidor de páginas HTML.
- El procesador de objetos consiste en el software que hace posible el reconocimiento y procesamiento de los *objetos* situados en las páginas HTML del servidor.
- El bloque de *objetos* consiste en una región de memoria RAM de 768 bytes, reservada en la memoria interna del microcontrolador 89C51RD2 para el almacenamiento de los *objetos* que son utilizados en las páginas HTML.
- Finalmente se encuentra el bloque UART, que consiste en la interfaz entre el procesador de objetos y el periférico de transmisión serial asíncrono integrado en el microcontrolador 89C51RD2, que permite la interacción del módulo SitePlayer con el microcontrolador central (en el sistema microcontrolado específico).

3. El archivo de interfaz. Todos los elementos creados en las páginas HTML que se encuentran almacenadas en el servidor HTTP del módulo SitePlayer pueden ser utilizados como *objetos*, y de esa forma pueden servir para ofrecer interacción entre el sistema microcontrolado y el usuario remoto en un explorador de Internet. Sin embargo, detrás de esa interacción existe un sistema de transmisión y comunicación el cual hace posible que el usuario pueda enviar órdenes y datos hacia la memoria de *objetos* en el módulo SitePlayer. Esta comunicación se basa en la utilización del *archivo de interfaz*.

El *archivo de interfaz* consiste en una herramienta bastante poderosa, ofrecida como parte del software del módulo SitePlayer. Este archivo permite realizar transmisiones de datos desde elementos de páginas HTML, hacia *objetos* en la memoria RAM del microcontrolador 89C51RD2. Posteriormente esos *objetos* pueden ser transmitidos / consultados hacia / desde el sistema microcontrolado de la aplicación específica a través

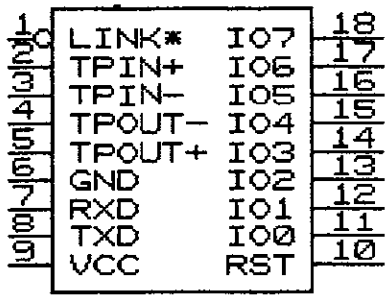
del canal asíncrono del microcontrolador central (u otro dispositivo que ofrezca un canal serial asíncrono).

El *archivo de interfaz* es utilizado para la interacción con el servidor HTTP, desde el explorador de Internet, como si se tratara de una aplicación CGI. En realidad funciona como una aplicación CGI que está desarrollada en el firmware del módulo SitePlayer.

El *archivo de interfaz* es creado como un archivo de texto, con extensión ".SPI", en el cual se indica la página HTML con la cual debe responder el servidor HTTP, luego de haber realizado la transmisión de los datos. Los datos que se transmiten figuran como si fueran los parámetros de la aplicación CGI.

4. Los pines del módulo SitePlayer. Tanto los dos dispositivos de hardware, como todos sus bloques funcionales, interactúan entre sí para ofrecer toda la funcionalidad del módulo SitePlayer. El sistema microcontrolado no interacciona directamente con los bloques funcionales, a excepción del bloque de *objetos*, en el cual lee y escribe todos los datos. Todos los demás bloques funcionales interactúan entre sí de forma independiente del sistema microcontrolado, para ofrecer el poderoso sistema de TCP/IP y servidor HTTP integrado. De esa forma, debido a tal independencia, el módulo SitePlayer únicamente ofrece unos cuantos pines para interactuar con un sistema microcontrolado y conectarse con un transformador / filtro que le permita acceder directamente a la capa física de una red Ethernet (típicamente por medio de un conector RJ-45).

Figura 4.8: "Módulo SitePlayer"



El pin *Link* se conecta a un LED que se enciende al detectar tráfico en la red a la cual se conecta el módulo. Los pines *TPx* se conectan a un transformador / filtro, que ofrece las salidas para conexión con un conector tipo RJ-45, listo para conectar el módulo SitePlayer con una red Ethernet. Los pines *RxD* y *TxD* pertenecen al puerto serial asíncrono del módulo SitePlayer, se conectan directamente con el puerto serial asíncrono del sistema microcontrolado. Este canal de comunicación es el único medio para la interacción entre el sistema microcontrolado y el módulo SitePlayer. El pin *Rst* es la entrada para la señal de reinicialización de todo el sistema del módulo SitePlayer, tanto del controlador Ethernet, como del microcontrolador integrado. Por último, los pines *IO0-IO7* corresponden con los 8 bits de datos bidireccionales para el puerto de uso general que ofrece el módulo SitePlayer. Las tareas de configuración, lectura y escritura de este puerto son realizadas por medio de los comandos correspondientes, enviados desde el sistema microcontrolado.

5. Proceso para la construcción del servidor HTTP. El servidor HTTP del módulo SitePlayer es una herramienta disponible para la aplicación específica del sistema microcontrolado. El módulo SitePlayer ofrece un servidor "vacío", es decir, es cuestión de la aplicación específica definir su configuración, estructura y contenido (páginas HTML). La construcción del servidor HTTP involucra un proceso de desarrollo en el cual se diseñan e implementan todas las características para el servicio de páginas HTML y para la interacción con el sistema microcontrolado.

El código que indica al módulo SitePlayer cómo y qué páginas HTML servir es denominado *archivo de definición.* En este archivo se define toda la estructura del servidor. La estructura consiste en todos los *objetos* y los parámetros del servidor, como por ejemplo, dirección IP, raíz del sitio de Internet en la estructura interna de archivos, entre otros. Posteriormente este archivo es compilado con la herramienta de software *SiteLinker*, generando así el *archivo binario de imagen*, el cual finalmente es transmitido a través de una conexión Ethernet, desde la estación de trabajo, hacia el módulo SitePlayer, para que finalmente sea almacenado en la memoria flash del microcontrolador 89C51RD2.

El proceso completo para la construcción del Servidor HTTP envuelve cuatro fases principales:

- Crear el *archivo de definición*, utilizando un editor de texto, en el cual se definen la configuración y todos los *objetos* del servidor HTTP.
- Crear las páginas HTML.
- Compilar el *archivo de definición* y cargar el *archivo binario de imagen* en la memoria flash del módulo SitePlayer.
- "Navegar" a través del Servidor HTTP y verificar su funcionamiento.

Los detalles acerca del proceso completo es un tema extenso, que no se expondrá debido a que en este proyecto no se desarrolló el módulo SitePlayer, simplemente se utilizó para la realización del proyecto del iPLC, y su inclusión redundaría en la información que ofrece el fabricante y extendería exageradamente este documento. El autor requirió aprender todo el proceso y sus detalles, sin embargo, eso no es indispensable para la exposición del proyecto hacia el lector. Si se desea profundizar en los detalles del proceso completo, se sugiere visitar el sitio de Internet del fabricante del módulo SitePlayer, del cual se puede descargar la documentación completa. La dirección de la página de inicio es HTTP://www.siteplayer.com.

En el capítulo VI se presenta la construcción del servidor HTTP implementada por el autor para el desarrollo del sistema completo del iPLC.

6. Comandos y mapa de memoria en el módulo SitePlayer. Como se ha mencionado con anterioridad, toda la interacción entre el sistema microcontrolado y el módulo SitePlayer es llevada a cabo por medio de comandos seriales. Los comandos más útiles son los de escritura / lectura, con los cuales se puede modificar el valor de los *objetos*, o leer el contenido de los mismos, así como también resulta posible escribir a localidades de memoria donde se encuentran registros especiales, para configurar el funcionamiento y parámetros del módulo SitePlayer.

Tabla 4.4: "Comandos del módulo SitePlayer"

Nombre	Código	Descripción
NOP	0x00	No tiene efecto. Recomendado como comando de inicialización.
Status	0x10	Retorna el byte de estado del módulo SitePlayer.
Reset	0x20	Reinicializa el módulo SitePlayer.
ComParams	0x33	Configura la tasa de transferencia serial y el retardo entre bytes.
UDPsend	0x50	Envía un paquete UDP (User Datagram Protocol).
Read	0xC0	Lee el contenido de un *objeto*. Utiliza un byte de dirección.
Write	0x80	Escribe en un *objeto*. Utiliza un byte de dirección.
ReadX	0xD0	Lee el contenido de un *objeto* con dirección de dos bytes.
WriteX	0x90	Escribe en un *objeto* con dirección de dos bytes.
ReadBit	0xE0	Lee una variable tipo bit, con un byte de direccionamiento.
WriteBit	0xA0	Escribe en una variable de bit, con un byte de direccionamiento.
ToggleBit	0xB0	Complementa una variable de bit. Utiliza un byte de dirección.

Tabla 4.5: "Mapa de memoria RAM en el módulo SitePlayer"

Dirección	Descripción
0x0000 a 0x02DF	Memoria para *objetos* regulares. Direccionable con uno o dos bytes.
0x2E0 a 0x2FF	Memoria para *objetos* tipo bit, direccionados con un byte, u *objetos* regulares direccionados con dos bytes.
0xFF00 a 0xFF1F	Registros de función especial.
0xFFE0 a 0xFFE5	Dirección MAC (6 bytes)
0xFFE6 a 0xFFE9	Dirección IP (4 bytes)

G. Interfaz para el conector RJ-45

Para ofrecer la capa física de la habilitación de la Internet en el sistema del iPLC, se agregó un transformador de Ethernet conectado al circuito integrado controlador de Ethernet del módulo SitePlayer. El transformador utilizado es el 20F001N (T1 en la figura A-1 del apéndice A).

H. Módulos de expansión

El iPLC está listo para la interconexión inmediata con módulos de expansión compatibles con el protocolo desarrollado (también como parte de este proyecto). Se diseñó el *módulo de expansión modelo* para los módulos de expansión, tanto en hardware como en software. Este módulo utiliza un microcontrolador PIC16F877 como controlador central.

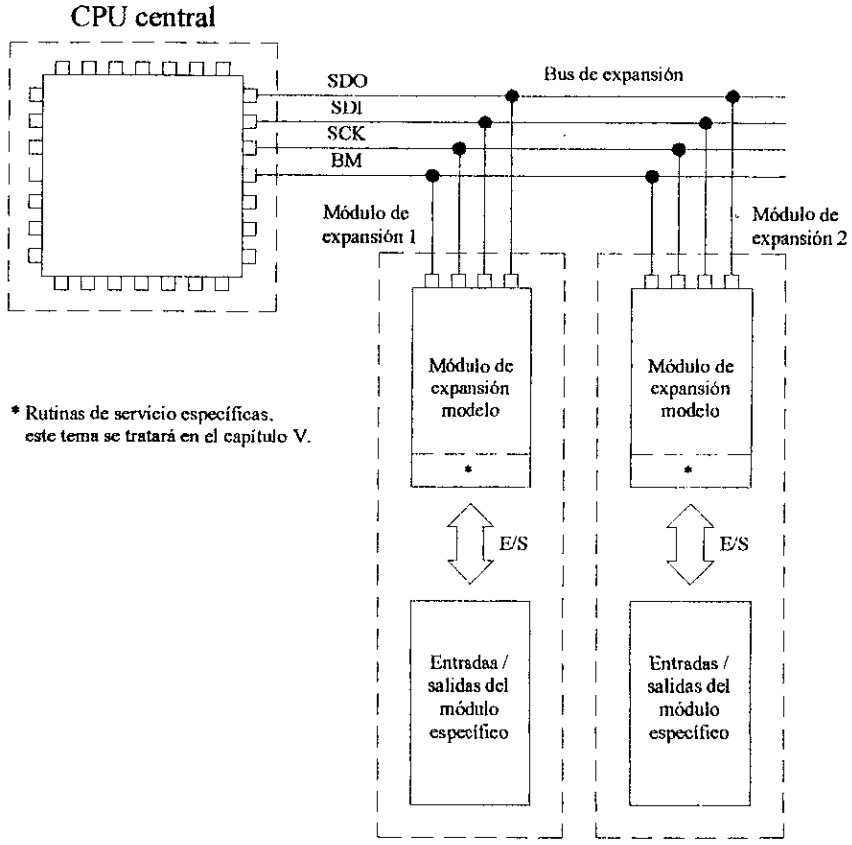
1. Módulo de expansión modelo.

a. El bus de expansión. Todas las transacciones entre la CPU central y los módulos de expansión son llevadas a cabo por medio de este bus. Consiste en cuatro bits, uno para transmisión de datos desde el microcontrolador central (bit SDO), otro para recepción de datos en el microcontrolador central (bit SDI), un bit para la señal de reloj (bit SCK) a 5 MHz, y un último bit para la señalización de respuesta / ocupado (bit BM). El bit BM (Busy Module) es exclusivamente controlado por los módulos de expansión. Debido a su categoría de bus, las señales de éste son compartidas por todos los módulos de expansión que se encuentren instalados. En el bus nunca se presentan colisiones debido a la implementación de una estrategia en el software del módulo de expansión modelo.

b. El controlador del módulo de expansión modelo. El microcontrolador PIC16F877 es el encargado de comunicarse con la CPU central, procesar todas las solicitudes y enviar los resultados correspondientes. Este controlador es el "cerebro" del módulo de expansión, al funcionar como interfaz entre la CPU central del sistema y la electrónica específica dentro del módulo de expansión mismo, electrónica que ofrece más capacidad al sistema completo (entradas / salidas digitales / analógicas, salidas PWM, salidas de impulsos, etc.).

En este módulo de expansión modelo se definen todas las señales de control necesarias para conectar cualquier tipo de módulo específico a la CPU central.

Figura 4.9: "El módulo de expansión modelo"

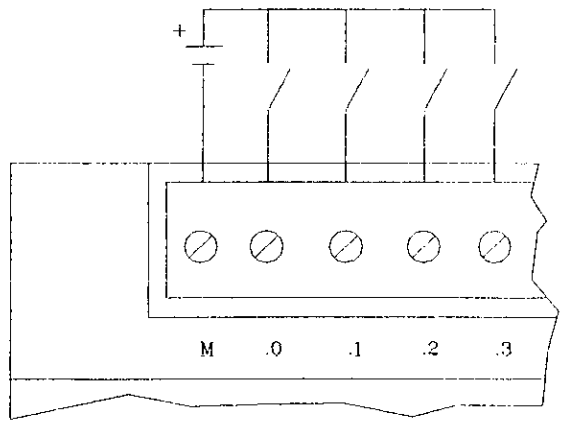


2. Electrónica del módulo de expansión digital (iMxD1). Para este módulo de ocho entradas y ocho salidas digitales, se utilizaron dieciséis bits, ocupando dos puertos del PIC16F877, conectados directamente a cuatro opto acopladores ECG3221, de esta forma se construyó un módulo de expansión con entradas y salidas aisladas eléctricamente de la lógica digital interna por hasta un pico de 5 KV, por lo que resulta un módulo seguro en cuanto a la protección tanto del módulo de expansión, como de la CPU central a la que se encuentra conectado. El diagrama esquemático del módulo de expansión digital puede consultarse en el apéndice A.

a. Conexión de las entradas en el módulo de expansión digital. Todos los módulos de expansión se alimentan con +24Vdc, suministrados desde la CPU central. Debido al acoplamiento óptico, para las entradas se posee una terminal de masa común, que puede corresponder con la tierra del circuito externo acoplado ópticamente. Todas

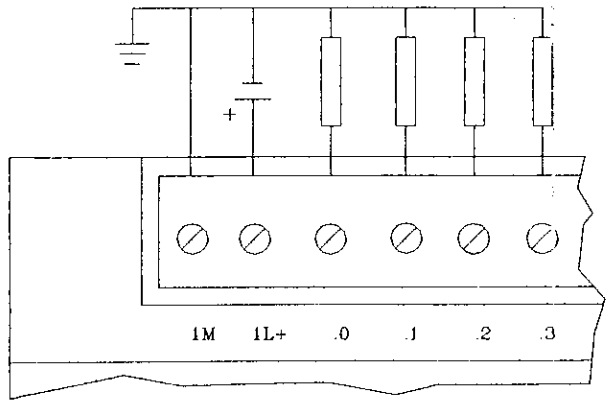
las terminales de entrada del módulo de expansión deben ser conectadas al dispositivo que ofrezca +24Vdc nominales para el nivel alto (1 lógico).

Figura 4.10: "Conexión de las entradas en el módulo de expansión digital iMxD1"



b. Conexión de las salidas del módulo de expansión digital. Puesto que las salidas son tipo fuente, una terminal de la carga debe ir conectada a la terminal de salida del módulo de expansión, mientras que la segunda terminal de la carga debe conectarse a la terminal común de masa instalada en el módulo de expansión, que se encuentra acoplada ópticamente y que puede coincidir con la tierra del circuito externo. La terminal "M" del módulo de expansión corresponde con la masa del circuito externo, mientras que la terminal "L+" corresponde con la alimentación de +24 Vdc nominales, ofrecidos por una fuente externa.

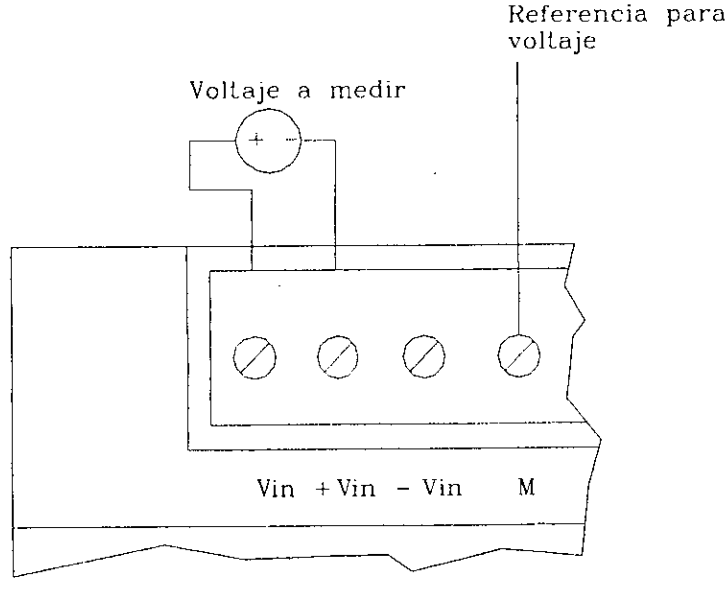
Figura 4.11: "Conexión de las salidas en el módulo de expansión digital iMxD1"



3. Electrónica del módulo de expansión analógico (iMxA1). Este módulo posee una entrada analógica simple (voltaje), que es precisamente una entrada analógica integrada en el controlador del módulo de expansión. Además posee una entrada analógica diferencial (voltaje), que fue construida a partir de otra entrada analógica integrada en el microcontrolador del módulo, y con la adición de una etapa diferencial que consta de un amplificador de instrumentación, y un filtro pasabajas con frecuencia de corte f_c = 3.1 KHz. Para la salida de voltaje se utilizó el DAC0808. Para la salida de corriente se utilizó la salida del DAC0808 como entrada a un circuito convertidor de voltaje a corriente. El diagrama esquemático del módulo de expansión analógico puede consultarse en el apéndice A.

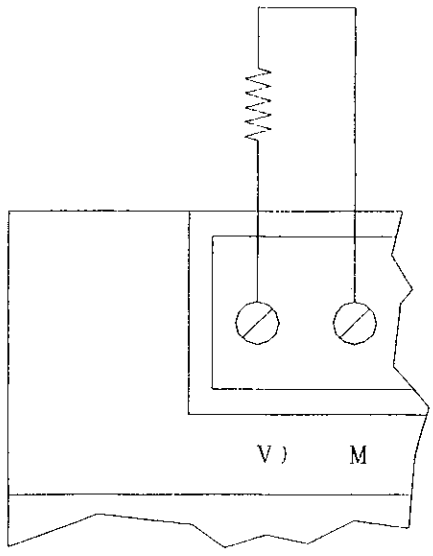
a. Conexión de las entradas en el módulo de expansión analógico. Si se trata de una entrada diferencial de voltaje, las dos terminales del dispositivo a medir deben ir conectadas a las terminales Vin+ y Vin- instaladas en el módulo de expansión. Si se trata de una entrada simple, entonces la terminal del dispositivo debe conectarse con la terminal Vin instalada en el módulo de expansión. Además, la entrada simple de voltaje debe ser conectada a la masa del módulo de expansión.

Figura 4.12: "Conexión de las entradas en el módulo de expansión analógico iMxA1"



b. Conexión de las salidas del módulo de expansión analógico. Se trata de dos terminales de salida simples, una de voltaje y otra de corriente, por lo que la carga debe ir conectada entre la terminal de salida deseada y la terminal de masa instalada en el módulo de expansión.

Figura 4.13: "Conexión de las salidas en el módulo de expansión analógico iMxAI"



I. Los LED's de estado del sistema

La tarjeta madre posee seis LED's que indican el modo de operación actual del iPLC y el último error ocurrido. Un LED bicolor indica el modo de operación del iPLC. Para el modo de ejecución, se enciende de color verde, mientras que para el modo detenido, el color es rojo. Otro LED, de color verde, se enciende cuando se detecta que el iPLC ha sido conectado a una red Ethernet, es el típico indicador "Link" de las tarjetas de red.

Los otros cuatro LED's notifican el código del último error ocurrido (ver el capítulo V). Todos estos LED's fueron mapeados utilizando 6 bits del puerto de entrada / salida que ofrece el módulo del stack de TCP/IP.

J. Mapeo físico de los periféricos en el microcontrolador de la CPU central

Todos los dispositivos y periféricos del sistema del PLC están mapeados físicamente en los puertos del microcontrolador central PIC16F877. La distribución de los puertos y los bits para el mapeo es la siguiente:

1. **Memoria flash y memoria RAM.** Se utilizó el puerto D, de ocho bits, en el microcontrolador central, para el bus de datos y direcciones del sistema. El bus de datos posee ocho bits, conectados directamente al puerto D del microcontrolador. El bus de direcciones está multiplexado con el bus de datos, utilizando dos Flip-Flops ECG74LS374, para conseguir el bus de direcciones de dieciséis bits necesario para el mapeo de toda la memoria RAM y flash del sistema. La velocidad de lectura flash / RAM, y de escritura RAM entre el microcontrolador central y estos dos periféricos es la máxima posible, pues la memoria flash soporta ciclos de lectura de 200 nS, la RAM de 15 nS, y el tiempo de ejecución de una instrucción dentro del microcontrolador central toma 200 nS (con el reloj a 20 MHz). Nótese que el tiempo de instrucción al que se está haciendo referencia es al del código de programa cargado al microcontrolador central (firmware del sistema), no al tiempo de ejecución de las instrucciones de programa del PLC (software de usuario cargado al sistema).

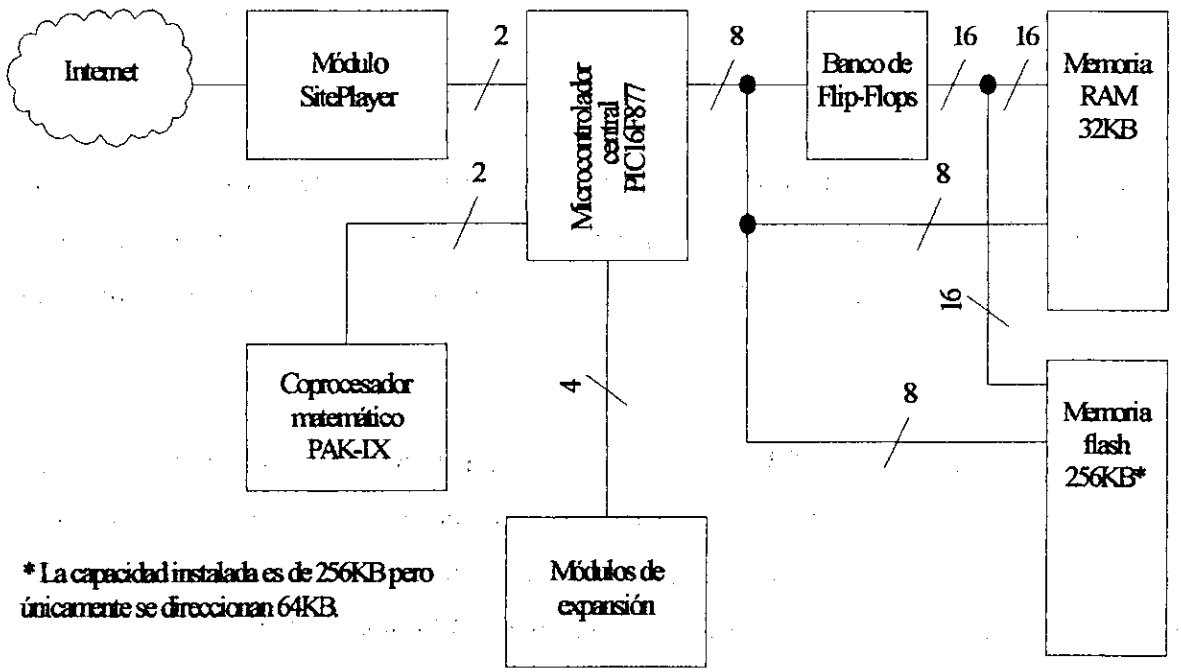
2. **Coprocesador matemático.** Este periférico fue mapeado utilizando tres bits del puerto A. Un bit bidireccional para datos, un bit para reloj, y un último bit para señalización de periférico ocupado. La transferencia de datos fue realizada utilizando protocolo serial síncrono a 100 K bits/s.

3. **Módulo de stack TCP/IP y servidor HTTP (SitePlayer).** Este periférico fue mapeado utilizando dos bits del microcontrolador central. Se utilizó protocolo serial asíncrono a 115.2 K bits/s. La comunicación entre el módulo TCP/IP y la red Ethernet a la que se encuentre conectado se realiza a una tasa de 10 M bits/s (10 Base T).

4. Reloj de tiempo real. Para este periférico se utilizó un segundo bus de datos, de cuatro bits, y tres bits adicionales para señales de control. El bus de datos y dos bits de control fueron mapeados en el puerto B, el tercer bit de control se mapeó en el puerto A, ambos puertos pertenecen al microcontrolador central.

5. Módulos de expansión. Para la comunicación con los módulos de expansión, se utilizó un bus de cuatro bits. Dos bits para transmisión y recepción de datos, un bit de reloj y un bit para señalización de respuesta (Acknowledge). Se utilizó protocolo SPI para la transmisión de datos, y un protocolo propio, "montado" sobre el SPI, diseñado para este proyecto, para la comunicación entre la CPU central y los todos los módulos de expansión. Este protocolo está basado en el envío de comandos y direcciones, luego esperar la respuesta del módulo, si está presente, y finalmente realizar la transacción entre éste y la CPU central. La transmisión de datos entre la CPU central y los módulos de expansión, a través del bus de expansión del sistema, se realiza a una tasa de 5 M bits/s.

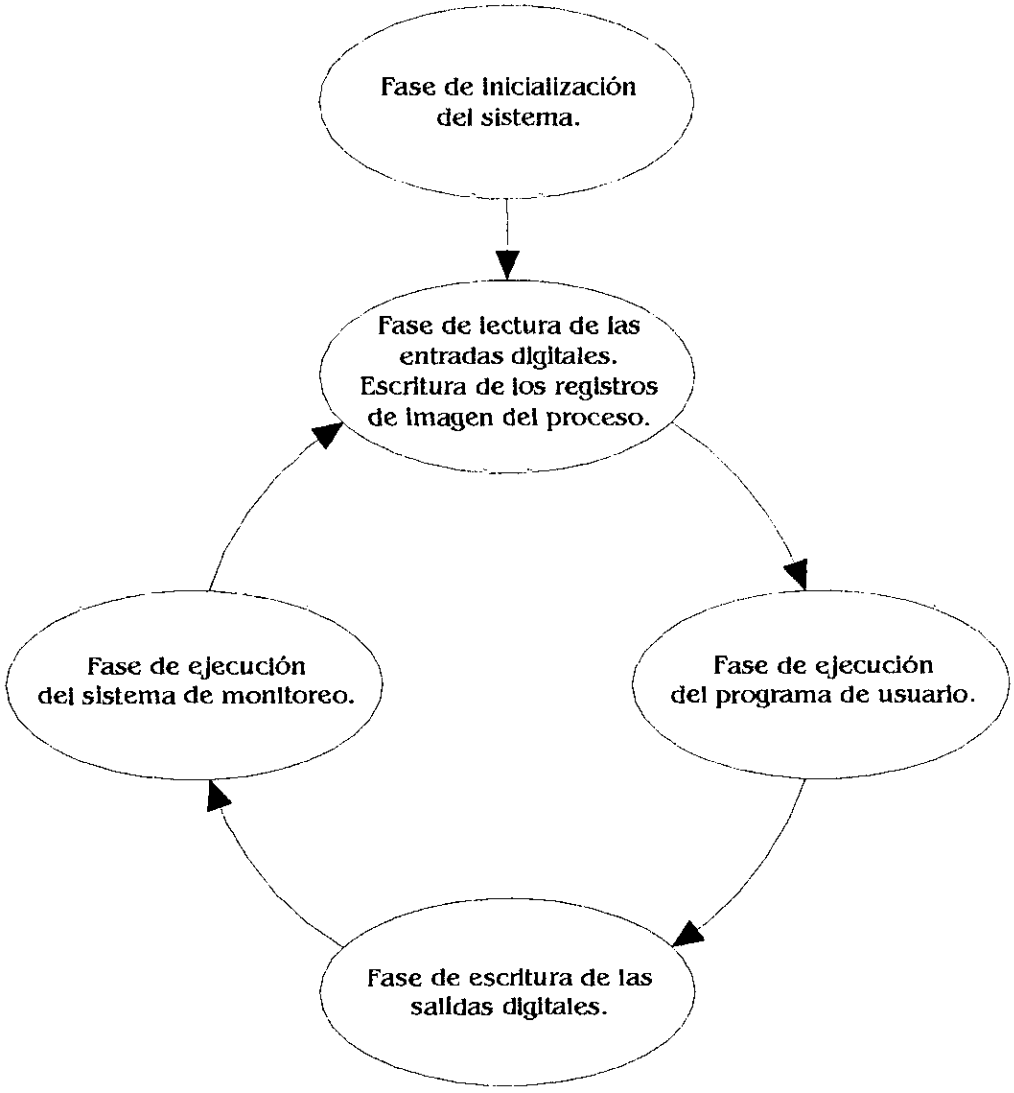
Figura 4.14: "Esquema de la arquitectura de la tarjeta madre del iPLC"



V. SISTEMA DE SOFTWARE DEL iPLC

Toda la funcionalidad del iPLC está abstraída en los modos de operación del sistema. El iPLC ofrece los modos de operación típicos: *Detenido* y *de ejecución.* Además se incluyó el submodo de *programación*, necesario para la reprogramación remota del sistema. El ciclo más complejo es el del modo de ejecución, pues en este modo, el iPLC realiza la máxima cantidad de tareas, completándose todas las fases que se muestran en la figura 5.1, este ciclo es modificado, eliminando algunas fases, cuando se activa el modo de operación detenido o el submodo de programación.

Figura 5.1: "El ciclo del iPLC"



A. La dualidad en la estructura del firmware del iPLC

El firmware de la CPU central presenta dos esquemas funcionales: El software de sistema y el software de microprocesador. El primero se refiere al software que define el funcionamiento del sistema completo, el iPLC como un todo. Este software proporciona la estructura lógica para la ejecución y el control de todas las fases del iPLC, tomando en cuenta a todas aquellas fases que constituyan el ciclo activo del PLC, y a todos los procesos asíncronos que pueden suceder dentro o fuera del mismo. Los eventos asíncronos pueden ser provocados por procesos de temporización, o por procedimientos de usuario, desde el entorno de programación / monitorización del sistema. El segundo esquema corresponde con la implementación de un *microprocesador virtual*, el cual se encarga de la ejecución y procesamiento de las instrucciones del programa de usuario, de una forma autónoma con respecto a todas las demás fases y variables del sistema.

1. Software de sistema. Este es una especie de "sistema operativo", ya que sobre este software se ejecutan todos los procesos del iPLC, desde la inicialización de periféricos, hasta la reprogramación y monitorización del iPLC. De hecho, este software es quien lleva a cabo la fase de ejecución del programa de usuario, que resulta ser un microprocesador desarrollado en software: el microprocesador virtual.

El software de sistema se encarga de ejecutar todas las fases que definen la operación del PLC. Estas fases son: Fase de inicialización, fase de lectura de la entradas digitales, fase de ejecución del programa de usuario, fase de actualización de las salidas digitales y fase de monitorización. Además, también se encarga de procesar eventos originados interna y externamente, estos son: Actualización de temporizadores e interacción con el servidor HTTP.

a. Ejecución de las fases. Se refiere a la ejecución de todos los eventos síncronos del sistema, que abarca precisamente a todas las fases del iPLC. Se dice que las fases son eventos síncronos ya que ocupan un lugar definido dentro del

funcionamiento cíclico del PLC, ocurriendo de forma periódica y en "sincronía" con todo el proceso interno del controlador central.

1) Interacción con los módulos de expansión. Esta funcionalidad se incluye aquí puesto que es realizada como parte de las fases del sistema. Todas las fases del sistema pueden eventualmente realizar algún tipo de interacción con los módulos de expansión.

2) Interacción con los periféricos. A excepción de algunas funcionalidades del servidor HTTP, todas las demás funciones implementadas en los dispositivos periféricos de la tarjeta madre interactúan de forma síncrona con el controlador central. Todas estas funciones son ejecutadas dentro de la fase de ejecución del programa de usuario.

3) Escritura de las variables del sistema. Cada una de las variables del sistema es actualizada en un momento específico durante alguna de las fases. El acceso a estas variables es síncrono, determinado por el software de sistema, quien se encarga de escribir en ellas justo en los momentos predefinidos. El acceso desde el programa de usuario está restringido a operaciones de lectura.

b. Eventos asíncronos. Aquí se encuentran las tareas realizadas de forma asíncrona con relación al ciclo de operación del sistema. El software de sistema posee rutinas específicas para la actualización de todos los temporizadores. Esta actualización es realizada en cualquier parte del ciclo, cuando corresponda según el tiempo transcurrido dentro de cada ciclo. Además, el software de sistema proporciona las funciones necesarias para el servicio de los eventos generados por el usuario, desde el entorno de programación / monitorización del sistema, en el servidor HTTP. La interacción con el servidor HTTP se realiza por medio de un canal de datos asíncrono. El software de sistema siempre está atento a los datos que reciba en ese canal, cuando el usuario desea interactuar con el iPLC, el software de sistema procesa los datos y actúa según se

requiera. Básicamente. la interacción con el iPLC puede referirse a la reprogramación del sistema, cambios del modo de operación, o procedimientos de monitorización.

2. El software de microprocesador. Este es el que implementa al microprocesador virtual. El microcontrolador PIC16F877 posee la arquitectura de un microprocesador y diversos periféricos conectados a éste. Todos esos dispositivos existen en el hardware interno del microcontrolador, y fueron utilizados para el desarrollo del sistema completo de la CPU central, es decir, todo el firmware del controlador central es ejecutado por este sistema de microprocesador y periféricos, internos en el controlador central mismo. Dentro del programa cargado al controlador central se encuentra un segmento que emula el funcionamiento de un microprocesador, este es el denominado *microprocesador virtual*, que se refiere a la implementación de un microprocesador en software.

La función del microprocesador virtual es ejecutar el programa de usuario, previamente cargado al iPLC, almacenado en la memoria flash de la tarjeta madre. El microprocesador virtual funciona únicamente durante la fase de ejecución dentro del ciclo del iPLC, al momento de ejecutar el segmento de programa que lo implementa.

a. Estructura del microprocesador virtual. El microprocesador virtual posee sus propios registros internos, banderas, pila de programa, y contador de programa.

Los buses externos fueron mapeados en un puerto de ocho bits del microcontrolador central, específicamente en el puerto D. El bus de datos posee ocho bits, y se encuentra multiplexado con el bus de direcciones, de dieciséis bits. Los ocho bits del puerto D en el microcontrolador central funcionan directamente como el bus de datos. Estos ocho bits fueron conectados a un banco de flip-flops para obtener el bus de direcciones multiplexado de dieciséis bits.

Las señales de control para lectura, escritura y selección de dispositivo, fueron mapeadas en los tres bits del puerto E del microcontrolador central. Únicamente se

requirió un bit para la señal de selección del dispositivo ya que el microprocesador virtual interactúa directamente solo con dos de ellos: La memoria flash y la memoria RAM.

b. Funcionamiento del microprocesador virtual. Básicamente, la fase de ejecución del programa de usuario consiste en la ejecución del microprocesador virtual. Cuando el ciclo del PLC llega a la fase de ejecución, el control del programa se traslada hacia el segmento de software que implementa al microprocesador virtual. Estando en este segmento, el funcionamiento del microprocesador en software (virtual) da inicio, quedando como fondo la ejecución del software de sistema.

La primera tarea del microprocesador virtual consiste en la inicialización de las banderas, el contador de programa y la pila utilizada para la lógica del programa de usuario. Posteriormente se ingresa en un ciclo de dos niveles.

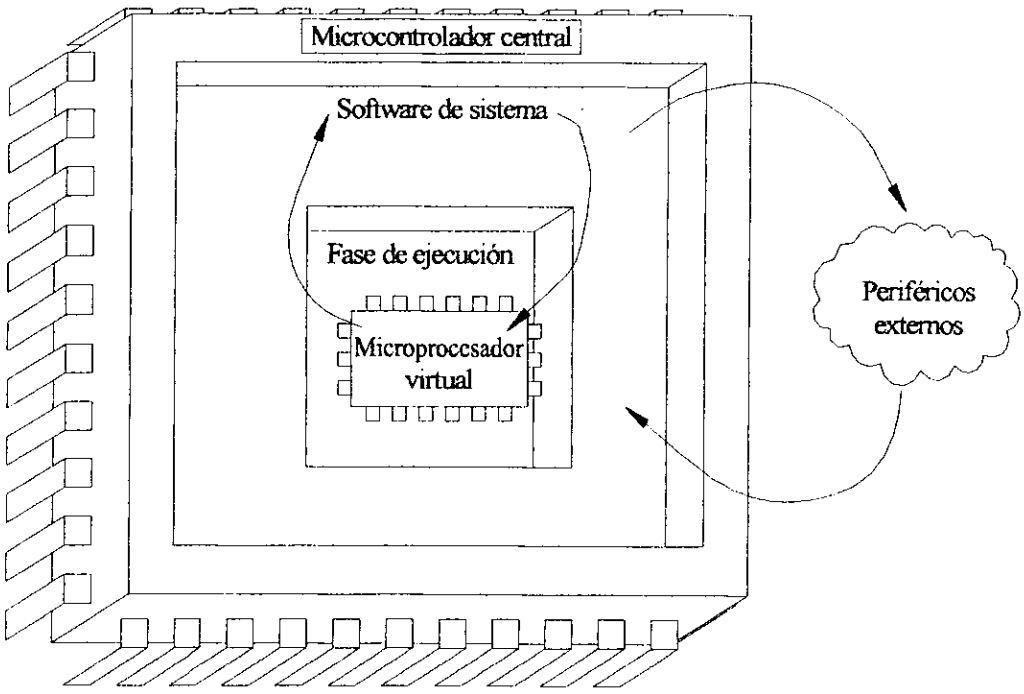
En el nivel más interno se realiza la lectura de las instrucciones de programa, hasta encontrar el final de un circuito (ramal del programa de usuario cargado al PLC), en ese punto se borra el contenido de la pila lógica y se inicia la lectura de las instrucciones del siguiente circuito.

El nivel más externo del ciclo finaliza cuando ya se procesaron todos los circuitos del programa de usuario. En ese momento, el programa de usuario concluye, al igual que la fase de ejecución.

Los detalles acerca de todas las fases, eventos y rutinas serán presentados en secciones posteriores, con el propósito de evitar que el lector pierda la visión general del sistema.

La figura 5.2 muestra que el software de sistema "contiene" al microprocesador virtual cuando se ejecuta la fase de ejecución. El microprocesador virtual interactúa directamente con el software de sistema, mientras que este último lo hace con todos los periféricos.

Figura 5.2: "La dualidad en la estructura del firmware del iPLC"



B. Modo de ejecución

En este modo de operación el ciclo del iPLC es el más largo posible, pues realiza todas las tareas del sistema. Estas tareas comprenden la lectura de las entradas digitales, la ejecución del programa de usuario, la actualización de las salidas digitales, y la ejecución del sistema de monitorización, con el cual se procesan todas las solicitudes de variables a monitorizar desde el entorno correspondiente en el servidor HTTP.

C. Modo detenido

En este modo, el iPLC únicamente ejecuta la fase de monitorización, pues al no ejecutar el programa de usuario (característica esencial del modo detenido), se considera prescindible la inclusión de las fases de lectura y escritura de las salidas digitales

instaladas en el sistema. Cuando el sistema lleva a cabo la transición del modo de ejecución al modo detenido, se realiza la única escritura a todas las salidas, para colocarlas en cero. Con esto se evita que algún dispositivo o máquina permanezca activo cuando el iPLC esté en el modo detenido.

D. Submodo de programación

En este modo el sistema se encuentra preparado para la recepción del código de programa ingresado por el usuario, para que éste sea almacenado en la memoria flash de la tarjeta madre. Este modo es una variación del modo detenido, pues la diferencia con respecto a este último, es que en el modo de programación, el sistema deshabilita la fase de monitorización y habilita la función de grabación en memoria flash para los datos que reciba en su puerto serial asíncrono, medio a través del cual recibirá todos los códigos del programa de usuario.

1. Estado del sistema. Cuando el sistema se coloca en el submodo de programación, primero se habilita la recepción de datos en el canal asíncrono del controlador central, luego se entra en el ciclo del modo detenido. En la fase de monitorización se evalúa el modo activo, si el sistema se encuentra en el submodo de programación, la función de monitorización queda anulada. La anulación de las solicitudes de monitorización es necesaria para evitar colisiones en el canal de datos, puesto que el servicio de estas solicitudes, y la transmisión de datos de reprogramación, comparten el único canal asíncrono del sistema.

Cuando el sistema ya está esperando datos en el canal asíncrono, recibe cada byte y ejecuta la función de grabación en memoria flash, una vez por cada byte recibido. Luego de grabar cada byte, se retorna al ciclo del modo detenido, a esperar que llegue el próximo byte. El sistema continúa en ese ciclo de grabación hasta el momento en que se active el modo detenido o el modo de ejecución, momento en el cual deshabilita la recepción de datos en el canal asíncrono, y reactiva la función de monitorización.

2. Grabación en memoria flash. Cuando el sistema se encuentra en el submodo de programación, recibe cada byte de código de programa y lo almacena en memoria flash. La grabación es efectuada por medio de la función de escritura flash incorporada en el software de sistema (rutina "flash write"). En esta función se encuentra desarrollado el algoritmo completo para la grabación de datos en la memoria flash de circuito integrado CAT28F020. Esta memoria requiere de un procedimiento especial para realizar grabaciones. Los detalles acerca de los requerimientos de grabación pueden consultarse en las hojas de datos proporcionadas por el fabricante, o puede consultarse la implementación efectuada en este proyecto, en el código fuente del software de sistema (ver el apéndice E).

La grabación de datos en la memoria flash es completamente segura, pues la función incorpora un método para la verificación de escritura, comparando los datos escritos con los datos que se desean escribir. La función toma en cuenta el dato proporcionado por el fabricante para el máximo número N_{Wmax} de intentos de escritura. Si durante el intento de grabación de cualquier byte, se supera el valor N_{Wmax}, entonces el circuito integrado de memoria flash debe estar dañado, por lo que el sistema realiza la notificación de error correspondiente (código de error # 2), tanto en los LED's de la tarjeta madre, como en el entorno de monitorización.

Todos los datos que se graban en la memoria flash son bytes previamente codificados a partir del código fuente del programa de usuario. El código fuente es ingresado desde el entorno de programación del sistema.

3. Borrar el programa de usuario. Previo a la grabación de datos en memoria flash, cuando se desea reprogramar al iPLC, es necesario borrar el programa de usuario grabado en flash durante la última reprogramación del sistema. Para esto se dispone de un acceso en el entorno de programación. Siempre que se desee reprogramar al iPLC, lo primero que debe hacerse es borrar el contenido de la memoria flash, luego se puede continuar con el procedimiento, cargando el nuevo programa en la memoria con el acceso correspondiente, ofrecido en el entorno de programación.

Cuando se selecciona la opción de borrar el programa de usuario, el sistema ejecuta la función de borrado para la memoria flash (rutina "flash erase"). En esta función se implementa un procedimiento de verificación automático. Con la verificación se determina si el contenido de toda la memoria flash a sido efectivamente borrado. Si la memoria no ha sido borrada correctamente, se realiza un nuevo intento, tomando en cuenta el número máximo N_{Emax} de intentos de borrado, ofrecido por el fabricante. Si se supera el valor N_{Emax}, se concluye que el circuito integrado de la memoria flash está dañado, por lo que se realiza la notificación correspondiente (Código de error # 1).

E. Determinación del modo de operación

El sistema necesita determinar cuál es el modo de operación activo, pues dependiendo de ello debe habilitar / deshabilitar ciertas funciones y ejecutar o cancelar ciertas fases dentro del ciclo. El modo de operación del iPLC es exclusivamente seleccionado por el usuario, quien realiza la elección utilizando el entorno de programación en el servidor HTTP. Cada modo de operación posee un código de dos bits. Cuando el usuario selecciona un modo de operación, el código correspondiente aparece en dos bits del puerto de E / S del módulo SitePlayer (servidor HTTP). Esos dos bits se encuentran conectados a un puerto del microcontrolador central. El software de sistema posee un registro mapeado en ese puerto, de tal forma que cuando el usuario selecciona un modo de operación, el software de sistema puede determinarlo por medio de la lectura de dicho registro. Este registro está implementado como una marca especial (SM5), la cual es leída de forma continua en diversas fases del ciclo.

Cuando el sistema determina que se ha cambiado de modo de operación, ejecuta todas las acciones necesarias para la activación del mismo. El sistema posee dos ciclos principales de operación, uno en el cual ejecuta todas las fases, que corresponde al modo de ejecución, y otro que corresponde al modo detenido, compartido con el submodo de programación. Básicamente, cuando se detecta un cambio de modo de operación, el sistema cambia el ciclo activo.

F. Los contadores del sistema

El iPLC posee cuarenta contadores de dieciséis bits cada uno, quince de esos bits almacenan la cuenta actual, mientras que el bit más significativo corresponde al bit C, que se hace igual a cero (0) cuando la cuenta actual es menor que el valor predeterminado, y se hace igual a uno (1) cuando la cuenta actual es mayor o igual que el valor predeterminado. El valor predeterminado (VP) se indica como parámetro en la instrucción del programa de usuario.

Todos los contadores están implementados en software, en el área de memoria C (ver tabla 4.1). Todos los contadores implementados en el iPLC son del tipo ascendente, por lo que su comportamiento se limita a incrementar una unidad en su valor de cuenta actual por cada flanco ascendente que sea detectado en su entrada.

Los contadores poseen un tope unipolar de +32767 para el valor de la cuenta actual. Si se presentan flancos ascendentes luego de haber alcanzado el valor tope, la cuenta actual simplemente se mantiene en dicho tope.

Figura 5.3: "Formato de la palabra para los contadores del iPLC"

D15	D14	D13	D1	D0
bit C	Msb de la cuenta actual			Lsb de la cuenta actual

La única forma de acceder a los contadores es a través del programa de usuario. Por medio de la instrucción de acceso para los contadores, ofrecida por el lenguaje de programación, es posible incrementar el valor de la cuenta actual, o inicializarla con cero.

G. Los temporizadores del sistema

El iPLC posee en total noventa y seis temporizadores, todos implementados en software. Se implementaron cuatro temporizadores de 1 mS, mapeados en el rango 0x0110 a 0x0117, 16 temporizadores de 10 mS, mapeados en el rango 0x0118 a 0x0137, y setenta y seis temporizadores de 100 mS, mapeados en los dos rangos 0x0138 a 0x016F y 0x018F a 0x01EF. Todos los temporizadores ocupan dieciséis bits, de los cuales, catorce bits <D13:D0> son utilizados para almacenar la cuenta de tiempo actual, el bit D14 es utilizado para la habilitación del temporizador, y el bit más significativo (D15) corresponde al bit T, para indicar si ya se cumplió el tiempo predeterminado. Los temporizadores en hardware "Timer0" y "Timer1" que posee el PIC16F877 fueron utilizados como bases de tiempo para la implementación de los temporizadores en software del iPLC.

Figura 5.4: "Formato de la palabra para los temporizadores del iPLC"

D15	D14	D13	D12	D1	D0
Bit T	bit de habilitación	Msb del valor actual del temporizador			Lsb del valor actual del temporizador

Los temporizadores son habilitados o inicializados por medio de las instrucciones correspondientes implementadas en el lenguaje de programación del iPLC. Esas instrucciones son utilizadas en el programa de usuario.

El acceso a los temporizadores puede ser a través del programa de usuario y a través del programa del iPLC (firmware), que se encarga de las actualizaciones automáticas de todos los registros en el área de los temporizadores.

1. **Actualización de los registros de temporizadores.** El sistema realiza las actualizaciones según las bases de tiempo ofrecidas por el controlador central. El temporizador 0 (timer0) que se encuentra implementado como un periférico interno en el controlador central (PIC16F877), fue configurado para que ofrezca una interrupción periódica en el sistema, con periodo T_{tI} = 1 mS. Esta interrupción fue utilizada como base de tiempo para los temporizadores de 1 y 10 milisegundos. En el caso de los temporizadores de 1 milisegundo, la actualización es directa, es decir que por cada interrupción de la base de tiempo, se realiza una actualización de todos los temporizadores de 1 milisegundo. La actualización de los temporizadores de 10 milisegundos se realiza de forma indirecta, utilizando una variable de contador, que se incrementa en una unidad por cada interrupción temporizada de 1 milisegundo, de tal forma que cuando esta variable alcanza el valor de 10 unidades, se realiza la actualización de todos los temporizadores de 10 milisegundos. Para la actualización de los temporizadores de 100 milisegundos se utilizó la base de tiempo ofrecida por el temporizador 1 del controlador central (timer1), configurado para provocar una interrupción al sistema cada 100 milisegundos.

Todas las actualizaciones de los temporizadores hacen uso de una rutina común, que se encarga de realizar un barrido de palabras (2 bytes) en rangos de la memoria RAM. Los rangos son parametrizados desde rutinas de nivel superior, correspondientes a las rutinas individuales para cada uno de los tipos de temporizadores (1, 10 y 100 mS). Durante el barrido se evalúa el bit número 15 de cada palabra (D14), que corresponde con el bit de habilitación de los temporizadores. Si este bit está en alto (temporizador habilitado), entonces se suma una unidad al valor del tiempo actual, de lo contrario el valor del tiempo actual no es modificado. Los temporizadores implementados en el iPLC poseen dos características importantes: No se altera el valor de los temporizadores cuando éstos no están habilitados, además mantienen activado su bit T luego de haberse cumplido el tiempo predeterminado. Esas dos características definen el tipo de todos los temporizadores del iPLC: Temporizadores de retardo a la conexión memorizado (Retentive On-Delay Timer), que es uno de los tipos de temporizadores comunes en los PLC's comerciales.

La utilización de las interrupciones en el sistema para generar las bases de tiempo, y efectuar las actualizaciones de los temporizadores, ofrece la ventaja de descentralización de esta tarea con respecto a las tareas cíclicas dentro del controlador central. El controlador central realiza las tareas cíclicas que correspondan según el modo de operación activo, y de forma independiente, el sistema se encarga de actualizar los valores de todos los temporizadores. Cuando se está en la fase de ejecución del programa de usuario, y una instrucción de programa desea acceder al valor de un temporizador, simplemente se lee el contenido del registro correspondiente, que almacena el dato actualizado por el sistema. De esta forma se logra que el sistema de actualizaciones para todos los temporizadores sea una tarea "transparente" para la ejecución del programa de usuario.

2. Los temporizadores y los diferentes modos de operación del iPLC. El funcionamiento de los temporizadores depende del modo de operación activo del iPLC. En el modo de ejecución, el sistema mantiene activa la función de actualización de todos los temporizadores, por supuesto los que estén habilitados. De esta forma, en este modo de operación, los registros de los temporizadores se mantienen en un proceso de lectura y escritura según se realicen las actualizaciones automáticas y según se ejecuten las instrucciones del programa de usuario.

Cuando el proceso que es controlado por el iPLC se detenga, los temporizadores también deben detenerse, y conservar el valor de tiempo actual. Por ello, en el modo detenido y en el submodo de programación se elimina la función de actualización de los temporizadores, de tal forma que únicamente cuando el proceso externo se esté ejecutando, se actualicen los temporizadores correspondientes a las variables de éste, y conserven el valor que corresponda cuando la operación se detenga. Con esto el iPLC puede estar operando en el modo de ejecución, luego pasar al modo detenido, y posteriormente reactivar su modo de ejecución, permitiendo que el proceso de la planta se detenga para algún tipo de servicio, revisión o por alguna falla, evitando que se pierda la información de temporización de los procesos en ejecución. Si la aplicación lo requiere, el programa de usuario podría borrar los temporizadores al entrar al modo detenido.

Durante la fase de inicialización del iPLC, luego de haber conectado la alimentación eléctrica, el sistema inicializa todos los temporizadores con el valor cero. Debido a esta característica, no es necesario que el programa de usuario inicialice la memoria de temporizadores. La excepción son aquellas aplicaciones que requieran borrar los temporizadores luego de una transición desde el modo detenido hacia el modo de ejecución.

H. El reloj de vigilancia del tiempo de ciclo (Watchdog Timer)

El sistema incluye un reloj que cuenta el tiempo de cada ciclo del iPLC. Este reloj es reinicializado con cero (0) cada vez que se inicia un nuevo ciclo. Durante la ejecución de cada ciclo, de forma independiente a éste y por medio de interrupciones cada 10 mS, se compara el valor de este reloj con el valor predeterminado de 500mS, si el tiempo del último ciclo (el actual) es mayor o igual que este valor predeterminado, entonces el sistema pasa al modo detenido y se realiza la notificación con el código de error correspondiente. Si en el diseño del sistema automático se contempla un tiempo de ciclo mayor o igual que 500 mS, entonces el reloj de vigilancia puede ser reinicializado por el programa de usuario utilizando la instrucción WDTR.

La función de temporización para este reloj se encuentra incorporada en la rutina de actualización de los temporizadores de 10 milisegundos. El reloj de vigilancia del tiempo de ciclo es básicamente un contador de periodos de 10 milisegundos. Cuando este contador alcanza un valor igual a 50, realiza la notificación de "tiempo de ciclo excedido", utilizando la función de notificación de errores incorporado en el sistema. La función de conteo para este reloj se deshabilita cuando el iPLC se encuentra en el modo detenido. No importa el valor que el reloj posea antes de entrar al modo detenido, ya que el sistema borra su contenido siempre que se realiza una transición entre los modos de ejecución y detenido, en cualquier sentido. El registro que almacena el valor predeterminado es una variable de sistema, por lo tanto no puede ser escrita desde el programa de usuario.

I. Registros de marcas especiales (SM)

Los registros de marcas especiales son variables de sistema. Cada una de las marcas especiales, al igual que el resto de variables de sistema, son escritas durante la ejecución de alguna de las fases del PLC, dependiendo de la funcionalidad que cada una tenga. Las marcas especiales no pueden ser escritas desde el programa de usuario.

Todos los registros de marcas especiales se encuentran implementados en la memoria RAM interna del microcontrolador central. La memoria RAM externa posee un mapeo de redireccionamiento. Cuando el programa de usuario desea acceder a la región de marcas especiales en la memoria RAM externa, el software de sistema ejecuta un redireccionamiento hacia las posiciones correspondientes en la memoria interna del controlador central. En estas posiciones de memoria se encuentran declaradas las variables de sistema que corresponden con los registros de marcas especiales.

1. El registro de estado del sistema (SM5). La marca especial 5 posee el registro de estado del iPLC, este registro es bastante útil ya que proporciona información acerca del modo de operación y de los errores que eventualmente pueden ocurrir en el sistema. Téngase presente que estos errores no son fallas del sistema, sino problemas presentados debido a condiciones ajenas al funcionamiento del mismo. En una sección posterior se tratará acerca de estos errores.

Este registro posee ocho bits. Los bits <D6:D5> almacenan el código para el modo activo del sistema. Los cuatro bits menos significativos almacenan el código del último error ocurrido en el sistema del iPLC. El bit más significativo está mapeado con el bit de activación para el LED bicolor que indica el modo de operación en la tarjeta madre. El bit D4 se encuentra reservado para versiones futuras.

El sistema utiliza este registro para operaciones internas, de cambio entre modos de operación, así como también para operaciones externas, de notificación en el sistema de monitorización.

Figura 5.5: "Formato del registro de estado del sistema (marca especial 5)"

D7	D6	D5	D4	D3	D2	D1	D0
LED	Msb del modo de operación	Lsb del modo de operación	Reservado	Código de error	Código de error	Código de error	Código de error

2. Registro de marca especial siempre activa (SM0). Este registro es escrito una sola vez, durante la fase de inicialización del sistema, momento en el cual se le asigna el valor 0xFF. Este registro fue implementado para su utilización con la instrucción "LD" (Load), cuando se desea ejecutar una instrucción de forma incondicional. Siempre que el programa de usuario acceda a este registro, a cualquiera de sus bits, se retornará un uno lógico, que generalmente será escrito en la pila lógica del sistema si se utiliza la instrucción LD.

La funcionalidad de este registro podría ser implementada con una instrucción que retornara siempre un uno lógico, sin embargo, esto sería incompatible con el lenguaje STL de la serie S7-200 de Siemens, de hecho sería incompatible con todos los lenguajes ofrecidos por esa empresa.

3. Registro de marca especial SM1. Este registro de sistema fue implementado de tal forma que contenga el valor 0xFF para el primer ciclo del PLC, y 0x00 para todos los demás ciclos. Cada vez que el sistema pasa del modo detenido al modo de ejecución, el registro SM1 es inicializado con el valor 0xFF, valor que conserva durante todo el primer ciclo. Posteriormente, cuando se inicia el segundo ciclo en el modo de ejecución, SM1 adquiere el valor 0x00, valor que conserva indefinidamente, durante todos los demás ciclos del modo de ejecución.

El valor 0xFF es asignado durante la fase de inicialización del sistema, y reasignado en cada uno de los ciclos del modo detenido. Estando en el ciclo del modo detenido, cuando el sistema detecta que se ha elegido un cambio hacia el modo de ejecución, se

asigna el valor 0x00 a SM1, valor que es reasignado al final de cada ciclo, mientras el sistema permanezca en modo de ejecución.

4. Registro de marca especial para el tiempo de ciclo (SM2). Este registro está implementado como un contador de periodos de 1 milisegundo. El contador se incrementa cada 1 milisegundo durante el ciclo del PLC. Cuando se inicia la ejecución de un nuevo ciclo, el contador es reinicializado con el valor cero (0).

Para la temporización se utilizó la base de tiempo de los temporizadores de 1 milisegundo. El contador SM2 se incrementa cada vez que se ejecuta la rutina de actualización para esos temporizadores. Puesto que en el modo detenido los temporizadores no son actualizados, el registro SM2 tampoco es alterado en este modo de operación. Únicamente interesa saber cuál es el tiempo de ciclo del PLC cuando se encuentra en el modo de ejecución, que está determinado principalmente por el tiempo que ocupe la fase de ejecución del programa de usuario. En los demás modos de operación SM2 no debe ser alterado, de tal forma que únicamente conserve su valor con propósitos de monitorización.

5. Registros de marca especial para los tiempos máximo y mínimo del ciclo del PLC (SM3 y SM4). El registro SM3 almacena el tiempo del ciclo más corto que se ha presentado desde la última transición desde el modo detenido hacia el modo de ejecución. Mientras tanto, el registro SM4 almacena el tiempo del ciclo más largo. Ambos registros son inicializados con el mismo valor y actualizados dentro de una misma rutina.

Al final de cada ciclo del modo de ejecución se llama a una rutina que se encarga de actualizar los registros SM3 y SM4. Esta rutina determina si el sistema está finalizando la ejecución de su primer ciclo, de ser así, escribe el valor del tiempo de ciclo actual en ambos registros. Esto funciona como una inicialización adecuada, pues para todo primer ciclo en el modo de ejecución, el tiempo de ciclo máximo y el tiempo de ciclo mínimo son el mismo: el único tiempo de ciclo. No se pueden tomar valores iniciales arbitrarios,

pues de no presentarse tiempos fuera del rango arbitrario, los datos ofrecidos por los registros SM3 y SM4 serían erróneos. Luego, para todos los demás ciclos del modo de ejecución, la rutina de actualización compara el valor del tiempo de ciclo actual, con los valores de SM3 y SM4, actualizándolos según corresponda, si el tiempo de ciclo actual es el nuevo mínimo o el nuevo máximo.

6. **Registro de marca especial de reloj de ciclo (SM6).** Este registro es inicializado con el valor 0x00 durante la fase de inicialización del sistema. Luego es complementado al inicio de cada ciclo del modo de ejecución. De esta forma, SM6 ofrece una señal que se complementa en sincronía con cada ciclo del PLC (en modo de ejecución).

Este registro puede ser útil para las tareas que necesiten tomar en cuenta el número de ciclos ejecutados por el PLC. En el modo detenido este registro no es complementado ya que no interesa contar o ejecutar tareas en sincronía con los ciclos de este modo de operación.

J. Fase de inicialización del sistema

Esta fase se lleva a cabo cada vez que el iPLC es encendido, inmediatamente después de aplicarle la alimentación eléctrica (24 Vdc). Durante esta fase, el iPLC realiza las tareas necesarias para la inicialización de todos los periféricos conectados al sistema, incluyendo a los periféricos instalados en la tarjeta madre y a los módulos instalados por medio del bus de expansión. La CPU central inicializa la memoria RAM del sistema, inicializa a todos sus periféricos, ya sean de hardware o de software, tanto los integrados en el microcontrolador central, como los externos, y además realiza la detección de todos los módulos que se encuentren conectados al bus de expansión, los cuales le envían la información acerca del tipo de módulo, para que esto sea almacenado y posteriormente visualizado en el entorno de monitorización. En esta fase también se inicializan las variables de sistema.

Inmediatamente después de aplicar la alimentación eléctrica al iPLC, éste entra en su fase de inicialización, la cual ocupa aproximadamente 2.3 segundos, por lo que el sistema debe estar libre de comandos de usuario durante ese periodo.

En la inicialización del sistema se ejecuta una lista de tareas secuenciales que se detalla a continuación.

1. **Configuración del módulo SPI maestro.** Se realiza la configuración y activación del periférico para el módulo maestro de la comunicación SPI (Serial Peripheral Interface). Este periférico se encuentra integrado en el microcontrolador central. Primero se configuran las señales de control y datos, posteriormente se realiza una espera de 1.03 segundos con el propósito de permitir la inicialización de los módulos que se encuentren conectados al bus de expansión.

2. **Configuración del módulo USART (Universal Synchronous / Asynchronous Receiver / Transmitter).** Se configura el módulo USART integrado en el microcontrolador central para que opere como un canal de datos asíncrono. Primero se configura un canal a 9.6 Kbps, necesario para la comunicación con el módulo SitePlayer. Posteriormente se transmiten los comandos necesarios para reconfigurar el puerto serial del módulo SitePlayer, de 9.6 Kbps a 115.2 Kbps. Finalmente se reconfigura el módulo USART del microcontrolador central, para que opere a 115.2 Kbps, de tal forma que pueda continuar comunicándose con el módulo SitePlayer, a la máxima tasa de transferencia que éste soporta.

3. **Configuración de las señales de control.** Se configuran todos los bits que serán utilizados como señales de control del sistema. Todos estos bits pertenecen a los diferentes puertos de entrada / salida integrados en el microcontrolador central.

En la configuración se define, para cada bit de control, si debe ser un pin de entrada o salida, y se asigna el valor inicial que cada señal de control debe poseer para que el arranque del sistema presente las condiciones adecuadas.

4. Configuración del coprocesador matemático. Se envía una señal de "Reset" al coprocesador matemático, para que éste inicialice su canal de comunicaciones. Luego se realiza una espera de 25.5 milisegundos, necesarios para que el coprocesador matemático inicialice sus registros internos. Finalmente se elimina la función de truncado y se elige la función de aproximación para el manejo de enteros en el coprocesador matemático.

5. Configuración de los temporizadores. Se inicializa la región de memoria asignada para los noventa y seis temporizadores del sistema, escribiendo el valor 0x0000 en todos sus registros. Además se configuran los módulos "Timer0" y "Timer1", que corresponden a los dos periféricos de temporizadores integrados en el microcontrolador central, que funcionan como las bases de tiempo para todos los temporizadores del sistema.

6. Inicialización de los contadores. Se inicializa la región de memoria asignada a los cuarenta contadores del sistema, escribiendo el valor 0x0000 en todos sus registros.

7. Selección del modo de operación. El sistema debe iniciarse en el modo detenido, por ello se ejecuta una orden para la activación de este modo. La orden debe provocar que el módulo SitePlayer seleccione el código correspondiente para el modo detenido, y lo comunique al microcontrolador central. Para ello se realiza la transmisión de comandos que escriben el código del modo detenido, directamente en los bits correspondientes del puerto en el módulo SitePlayer. De esta forma se fuerza a que el sistema detecte la selección del modo detenido y provoque la activación interna de éste, dentro de su ciclo de operación.

8. Inicialización de los registros de flancos. Los registros que se utilizan para la detección de flancos deben estar correctamente inicializados, de lo contrario se puede provocar alguna detección falsa. El área para los registros de flancos ascendentes es inicializada con 0xFF en cada uno de sus registros, mientras que el área para los

registros de flancos descendentes es inicializada con 0x00. Ambos valores de inicialización garantizan la correcta detección del primer flanco (positivo o negativo).

9. Detección de los módulos conectados al bus de expansión. Se generan comandos de solicitud de información que son transmitidos a través del bus de expansión del sistema. Los comandos solicitan el tipo de los módulos que se encuentren conectados al bus de expansión. Se utiliza un ciclo en el cual se transmite el comando de solicitud hacia cada una de las direcciones existentes, desde la dirección 0x01 hasta la dirección 0x10. Con ese ciclo se hace un "barrido" de todos los posibles módulos instalados. Todos los módulos que se encuentren presentes, responden con el código correspondiente a su tipo. Luego, esa información recibida por el microcontrolador central es transmitida hacia la memoria RAM del servidor HTTP, para que permanezca almacenada allí, y desde el entorno de monitorización pueda ser utilizada para consultar la información acerca de todos los módulos de expansión instalados en el sistema.

10. Inicialización de las variables de sistema. La pila de programa (Program Stack) se inicializa con 0x00, pues al inicio aún no se han cargado posiciones de programa en ella. Otras variables de sistema que se "borran" con el valor de inicialización 0x00 son el Reloj de vigilancia de tiempo de ciclo, el tiempo de ciclo, la marca especial SM6 y el registro de la pila lógica, entre otras.

Algunas variables requieren de un valor inicial diferente, como por ejemplo, las marcas especiales SM0 y SM1, o el byte de estado del sistema, SM5. No se entrará en mayor detalle acerca de la inicialización de otras variables de sistema, pues se considera que no es pertinente.

11. Configuración de las interrupciones del sistema. Al final de la fase de inicialización se configuran todas las interrupciones del sistema, realizando habilitaciones / deshabilitaciones según se requiera para que el sistema responda de la forma adecuada a las condiciones iniciales que se presentan durante la ejecución de su primer ciclo en el modo detenido.

Previamente, al inicio de la fase de inicialización, se deshabilitan todas las interrupciones del sistema, de tal forma que las tareas de inicialización no sean interrumpidas por condiciones iniciales que en realidad no interesan.

K. Fase de lectura de las entradas digitales

Como primera fase dentro del proceso cíclico del sistema, el iPLC realiza la lectura de las entradas digitales presentes en los módulos de expansión instalados. Los datos leídos son escritos en los *registros de imagen del proceso para las entradas digitales*, en una región de memoria RAM reservada para tal efecto (memoria DI). Luego, todas las instrucciones del programa de usuario que deseen acceder a estas entradas digitales tomarán los valores que hayan sido escritos en dichos registros de imagen.

Esta fase es ejecutada por una rutina principal del software de sistema, llamada "Read digital inputs phase". Esta rutina ejecuta un ciclo en el cual se comunica con todos los módulos de expansión y solicita la lectura de todas las entradas digitales instaladas.

El software de sistema utiliza un ciclo de comunicación que "barre" todas las direcciones del bus de expansión. Se solicitan datos a todas las direcciones, pero únicamente los módulos instalados enviarán información válida hacia el microcontrolador central, el cual leerá 0xFF para todos los bytes de entradas digitales en las direcciones libres del bus de expansión. Estos datos también son escritos en la memoria DI, en los registros correspondientes.

L. Fase de ejecución del programa de usuario

Luego de haber ejecutado la fase de lectura de las entradas digitales, el iPLC se traslada a la fase de ejecución del programa de usuario, que previamente fue cargado al iPLC utilizando el entorno de programación, desde un explorador de Internet. En esta

fase el iPLC ejecuta una por una las instrucciones del programa que se encuentra almacenado en la memoria flash externa del sistema, por lo que hace uso de los buses externos de datos y direcciones para leer los códigos de programa almacenados en flash y para leer / escribir datos en la memoria RAM. Durante esta fase del ciclo se actualizan los *registros de imagen de las salidas digitales*, según se vayan modificando por las instrucciones ejecutadas, nótese que aún no se actualizan las salidas físicas. Todos los demás tipos de entradas y salidas instaladas en módulos de expansión, como las analógicas de voltaje, corriente, u otras especiales, son leídas y actualizadas físicamente por las instrucciones correspondientes durante la ejecución del programa de usuario, tal como sucede en los sistemas de automatización de la serie S7-200 de Siemens.

Como se había mencionado en una sección previa, esta fase consiste básicamente en la ejecución del microprocesador virtual. Este dispositivo virtual se encarga del procesamiento de todos los códigos de programa almacenados en la memoria flash. Este procesamiento constituye la ejecución del programa de usuario.

El microprocesador virtual lee el código de la primera instrucción de programa, posteriormente lee todos los argumentos de la instrucción, y finalmente efectúa el procesamiento de ésta. Todas las instrucciones del iPLC están implementadas en el firmware del microcontrolador central, de tal forma que todo el procesamiento se realiza dentro de éste. Luego que el microprocesador virtual lee todos los códigos para una instrucción, ejecuta la rutina implementada en el firmware para su procesamiento. Después de realizar el procesamiento de la primera instrucción, la ejecución retorna desde la rutina de la instrucción que se procesó, hacia el ciclo principal que ejecuta todos los circuitos del programa de usuario (leer la sección "Funcionamiento del microprocesador virtual", previamente expuesta). Luego de procesar la última instrucción del último circuito en el programa de usuario, esta fase concluye y el control del programa retorna hacia el ciclo del modo de operación activo en el sistema. El final del programa de usuario se determina por medio del código de instrucción especial 0x7E, que siempre es el último código almacenado en la memoria flash.

M. Fase de actualización de las salidas digitales

Durante esta tercera fase el sistema realiza la actualización física de los estados de las salidas digitales en los módulos de expansión instalados. La actualización se realiza a través de la escritura de los registros de imagen de las salidas digitales, en los registros de salidas digitales físicas en los módulos de expansión presentes. Para ello, la CPU central debe enviar el comando correspondiente, a través del bus de expansión, seguido por los datos a escribir en las salidas físicas del sistema, todo dentro de un ciclo que comunica a la CPU central con cada dirección del bus de expansión.

El software de sistema utiliza un ciclo de comunicación que "barre" todas las direcciones del bus de expansión. Se escriben datos a todas las direcciones, pero únicamente los módulos instalados recibirán y actualizarán el estado de sus salidas digitales físicas.

N. Fase de monitorización

Esta es la fase del ciclo que procesa las solicitudes de variables a monitorizar, efectuadas desde el entorno de monitorización del sistema de software del servidor HTTP. Para optimizar el tiempo de ciclo del iPLC, el sistema procesa únicamente una variable de monitorización por cada ciclo del iPLC, de tal forma que todas las variables solicitadas puedan ser procesadas en un periodo adecuado, para su posterior visualización en la página HTML correspondiente, sin afectar drásticamente el tiempo de ciclo del sistema.

Se pueden realizar solicitudes de monitorización para todas las variables del sistema. El sistema de software permite monitorizar el contenido de los 32KB de la memoria RAM, aún estando en modo detenido, característica que resulta muy útil ya sea en los casos que se presentan errores, o en casos en los que simplemente se desea examinar

ciertos parámetros o variables en la planta de producción luego de haberse detenido el proceso.

La rutina de monitorización utiliza un apuntador para procesar todas las variables a monitorizar, una por una según el turno correspondiente. Todas las variables a monitorizar están almacenadas en una lista en la memoria RAM del servidor HTTP. Cada vez que se ejecuta la rutina de monitorización, en el software de sistema, el apuntador accede a la región de solicitudes de monitorización, en la RAM del servidor HTTP, y proporciona la dirección y longitud de la variable a monitorizar, que debe estar en el rango de los 32 KB de memoria RAM del sistema. Con esa información, se procede a leer la variable mapeada en la memoria RAM externa. Los bytes de la variable son enviados hacia el servidor HTTP, donde son almacenados en su memoria RAM, en la región correspondiente a los datos que se desplegarán en la página de monitorización.

Se puede monitorizar hasta un máximo de 40 variables de forma simultánea. El procedimiento anterior es realizado una vez por cada ciclo de operación, de tal forma que todas las solicitudes de monitorización pueden ser atendidas en un máximo de 40 ciclos del iPLC. Nótese que el tiempo de ciclo del iPLC típicamente será de algunas decenas de milisegundos, por lo que todas las solicitudes de monitorización pueden ser atendidas en un periodo de tiempo adecuado para su actualización en la página de monitorización.

Esta es la última fase que se ejecuta en el modo de ejecución, y la única en el modo detenido.

O. Software de los módulos de expansión

Todos los módulos poseen un software común: El software del módulo de expansión modelo. Este funciona como el "corazón" del sistema de software para todos los módulos de expansión que se pueden conectar con el iPLC. La variación entre los diferentes módulos de expansión es la electrónica para interfasar al microcontrolador

PIC16F877 con el tipo de entradas / salidas especiales que se hayan implementado, y el segmento de software que se desarrolle para la interacción con esa electrónica.

1. **Software del módulo de expansión modelo.** Este software está diseñado para ser utilizado como la base del firmware que se cargue en el microcontrolador de cada módulo de expansión. En este software se encuentra todo lo necesario para "transformar" a un microcontrolador PIC16F877 en el controlador central de cualquier módulo de expansión.

Este software se basa en un esquema de comandos. Toda la comunicación entre la CPU central y cualquier módulo de expansión está abstraída en forma de *transacciones.* Estas transacciones inician con el envío de un comando que solicita la realización de una transacción específica con el módulo de expansión que posee una dirección específica. La información completa está indicada en el comando que se transmite.

Todos los módulos de expansión en un sistema de PLC no hacen más que expandir las capacidades del sistema, ofreciendo algún tipo de entrada / salida digital / analógica o algún otro tipo especial e incluso mucho más complejo. De cualquier forma, todo lo que ofrecen los módulos de expansión puede verse como puntos de entrada, salida, o propiedades, a los cuales se desea leer o escribir. Con este esquema de comandos / transacciones, lo único que se requiere es que todo acceso a un punto en un módulo de expansión sea visto como una transacción en la cual desea realizarse una lectura o escritura en dicho punto. De esta forma, para acceder a toda la funcionalidad de los módulos de expansión instalados, no importa a qué tipo de módulo de expansión se desee acceder, simplemente se necesita definir todos los tipos de transacciones (comandos) posibles en el firmware del iPLC, y que cada módulo de expansión posea las rutinas de servicio para los tipos de los puntos de acceso propios, que serán solicitados desde la CPU central, por medio del comando correspondiente.

En el software del sistema cada comando es en realidad manejado como un *tipo de transacción*, y cada tipo de transacción tiene un código único. Este código único es el

que finalmente se transmite a través del bus para comunicarse con los módulos de expansión.

Finalmente, con este esqueleto del software, lo único que cada tipo de módulo de expansión debe agregar son las rutinas para el servicio de todos los tipos de transacciones que es capaz de atender. De esta forma, un módulo de expansión con entradas y salidas digitales debe agregar dos rutinas: Una para la lectura de entradas digitales y otra para la escritura en salidas digitales.

Cada una de las rutinas de servicio está completa y exclusivamente definida por la electrónica específica de cada módulo de expansión, que a su vez define el tipo del mismo. Por lo tanto, para diseñar nuevos módulos de expansión, solo se requiere diseñar los circuitos adicionales al módulo de expansión modelo, y las rutinas que se comuniquen con dichos circuitos, sin ocuparse de la interacción con la CPU central.

a. Inicialización del módulo de expansión modelo. Inmediatamente después de aplicar la alimentación eléctrica se ejecutan las tareas necesarias para configurar todos los periféricos y registros internos, de tal forma que el módulo de expansión esté listo para establecer comunicación con la CPU central. La secuencia de tareas es como se presenta a continuación.

1) Configuración del puerto serial asíncrono. Todos los módulos de expansión utilizan un puerto serial asíncrono ofrecido por su microcontrolador. Este puerto se configura a 9.6 Kbps, y se utiliza para la asignación de direcciones.

2) Configuración del canal de comunicación. Se utiliza el periférico de comunicación SPI (Serial Peripheral Interface) interno en el PIC16F877 para la conexión con el bus de expansión. En esta etapa se configura este dispositivo, de tal forma que el controlador del módulo de expansión pueda operar como un dispositivo SPI esclavo, que será gobernado por el microcontrolador de la CPU central (SPI maestro).

3) Configuración e inicialización de las señales de control. Se configuran los pines que serán utilizados como señales de control. Además se asignan los valores requeridos a cada una de las variables de control para que el módulo de expansión pueda comunicarse con los módulos adyacentes. Esto es necesario para la asignación de direcciones.

4) Asignación de direcciones. Todos los módulos conectados al bus de expansión reciben una dirección al momento de inicializarse. La asignación de direcciones es de forma dinámica. El procedimiento de asignación es de la forma siguiente: Cada módulo posee una señal de entrada denominada EM_{IN} y una señal de salida EM_{OUT}. Todos los módulos son conectados en cascada. La salida EMout de cada módulo se conecta con la entrada EMin de su módulo adyacente. De esta forma todos los módulos, excepto el último, tendrán sus entradas EMin conectadas a la salida EMout del módulo adyacente correspondiente. Durante la etapa de inicialización de las señales de control, cada módulo coloca en bajo su señal EMout. Luego de un tiempo de espera igual a 0.3 segundos cada módulo evalúa el estado de su señal EMin. Todos los módulos, excepto el último, leerán un cero lógico en su señal EMin. Así, el módulo que lea un uno lógico se asigna automáticamente la dirección 0x01 y todos los demás módulos ingresan en un ciclo de espera hasta recibir un byte en su puerto serial asíncrono. Luego que el último módulo adquirió la dirección Y = 0x01, transmite el valor Y + 1 hacia el módulo adyacente. Cada módulo, al recibir un byte serial X, efectúa la transmisión X + 1 hacia su módulo adyacente. De esa forma, el último módulo inicia una cadena de transmisiones seriales, que consigue la asignación de direcciones a todos los módulos conectados al bus, desde la dirección 0x01 hasta la dirección 0x0k, donde k es la cantidad de módulos conectados. Esta es la forma en que se asignan las direcciones a cada uno de los módulos de expansión.

b. El ciclo del esclavo. Luego de ejecutar la fase de inicialización, el módulo de expansión modelo se traslada hacia el *ciclo SPI esclavo*. Este ciclo mantiene una revisión continua del canal SPI, esperando hasta que la CPU central transmita un byte. La aparición de ese byte indica el inicio de una transacción, por lo que la ejecución del

programa se traslada hacia una rutina en la cual se determina si la transacción debe realizarse con este módulo. De ser así se atiende la transacción, de lo contrario ésta se ignora.

c. Atención de una transacción. Luego de salir del ciclo SPI esclavo, cuando la transacción debe ser atendida, el software del módulo de expansión modelo extrae la información acerca del tipo de la transacción que se desea efectuar. Dependiendo de ese tipo de transacción, la ejecución del programa se traslada hacia la rutina de servicio correspondiente, que implementa dicha transacción. Al finalizar la rutina de servicio, el módulo de expansión retorna al ciclo SPI esclavo para esperar la próxima transacción.

d. Identificación del tipo de módulo de expansión. El software del módulo de expansión modelo implementa una única rutina de servicio. Todas las demás rutinas de servicio deben ser ofrecidas por los módulos de expansión específicos. Esta rutina atiende la transacción que solicita información acerca del tipo del módulo de expansión. Esta transacción es necesaria al inicializar el iPLC, por ello, todos los módulos deben ser capaces de atenderla, razón por la cual se incluye en el módulo de expansión modelo.

2. Software del módulo de expansión digital implementado. Consiste en el software del módulo de expansión modelo más dos rutinas de servicio, una para lectura de las entradas digitales (Read digital inputs) y otra para escritura en las salidas digitales (Write digital outputs). En la rutina de lectura simplemente se lee el estado del puerto B en el microcontrolador PIC16F877 del módulo de expansión, al cual se encuentran conectados los puntos de entrada físicos, y se envía el byte resultante como un byte de datos hacia la CPU central, la cual escribirá este byte en el registro imagen correspondiente.

Por otra parte, la rutina de escritura recibe un byte enviado por la CPU central y lo escribe en el búfer de salida del puerto D, al cual se encuentran conectados los puntos de salida físicos.

3. Software del módulo de expansión analógico implementado. Consiste en el software del módulo de expansión modelo más dos rutinas de servicio, una para lectura de las entradas analógicas (Read analog input), y otra para escritura en las salidas analógicas (Write analog output). La rutina de lectura recibe el número del elemento al cual se desea acceder desde la CPU central, realiza una conversión analógico – digital en el punto correspondiente, y envía el byte resultante hacia la CPU central. La rutina de escritura recibe el número y valor del elemento al cual se desea escribir, escribe el valor en el registro de salida correspondiente con el número de elemento, y finalmente realiza una conversión digital – analógico para dicho registro.

P. La comunicación con los módulos de expansión

El *protocolo de comunicación* entre la CPU central y los módulos de expansión fue "montado" sobre el *protocolo de transmisión* SPI (Serial Peripheral Interface) a 5 Mbps. Este protocolo de transmisión es ofrecido por un periférico interno en el microcontrolador central, por lo que únicamente se realiza la configuración de éste en la fase de inicialización del sistema.

1. El protocolo de comunicación. Todas las transacciones entre la CPU central y los módulos de expansión se llevan a cabo por medio de un protocolo basado en la transmisión de una secuencia de tres bytes. Esta secuencia es enviada desde la CPU central, hacia todos los módulos de expansión, a través del bus de expansión del sistema.

Los bytes han sido denominados *byte de transacción, byte de elemento* y *byte de datos*. El byte de transacción posee el código de la transacción que se desea realizar. El byte de elemento indica el número del elemento al cual se desea acceder en el módulo de expansión, es decir el número de entrada / salida física en el módulo de expansión. El último byte contiene los datos de la transacción. Los accesos a elementos de los módulos de expansión pueden deberse a dos necesidades: Ejecutar instrucciones del programa de

usuario que poseen como parámetros a estos elementos, y ejecutar tareas de sistema como las fases de inicialización, lectura o actualización. El protocolo funciona de la siguiente forma: Cuando se ejecuta una instrucción que requiere acceder a un elemento en un módulo de expansión (entrada / salida digital / analógica, etc.), la CPU genera un byte que posee el código de la transacción que se desea efectuar, en los cuatro bits más significativos, y coloca la dirección del módulo a acceder en los restantes cuatro bits, este es el byte de transacción. Luego envía un byte para indicar cuál es el número de elemento al que desea acceder. Posteriormente espera hasta que el módulo de expansión responda con la indicación de estar listo para realizar la transacción (señal Busy Module en el bus de expansión), cuando dicha señal aparece, la CPU central efectúa la transacción (transmisión / recepción de datos).

Finalizada la transacción, tanto la CPU como el módulo de expansión liberan el bus para permitir futuras transacciones. Si el comando no existe en alguno de los módulos de expansión instalados, o el elemento a acceder no está presente, se genera una condición de error, que es notificada por medio de LED's indicadores, tanto en la CPU como en la página del entorno de monitorización. La activación de la señal de respuesta "Busy Module" puede ocurrir antes o después del último byte de la transacción, dependiendo del tipo de ésta. Por ejemplo, en una transacción de lectura analógica, la señal BM se activa cuando la conversión analógico – digital ha sido finalizada y el dato (último byte) está listo para ser transmitido. En cambio, en una transacción de escritura analógica, la señal BM se activa cuando la conversión digital – analógico ha sido finalizada y el dato (último byte) fue correctamente escrito en la salida analógica correspondiente.

Figura 5.6: "Byte de inicio de transacciones entre la CPU y los módulos de expansión"

[Comando de transacción requerida] [Dirección del módulo a acceder]

D7	D6	D5	D4	D3	D2	D1	D0

Como se puede ver en la figura 5.6, el diseño del sistema contempla un máximo de 16 tipos de transacciones y 15 módulos de expansión, pues la dirección 0000b está reservada para versiones futuras, en las cuales probablemente esa dirección corresponda a entradas / salidas especiales instaladas en la tarjeta madre. Se dispone de un byte completo para indicar el número de elemento a acceder, por lo que cada módulo de expansión puede tener hasta 256 puntos de entrada y salida, ambos simultáneamente, pues el código de transacción para acceder a las entradas es distinto que el código para acceder a las salidas.

Tabla 5.1: "Códigos de transacción entre la CPU central y los módulos de expansión"

Código	**Tipo de transacción**
0x00	Solicitud de información acerca del tipo de módulo
0x01	Leer una entrada digital
0x02	Escribir una salida digital
0x03	Leer una entrada analógica
0x04	Escribir una salida analógica
0x05 a 0x0F	Disponibles para módulos de expansión con entradas / salidas especiales

Cuando la CPU central solicita una transacción, luego de enviar el código de la transacción y la dirección del módulo que desea acceder, espera a recibir la señal de aceptación para la transacción, que debe ser enviada desde el módulo de expansión direccionado. El tiempo de espera máximo es $T_{Bwmax} = 357$ microsegundos. Vencido el tiempo T_{Bwmax} la CPU central cancela la transacción y realiza la notificación del error correspondiente (código de error # 3).

2. Estrategia para evitar colisiones en el bus de expansión. Todos los módulos de expansión poseen su propia fase de inicialización. Una de las tareas que realizan en esa fase es la configuración de su propio periférico SPI. En esta fase, configuran su pin SDO (salida de datos) como un pin de entrada, con lo cual se inhibe la transmisión de datos. Puesto que todos los módulos de expansión conectados al bus realizan esta configuración inicial, el bus de expansión se encuentra libre. Todos los

módulos conectados al bus se mantienen en un ciclo en el cual "escuchan" todo lo que se transmite. Cuando la CPU central realiza una solicitud de transacción, luego de enviar el código de la transacción y la dirección del módulo a acceder, cada módulo determina si la transacción debe realizarse con él. Únicamente el módulo direccionado envía la señal de respuesta y reconfigura su pin SDO como un pin de salida, con lo cual adquiere la capacidad para transmitir a través del bus de expansión. Cuando la transacción finaliza, el pin SDO es reconfigurado como un pin de entrada, liberando así el bus, para que pueda realizarse una nueva transacción con cualquier otro módulo.

Q. Errores en el sistema

Cuando se habla de errores en un sistema de PLC, se hace referencia a aquellos errores que son ajenos al funcionamiento del sistema del PLC, y en cambio, son provocados por condiciones del programa de usuario. En el diseño del sistema de software se consideraron todos los errores que pueden ocurrir durante la ejecución del programa de usuario, y algunos errores especiales de reprogramación. Este conjunto de errores está plenamente caracterizado, detectado y notificado por el firmware del iPLC. La mayoría de los errores son en tiempo de ejecución, aunque existen dos debidos a problemas de reprogramación. Todos los errores se listan en la tabla 5.2.

1. Detección de errores. Existen diferentes tipos de errores, y la situación de ocurrencia para cada tipo de error está plenamente definida, de tal forma que se tiene conocimiento acera de las circunstancias necesarias para que el error se produzca. Es decir, se sabe que tipo de error puede ocurrir en cada una de las subrutinas del software de sistema. Esto conlleva a la capacidad para detectar la ocurrencia de errores por medio de la verificación correspondiente en cada una de las rutinas de funcionamiento del software de sistema. Cada rutina del software de sistema posee una subrutina de verificación para la ocurrencia del tipo de error específico que se espera pueda presentarse.

2. Manejo de errores. Luego de haber detectado un error, se utiliza una rutina de *notificación de errores* que se encarga de proporcionar el código del error detectado. Este código binario es mostrado en los LED's de Estado de la tarjeta madre, y además son transmitidos hacia el servidor HTTP para su visualización en el entorno de monitorización. En el momento de la detección, el sistema realiza la notificación e inmediatamente pasa al modo detenido. De esta forma se coloca en un estado seguro y se permite que la persona quien monitoriza remotamente pueda determinar cuál fue la causa del error, observando los LED's gráficos para el código de error, y monitorizando todas las variables del proceso. Luego de la ocurrencia de algún error, el usuario puede ordenar que el sistema reactive el modo de ejecución, sin embargo, si la condición de error aún persiste, el sistema volverá al modo detenido automáticamente y volverá a realizar la notificación correspondiente. Para que el sistema pueda continuar con la operación normal en el modo de ejecución es indispensable que la condición de error desaparezca.

3. Tipos de errores.

Tabla 5.2: "Códigos de error del sistema"

Código (binario)	Error
0000	No han ocurrido errores.
0001	No se pudo borrar la memoria flash.
0010	No se pudo grabar en la memoria flash.
0011	Error de entrada / salida
0100	Stack overflow.
0101	Stack underflow.
0110	Error de vigilancia.
0111	Número entero demasiado grande.
1000	Número real demasiado grande.
1001	Número real demasiado pequeño.
1010	División entre cero.
1011	Error numérico.
1100	Error de dominio.

a. No se pudo borrar la memoria flash. El nombre del error ofrece una descripción clara. La operación de borrado debe ser ejecutada cada vez que se desee reprogramar al iPLC. Cuando sucede este error lo más probable es que el circuito integrado de memoria flash esté dañado, por alguna condición extrema, o porque finalizó su vida útil (típicamente sus ciclos de reescritura).

b. No se pudo grabar la memoria flash. Este error ocurre cuando al menos uno de los códigos de programa, generados a partir de la compilación del código fuente ingresado por el usuario, no pudo ser almacenado en la localidad correspondiente en la memoria flash de la tarjeta madre. Las razones de ocurrencia son las mismas que para el error anterior.

c. Error de entrada / salida. Se refiere a que el sistema no ha podido acceder al elemento deseado en un módulo de expansión. Este error ocurre cuando se intenta acceder a un punto de entrada / salida que no existe. Puede ser que el módulo de expansión direccionado no posea ese número de elemento, es decir, que esté fuera de rango, o que el módulo mismo no haya sido detectado.

d. Stack overflow. Ocurre cuando se carga una dirección en la pila de programa (Stack), estando ésta llena. Este error es provocado exclusivamente por la instrucción 'CALLSBR' que transfiere el control del programa a otra localidad, y carga en pila de programa la localidad actual, a la que debe retornar.

e. Stack underflow. Ocurre cuando se intenta extraer una dirección de la pila de programa, estando ésta vacía. Es provocado exclusivamente por las instrucciones 'RET' y 'CRET', que trasladan la ejecución del programa hacia la dirección que extraen de la pila.

f. Error de vigilancia. Ocurre cuando el tiempo del ciclo de operación del sistema en modo de ejecución excede de los 500 milisegundos, predeterminados en una variable de sistema.

g. Número entero demasiado grande. Ocurre cuando el coprocesador matemático realiza una operación con números enteros, cuyo resultado está fuera del rango de manipulación (24 bits). La operación puede ser una función matemática o la conversión de un número de punto flotante en un número entero.

h. Número real demasiado grande / pequeño. Ocurre cuando el coprocesador matemático realiza una operación con números de punto flotante, cuyo resultado está fuera del rango de manipulación ($\pm$ 6.80564693 x 10^{38} y $\pm$ 1.17549435 x 10^{-38}).

i. División entre cero. Ocurre cuando el coprocesador matemático intenta realizar una división entre cero.

j. Error numérico. Ocurre cuando el coprocesador matemático intenta realizar una operación con un dato no numérico. Esto puede ocurrir cuando se intenta operar con un argumento que no cumple con el formato IEEE754.

k. Error de dominio. Ocurre cuando el coprocesador matemático intenta realizar una operación con un argumento cuyo valor se encuentra fuera del dominio de la función matemática a ejecutar.

VI. SISTEMA DE SOFTWARE DEL SERVIDOR HTTP

El sistema de software del servidor HTTP tiene dos objetivos específicos:

- Ofrecer un *entorno de programación* que permita la reprogramación remota del iPLC.
- Ofrecer un *entorno de monitorización* que permita la monitorización remota del estado de operación del iPLC y el contenido de todas sus variables.

Toda la estructura del servidor fue desarrollada para satisfacer esos dos objetivos. De tal forma que se implementó un sistema de monitorización basado en dos páginas HTML y un sistema de programación basado en una sola página HTML. Ambos sistemas están desarrollados sobre HTML y JavaScript. El código HTML es utilizado para la presentación de todos los elementos en las páginas de Internet, mientras que el código JavaScript es utilizado para la implementación de todas las funciones de monitorización y programación. En otras palabras, los programas desarrollados en lenguaje JavaScript constituyen el "corazón" de los dos sistemas, mientras que el código HTML únicamente presenta resultados y ofrece campos de entrada de datos para la interacción con el usuario. Todos los campos de entrada son parámetros para las funciones del código JavaScript, y todos los resultados presentados en la pantalla del explorador de Internet son el resultado del procesamiento con las funciones del código JavaScript.

A. El archivo de definición del servidor HTTP

En este archivo se encuentra la configuración del servidor HTTP, y la definición de todos los objetos. En la sección de configuración se utilizan directivas de compilador para definir el valor de diversos parámetros. En esta sección se encuentra la siguiente configuración:

- *$DHCP on.* Esta directiva permite que el módulo SitePlayer obtenga una dirección IP desde un servidor DHCP

- *$DeviceName "iPLC".* Define un nombre para identificar al módulo SitePlayer en una red interna. Para utilizar esta característica se debe instalar un programa especial en las estaciones de trabajo conectadas a la red, desde las cuales se desea detectar al dispositivo.

- *$DownloadPassword "Esquit".* Establece a "Esquit" como la clave de autorización para cargar datos en la memoria flash interna del microcontrolador 89C51RD2.

- *$Include "C:\Archivos de Programa\SitePlayer\pcadefv2.inc".* Hace que el compilador incluya al archivo "pcadefv2.inc" al final de este archivo de definición, de tal forma que lo tome en cuenta para la generación del *archivo binario de imagen.* El archivo "pcadefv2" posee más definiciones de *objetos* especiales, como por ejemplo, el *objeto de comunicación serial*, en el cual se encuentra mapeado el registro de salida del puerto serial del módulo SitePlayer, necesario para la comunicación con el sistema microcontrolado.

- *$InitialIP "192.168.0.1".* Establece la dirección IP inicial para el módulo SitePlayer. Si el módulo se conecta a un servidor DCHP, esta dirección puede ser sustituida por que resuelva dicho servidor.

- *$SiteFile "C:\Mis Documentos\Carlos\TesisSP\Sys1.spb".* Define el nombre y ubicación del *archivo binario de imagen* que será generado al compilar este *archivo de definición.*

- *$SitePath "C:\Mis Documentos\Carlos\TesisSP\Sistema1".* Define la raíz del sistema de archivos para el servidor HTTP.

En la sección de definición de *objetos* se encuentra la declaración de todos los *objetos* utilizados para la interacción entre las páginas HTML y el microcontrolador central del iPLC. A continuación se presenta la lista de *objetos* declarados:

- ***PLC_Status***. *Objeto* tipo byte, utilizado para desplegar el byte de estado del iPLC.

- ***Trefresh***. *Objeto* tipo Word, utilizado para almacenar el parámetro del periodo de monitorización, ingresado por el usuario desde la página HTML correspondiente.

- ***a1 — a40***. *Objetos* Tipo Double, utilizados para almacenar las direcciones de las variables que el usuario desea monitorizar (40 como máximo).

- ***d1 — d40***. *Objetos* tipo Double, utilizados para almacenar los datos de todas las variables que se están monitorizando.

- ***n1 - n40***. *Objetos* tipo String de 7 caracteres, utilizados para almacenar los nombres de las variables que el usuario desea monitorizar.

- ***emt1 – emt15***. *Objetos* tipo Byte, utilizados para almacenar los códigos de identificación de todos los módulos de expansión conectados al bus.

En el apéndice B se presenta el *archivo de definición* del servidor HTTP, en él se puede consultar la sintaxis precisa para todas las configuraciones y definiciones. El texto presentado allí es el que fue compilado y cargado al módulo SitePlayer.

B. Entorno de programación

El entorno de programación consiste en una sola página HTML llamada ***Prog.htm***. Esta página permite que el usuario remoto pueda reprogramar el iPLC, y que además

pueda controlar el modo de operación del sistema. En esta página el usuario puede ingresar el código fuente del programa que desea cargar al iPLC, puede compilarlo y finalmente enviarlo a través de la Internet para que sea cargado en la memoria flash de la tarjeta madre del iPLC. Previo a la programación del iPLC éste debe ser borrado, para ello se dispone de un acceso en la página, que ejecuta la función de borrado. También se dispone de enlaces para la elección entre los modos de operación detenido, de ejecución, y el submodo de programación.

Para poder acceder al entorno de programación, el usuario remoto tiene que ejecutar el explorador de Internet y luego ingresar la dirección HTTP://192.168.0.1/Prog.htm, con esto el explorador de Internet cargaría la página de programación, servida desde el módulo SitePlayer incrustado en el iPLC.

La dirección IP predeterminada en el iPLC es 192.168.0.1, sin embargo, puede ser cambiada en el *archivo de definición*. Si el iPLC se instala en una planta industrial, entonces se requeriría cambiar esa dirección IP y sustituirla por alguna dirección disponible en la red de la planta. Esta dirección IP puede ser privada, por lo que el acceso al iPLC estaría limitado dentro de la misma red. En cambio, si la dirección IP disponible es pública, el iPLC puede ser "navegado" desde cualquier parte del mundo.

1. Interacción con el usuario. La interacción con el usuario se lleva a cabo por medio de los elementos disponibles en la página HTML. La página Prog.htm posee tres tipos de elementos: enlace, botón y área de texto.

a. El enlace "activar el modo de ejecución". Al presionar sobre este enlace se activa el modo de ejecución del iPLC, efectuando la notificación visual tanto en los LED's del sistema remoto, como en los LED's gráficos del entorno de monitorización.

b. El enlace "activar el modo detenido". Al presionar sobre este enlace se activa el modo detenido del iPLC, efectuando la notificación al igual que para el modo de ejecución.

c. El enlace "activar el submodo de programación". Al presionar sobre este enlace se activa el submodo de programación del sistema. Es necesario activar este modo siempre que se vaya a cargar un nuevo programa de usuario al iPLC, debe hacerse ANTES de introducir el código fuente del programa, pues al activar el modo, la página HTML es recargada, borrándose el contenido del área de texto.

d. El enlace "borrar el programa de usuario cargado al iPLC". Este enlace permite borrar el programa de usuario que se encuentre almacenado en la memoria flash de la CPU central.

e. El área de texto. Este elemento es utilizado para que el usuario ingrese el código fuente del programa que desea cargar al iPLC. El programa debe estar en el lenguaje tipo STL desarrollado para el iPLC, con una instrucción de programa por cada línea de texto.

f. El botón "compilar el código fuente". Con este botón se activa el compilador integrado en el servidor HTTP. Al presionarlo se efectúa la compilación de todo el código fuente que haya sido ingresado en el área de texto. De encontrarse errores de sintaxis o errores en los parámetros de las instrucciones, éstos son notificados por medio de un cuadro de diálogo. De no encontrarse errores, aparece un cuadro de diálogo notificando la compilación exitosa, habilitando al sistema para que se proceda con la programación del iPLC. Al compilar el código fuente se generan los códigos de programa en el lenguaje del iPLC (parte del firmware del sistema), estos códigos son los que se almacenan en la memoria flash durante la reprogramación del sistema.

g. El botón "cargar el programa en la memoria del iPLC". Este botón permite programar el iPLC con el código fuente previamente compilado. La función de este botón estará habilitada únicamente luego de haberse realizado una compilación exitosa. Todos los códigos de programa de usuario generados con el compilador son enviados a través de la conexión Ethernet hacia el microcontrolador 89C51RD2, el cual a su vez, los retransmite hacia el microcontrolador central PIC16F877, para que este último

los transmita hacia el IC flash externo, donde quedan almacenados de forma permanente para su futura ejecución.

Los tres enlaces para la selección de los modos de operación ejecutan una función que escribe el código correspondiente con el modo seleccionado, en los dos bits más significativos del puerto de entrada / salida del módulo SitePlayer, con lo cual efectivamente se provoca el cambio del modo de operación en el iPLC (ver la sección E del capítulo V).

2. El compilador. La parte más complicada del entorno de programación es el compilador, que hace posible que el código fuente del programa de usuario sea transformado en los códigos de programa compatibles con el lenguaje interno del iPLC, almacenado como parte del firmware del microcontrolador central en la tarjeta madre. El compilador fue desarrollado en el lenguaje JavaScript, introducido en el código HTML de la página Prog.htm. De esta forma, el compilador se encuentra dentro del propio iPLC, específicamente en la memoria flash del microcontrolador 89C51RD2 integrado con el módulo SitePlayer.

a. Análisis del texto. La página Prog.htm ofrece un área de texto para que el usuario ingrese el código fuente del programa que desea cargar al iPLC. Cuando el programa ha sido ingresado, y se presiona el botón de compilación, la primera tarea realizada por el compilador interno es el *análisis del texto*. Esta tarea consiste en analizar línea por línea todo el código fuente, detectar y extraer todos los elementos de las instrucciones que se hayan ingresado. Con esta primera fase de detección y extracción, el compilador ofrece todos los componentes clave en cada una de las instrucciones, básicamente se trata de la separación de la instrucción y todos los argumentos que ésta posea. Teniendo todos los elementos detectados y separados puede procederse con el procesamiento de cada uno de ellos. La tarea realizada en esta fase es similar a lo que en teoría de compiladores se conoce con el nombre *parsing*. Ejemplo: Supóngase que en el código fuente ingresado por el usuario se encuentra la línea de programa " MovB

0x7F,VB125 ", entonces, durante la fase de análisis del texto, el compilador realizaría la siguiente detección y extracción de elementos:

- MovB: Instrucción del lenguaje.
- 0x7F: Argumento (tipo constante).
- VB: Área de memoria de variables de usuario (memoria V), tipo Byte.
- 125: Número de elemento dentro de la región de memoria V.

b. Parametrización. En esta etapa de la compilación se ejecuta una función en la cual se lleva a cabo la *parametrización* de cada instrucción de acuerdo a los elementos extraídos previamente. Esta parametrización consiste en la asignación de tres argumentos para cada una de las instrucciones detectadas. Con este procedimiento de parametrización, al relacionar cada instrucción con sus tres argumentos, se logra crear una caracterización única para cada una de las instrucciones en el lenguaje del iPLC, utilizadas en el código fuente que ingresó el usuario.

1) Parámetro *a.* Consiste en un código único asignado a cada elemento de instrucción, cuyo valor pertenece al rango [1 , 106]. Este código es utilizado como un identificador para el *microprocesador virtual* del iPLC. Durante la fase de ejecución, cuando se lee el byte que posee el código de la instrucción, almacenado en la memoria flash, el *microprocesador virtual* ejecuta la instrucción cuyo código sea correspondiente con el parámetro *a.* En otras palabras, cada rutina de instrucción en el firmware del iPLC posee un código asociado: el Parámetro *a.*

2) Parámetro *b.* Este parámetro indica la cantidad de argumentos que posee la instrucción. Este valor es útil para una función posterior que requiere el número de elementos del tipo *argumento de instrucción* que debe procesar.

3) Parámetro *c*. Indica el tipo de datos compatibles con los argumentos de la función. Este parámetro hace posible determinar si los argumentos de la instrucción son válidos, y de ser así, para procesarlos adecuadamente, según su tipo, para generar los códigos de programa que correspondan. Este parámetro indica si la instrucción acepta datos tipo Bit ($c = 0$), Byte ($c = 1$), Word ($c = 2$), Double Word ($c = 3$) o Reales ($c = 4$).

El tipo de dato siempre es determinado a partir del primer argumento de la función, el tipo de los demás argumentos queda implícito.

Los parámetros son utilizados como argumentos de una función de procesamiento denominada *función X*.

Ejemplo: Considere que el código fuente ingresado por el usuario contiene la línea de texto "Div 5.29e3,VD2,VD5". Luego de la función de parametrización, se generaría la función X(81,3,4), donde el valor '81' indica el código que identifica de forma única a la instrucción "Div", el número '3' indica que la instrucción posee 3 argumentos, y el número '4' indica que el primer argumento es de tipo real. Los otros dos argumentos deben ser variables tipo real.

c. Detección de constantes. En esta función se determina si el argumento de una instrucción es una constante o una variable. De ser una constante, se determina si se trata en una constante entera o real.

Dependiendo del tipo de constante detectado, el compilador produce lo siguiente:

- Si detecta una constante que empieza con "0", por ejemplo "0345", se asume que se trata de una constante entera en código octal. De tal forma que se realiza la conversión octal – decimal, y se generan cuatro bytes con codificación binaria, conteniendo el valor decimal de la constante introducida.
- Si se detecta una constante que empieza con "0x", por ejemplo "0x30", se asume que se trata de una constante entera en código hexadecimal. De tal forma que se

realiza la conversión hexadecimal – decimal, y se generan cuatro bytes con codificación binaria, conteniendo el valor decimal de la constante introducida.

- Si se detecta una constante entera, pero que no cumple con ninguno de los casos anteriores, se asume que se encuentra en formato decimal, por lo que se generan directamente los cuatro bytes.

Aunque el compilador puede generar enteros de 32 bits, debe tenerse en cuenta que cualquier entero debe poder ser representado con 24 bits. Este límite en la longitud del entero está definido por el coprocesador matemático.

En el caso de las constantes reales, se utiliza una función especial, para generar los 32 bits en formato IEEE754 a partir del valor ingresado por el usuario, por ejemplo "5.29e3". Es indispensable realizar esta transformación hacia el formato IEEE754 debido a que este es el formato reconocido por el coprocesador matemático, aunque luego en el coprocesador mismo se realiza una segunda conversión, hacia el formato interno para la realización de las funciones matemáticas (formato propio del coprocesador matemático).

d. Generación de direcciones. El siguiente paso en el proceso de compilación es la generación de las direcciones de todos los argumentos en las instrucciones del código fuente. Las direcciones que aquí se generan corresponden con las localidades de memoria RAM de las variables a las cuales hacen referencia los argumentos en las instrucciones del código que ingresa el usuario. Para poder generar las direcciones a partir del texto de los argumentos en el código fuente, se utilizan algunos resultados de la función de *análisis del texto*, que previamente ha sido ejecutada. Precisamente, el *análisis del texto* provee tres argumentos sobre los cuales se basa la *generación de direcciones*, los argumentos son el *área de memoria*, el *tipo de dato*, y el *número de elemento*.

1) *Área de memoria.* Indica la región de memoria donde se encuentra mapeada la variable a la cual se hace referencia en el argumento de la instrucción que se

está procesando (del cual se desea generar una dirección). Con esta información se realiza un desplazamiento inicial a través de la memoria RAM de la tarjeta madre, dependiendo de la región indicada (consultar la sección C del capítulo IV).

2) *Número de elemento y tipo de dato.* Se utilizan conjuntamente para realizar un segundo desplazamiento en la memoria RAM externa del iPLC. Este desplazamiento está definido por el *número del elemento* que se desea acceder (en la región previamente especificada), multiplicado por la cantidad de bytes correspondientes con el *tipo del dato* (byte=1, Word=2, Double=Real=4).

Es así, que después de realizar los dos desplazamientos en el mapa de la memoria RAM externa del iPLC, esta fase de compilación provee la dirección RAM de cada argumento que procesa. Luego de procesar todos los argumentos en las instrucciones del código fuente ingresado por el usuario, el proceso de compilación llega a su fin.

e. El búfer de salida. Las fases de compilación presentadas en los incisos anteriores son ejecutadas para la compilación de todo el programa contenido en el área de texto de la página Prog.htm. Estas fases son ejecutadas dentro de un ciclo. Por cada ejecución de un ciclo se ejecutan todas las fases y se procesa una sola instrucción de programa en el código fuente, es decir, se compila una sola instrucción por cada ciclo de fases. Cada fase genera sus propios códigos de programa (que serán almacenados en la memoria flash del iPLC). En cada ciclo ejecutado se generan todos los códigos de programa para una sola instrucción (una sola línea de código fuente). Según se van generando los códigos de programa, son almacenados en el *búfer de salida*, que consiste en un arreglo dinámico de registros (bytes). De esta forma, al inicio del proceso de compilación el *búfer de salida* tiene un tamaño de 0 bytes, pero cuando dicho proceso concluye, el tamaño de este búfer ha crecido hasta N bytes, donde N es la cantidad de códigos de programa que codifican y representan de forma única al código fuente del programa de usuario ingresado en el área de texto de la página Prog.htm. El búfer completo es posteriormente transmitido hacia el iPLC, con el submodo de programación,

para que sea almacenado en la memoria flash de la tarjeta madre, y ejecutado durante la fase de ejecución del sistema (en el modo de ejecución).

f. Detección de errores en el código fuente. Se desarrollo una función para el despliegue de errores de compilación. Esta función se encarga de realizar una notificación por medio de un cuadro de diálogo, cuando el código fuente ingresado por el usuario contiene algún error de *sintaxis*, de *rango* o de *tipo*. Los errores de *sintaxis* se refieren por ejemplo, a la utilización de palabras no definidas en el lenguaje STL del iPLC, a errores en la cantidad de parámetros de una instrucción, a etiquetas o subrutinas no definidas, etc. Los errores de *rango* son detectados cuando algún argumento está fuera del rango físico o lógico del iPLC, por ejemplo cuando se desea acceder a una variable que sobrepasa los 32K bytes de memoria RAM instalada en la tarjeta madre (físico), o cuando se desea acceder a un número de elemento que ya no existe en la región de memoria especificada (se traslaparía con otra región, rango lógico). Los errores de *tipo* ocurren cuando el programa de usuario incluye algún argumento de tipo incompatible con la instrucción que lo utiliza.

Para la detección de los errores no se utiliza alguna función en particular, pues todas las funciones, a través de todas las fases, implementan técnicas para la detección de los errores que en cada una ellas puedan ocurrir.

g. Instrucciones de control de programa. El compilador reconoce instrucciones de salto (JMP) y de ejecución de subrutinas (CALLSBR). Para poder implementar la compilación de estas características se utilizan las palabras reservadas 'LABEL' y 'SBR'. Con la primera se definen etiquetas para las instrucciones de salto, mientras que con la segunda se definen subrutinas.

Durante la compilación de todas las instrucciones, cuando se detecta una etiqueta 'LABEL' o una etiqueta 'SBR', se almacena en un arreglo dinámico X la dirección del último código de programa generado, y simultáneamente se almacena en un arreglo dinámico Y la cadena de texto que define a la etiqueta o subrutina. De esta forma, al

final de la compilación, los arreglos *X* y *Y* funcionan como una pareja que posee todas las direcciones de las etiquetas y subrutinas creadas (en la memoria de programa), y los nombres asociados a éstas. Además, también durante la compilación, cuando se detectan las instrucciones 'JMP' y 'CALLSBR', se almacena una "solicitud" de salto en un arreglo dinámico *Z*. Esta "solicitud" consiste en el código de la instrucción (JMP o CALLSBR) y el nombre de la etiqueta a la cual se desea trasladar el control del programa. Además se almacena en un arreglo dinámico *W* la dirección donde se detectó la solicitud de salto (en la memoria de programa). Al final de la compilación, el arreglo *Z* posee todas las solicitudes de "salto" del programa de usuario, y el arreglo *W* posee las direcciones en donde éstas fueron requeridas. Así, al final de la compilación, se realiza una función de apareamiento entre todos los arreglos, apareando las direcciones de memoria de programa en donde existen solicitudes de salto, con las direcciones en la memoria de programa donde es encuentran las rutinas a las cuales se desea saltar (definidas por las etiquetas correspondientes). De esta forma se llenan todas las posiciones del búfer de salida que habían quedado vacías debido a la utilización de instrucciones de control de programa. Esas posiciones estaban vacías debido a que durante la compilación, hacían referencia a segmentos de programa que aún no habían sido definidos o que aún no habían sido apareados entre la posición del segmento a ejecutar, y la posición de programa desde donde se debe saltar hacia él.

3. Transmisión de información. La transmisión de la información se apoya en el *archivo de interfaz*. Como se mencionó en el capítulo IV, en la sección del módulo SitePlayer (sección F, inciso 3), las características tan poderosas del *archivo de interfaz* hacen que en su contenido únicamente se requiera indicar un redireccionamiento para el explorador de Internet. En otras palabras, únicamente se requiere indicar el nombre de la página HTML con la cual responderá el servidor HTTP después de efectuar la transmisión de los datos.

Luego de enviar datos desde la página Prog.htm, no se desea realizar un redireccionamiento hacia otra página del servidor HTTP. Sin embargo, cuando se envían

datos desde un explorador de Internet, el redireccionamiento es un requisito del funcionamiento intrínseco del servidor HTTP. Debido a ello, se realizó un redireccionamiento hacia la misma página Prog.htm. Para la transmisión de los datos desde el explorador de Internet hacia el servidor HTTP se utilizaron dos tipos de procedimientos: *Enlace directo* y *formulario.*

a. El archivo de interfaz implementado. A continuación se presenta el texto completo con la sintaxis del *archivo de interfaz* utilizado en el iPLC.

```
HTTP/1.0 302 Found
Location: /Prog.htm
```

Las dos líneas de texto que se muestran arriba conforman al *archivo de interfaz.* Ese archivo se encuentra situado en la raíz de la estructura de archivos del servidor HTTP (ver la sección A de este capítulo), almacenado como un archivo de texto, con la extensión ".SPI". El nombre del archivo es "Progspi.spi". Como puede verse, el redireccionamiento es hacia la misma página Prog.htm.

b. Los enlaces directos implementados. Las funciones para el cambio del modo de operación, así como la función para Borrar el iPLC, fueron implementadas utilizando enlaces directos, con el *archivo de interfaz* funcionando como una aplicación CGI. A continuación se presenta el código HTML de los enlaces directos creados en la página Prog.htm:

```
<A href="file:///C:/Mis%20documentos/Carlos/TesisSP/Sistema1/Progspi.spi ? io6=1 &; io5=0&; com=1"> <H3>Borrar el Programa de Usuario cargado al iPLC</H3></A><BR>

<A href="file:///C:/Mis%20documentos/Carlos/TesisSP/Sistema1/Progspi.spi ? io6=1& amp; io5=1"> <H3>Activar el Submodo de Programación</H3></A>

<A href="file:///C:/Mis%20documentos/Carlos/TesisSP/Sistema1/Progspi.spi ? io6=0&amp; io5=0">  <H3>Activar el Modo de Ejecución</H3></A>
```

<A href="file:///C:/Mis%20documentos/Carlos/TesisSP/Sistema1/**Progspi.spi** ? **io6**=0& **io5**=1"> <H3>Activar el Modo Detenido</H3></A>

De acuerdo a la implementación de los enlaces directos se debe destacar lo siguiente:

- Todos los enlaces directos usan al archivo **Progspi.spi** como si se tratara de una aplicación CGI.
- Todos los enlaces directos usan los parámetros **io6**, **io5** y **com**. Estos parámetros están definidos como *objetos* en el archivo "pcadefv2.inc" que se incluyó en el *archivo de definición*.

Los *objetos* **io6** e **io5** se encuentran mapeados en la dirección de los bits IO6 e IO5 respectivamente, del puerto de entrada / salida del módulo SitePlayer. Con cada uno de los enlaces se envían valores a estos dos bits, de tal forma que en el puerto de E / S del módulo SitePlayer aparezca el código para el modo de operación correspondiente, que el usuario a ordenado activar (ver sección E del capítulo V).

El *objeto* **com** está mapeado en la dirección del búfer de salida del puerto serial asíncrono del microcontrolador 89C51RD2 integrado en el módulo SitePlayer. Así, todos los bytes que se escriben a este *objeto*, son transmitidos a través del canal asíncrono del módulo SitePlayer, que está conectado con el canal asíncrono del iPLC. Cuando el sistema del iPLC detecta el código io6=1 e io5=0, y estando en él recibe un byte (cualquiera) en su canal asíncrono, procede con el borrado de la memoria flash externa, borrando así el último programa de usuario cargado al iPLC.

c. El formulario de transmisión. El procedimiento de formulario fue utilizado para la transmisión del búfer de salida. Se implementó un formulario con un único elemento que consiste en un campo de tipo *oculto* (hidden), en el cual se escriben todos los valores del búfer de salida, uno por uno. Como se explicó en una sección previa, todos los códigos de programa, listos para ser cargados en la memoria flash de la tarjeta madre del iPLC, se encuentran almacenados en un arreglo dinámico de registros

(bytes). Cada posición de este arreglo contiene uno de los códigos de programa que se almacenará en una localidad de la memoria flash. Cuando el sistema del iPLC se encuentra en el submodo de programación habilita su canal asíncrono de tal forma que todo lo que en él recibe es almacenado en la memoria flash externa. De esa forma, lo que se requiere del entorno de programación es que envíe todos los códigos de programa, un byte tras otro byte, estando en el submodo de programación. Para implementar esa funcionalidad se utilizó un ciclo de transmisión, en el cual se hace un "barrido" de todas las posiciones del búfer de salida, y se llama a una función de transmisión. La función de transmisión escribe el valor de una posición del búfer de salida, en el elemento *oculto* del formulario.

El elemento *oculto* del formulario es un *objeto* mapeado en la dirección del puerto serial asíncrono del microcontrolador 89C51RD2. Además, cada vez que se ejecuta la función de transmisión se realiza el envío del formulario. Con esto, cada vez que se ejecuta la función de transmisión se envía un byte de código de programa hacia el canal serial del microcontrolador central del iPLC.

El ciclo de transmisión está implementado como función del elemento tipo botón que posee el mensaje "Cargar el programa en la memoria del iPLC". Por lo tanto, cuando el usuario activa el enlace para el submodo de programación, ingresa el código fuente y lo compila, y finalmente presiona el botón de programación, todos los códigos de programa son almacenados en la memoria flash del iPLC. El resultado final es que el iPLC queda reprogramado de forma remota.

Por ejemplo, supóngase que el código fuente ingresado por el usuario, es tal que al ser compilado genera 100 códigos de programa. Cuando se presione el botón de programación se efectuarán 100 envíos del formulario. Cada envío posee un solo parámetro, el *objeto* **com**. Así, se envían los 100 códigos de programa hacia la memoria flash del iPLC. A continuación se presenta el código HTML del formulario implementado para la transmisión del búfer de salida.

<FORM name=out_form action=**Progspi.spi** method=get> <INPUT type=hidden name=**com**> </FORM>

Nótese la técnica de utilización del *archivo de interfaz* 'Progspi.spi'. Además, nótese que el único elemento del formulario posee el nombre 'com' con lo cual se está haciendo referencia al *objeto* **com**, definido en el *archivo de definición*, y mapeado en la dirección del puerto serial asíncrono del microcontrolador 89C51RD2 integrado en el módulo SitePlayer (en el apéndice B encontrará el listado del *archivo de definición* y el archivo incluido 'pcadefv2.inc').

C. Entorno de monitorización

El entorno de monitorización consiste de dos páginas HTML: ***Mon1.htm*** y ***Mods.htm***. La página Mon1.htm es la principal, pues en ella se puede realizar la monitorización de todas las variables del iPLC. La página Mods.htm también ofrece un servicio de monitorización, pero se limita a ser una página informativa acerca del bus de expansión, mostrando el tipo de los módulos que se detectaron conectados al bus durante la inicialización del sistema.

1. La página de monitorización "Mon1.htm". Esta página permite la monitorización de TODAS las variables del sistema, es decir, permite la visualización del contenido de los 32 KB de memoria RAM instalada en la tarjeta madre. De forma simultánea se puede monitorizar hasta un máximo de 40 variables, de cualquier tipo (Byte, Word, Double, Real). Además, esta página posee un campo de texto en el que el usuario debe ingresar el periodo de monitorización (en segundos), es decir, el tiempo que transcurre entre un despliegue de datos y la próxima actualización. La página ofrece 80 campos de texto, apareados, en los que el campo A de cada pareja es utilizado para que el usuario ingrese la localidad de memoria a monitorizar, mientras que en el campo B el sistema escribirá el valor contenido en dicha localidad al momento de realizarse la siguiente actualización de los datos en la página HTML. Además, esta página incluye

cinco LED's gráficos, uno de ellos es utilizado para la visualización del modo de operación del sistema, se enciende de color rojo para los modos detenido y de programación, y de color verde para el modo de ejecución. Los otros cuatro LED's son utilizados para la visualización de los códigos de error, si no se ha presentado algún error, los LED's permanecen apagados, de lo contrario se encienden de color amarillo. Por último, la página ofrece dos botones para la activación / desactivación de la monitorización respectivamente.

a. Visión general de la operación de la página Mon1.htm. Para monitorizar al iPLC, el usuario debe ingresar todos los nombres de las variables que desee monitorizar y el periodo de monitorización (en segundos) en el campo de texto correspondiente. Ya que se ha ingresado esa información, se presiona el botón "Monitorizar al iPLC", en ese momento el sistema de monitorización realiza la transmisión de todas las solicitudes y el periodo de monitorización hacia el servidor HTTP. Esta información queda en el módulo SitePlayer, no es retransmitida directamente hacia el microcontrolador central del iPLC. En cambio, durante la fase de monitorización, el iPLC realiza la solicitud hacia el módulo SitePlayer para recibir la información de una sola variable a la vez (ver sección N del capítulo V), solicitando la dirección de la variable a monitorizar. Cuando el iPLC obtiene el valor de la variable de monitorización, envía el dato hacia el módulo SitePlayer, para que sea desplegado en la próxima actualización. El ciclo del iPLC continúa, de tal forma que con cada fase de monitorización se procesa una nueva variable, hasta llegar a la última, luego se reinicia el ciclo de monitorización. La actualización de los datos continúa hasta que se presione el botón "Detener la monitorización".

La página almacena todos los nombres de las variables de monitorización ingresados por el usuario. De tal forma que cuando se recarga la página, aparecen los nombres de todas las variables que se solicitaron en la última sesión de monitorización. Sin embargo, los valores de las variables no serán los correctos, sino hasta que se efectúe la primera actualización de los datos. Si se cierra la página sin haber detenido la monitorización,

luego, al recargar la página el ciclo de monitorización y actualización de datos continuará automáticamente, con el último periodo de monitorización ingresado.

b. Solicitudes de monitorización. Las solicitudes de monitorización se refieren a todas las variables ingresadas por el usuario, en los campos de texto correspondientes, y cuyos valores se desea monitorizar de forma periódica en la página Mon1.htm. Para que el sistema de monitorización tenga solicitudes, el usuario debe ingresar el nombre de las variables, indicando la región de memoria, el tipo de dato y el número de elemento dentro de esa región, por ejemplo, el nombre VB100 hace referencia al byte # 100 de la memoria V. Posteriormente cuando se presiona el botón "monitorizar al iPLC", el sistema de monitorización no envía los nombres de las variables hacia el servidor HTTP, sino que primero ejecuta una función casi idéntica a la que se implementa en la fase de generación de direcciones del compilador. Esta función es necesaria para obtener las direcciones RAM de todas las variables a monitorizar, pues cuando la fase de monitorización del iPLC requiera información acerca de las solicitudes de monitorización que debe atender, no "entenderá" nombres, sino direcciones en la memoria RAM externa de su tarjeta madre. La función generadora de direcciones en esta página de monitorización agrega una rutina especial utilizada para la monitorización de variables tipo real.

c. Monitorización de variables reales. Debido al formato numérico interno del coprocesador matemático, se presenta la necesidad de utilizar el formato IEEE754 para almacenar todas las variables reales en la memoria RAM de la tarjeta madre, de tal forma que todas las funciones matemáticas puedan asumir que los datos están en formato IEEE754, y no se realicen conversiones innecesarias o se generen resultados erróneos. El problema consiste en que si durante la compilación no se transforman todas las constantes al formato IEEE754, pueden presentarse situaciones en las cuales el coprocesador matemático intente realizar una función matemática con un argumento en formato entero y otro en formato IEEE754, por ejemplo. En cambio, si se garantiza que todas las constantes almacenadas en la memoria RAM están en el formato IEEE754, todas las funciones matemáticas simplemente ejecutan una conversión desde IEEE754

hacia el formato interno y posteriormente calculan la función, evitando así problemas de discrepancias entre el tipo de cada argumento.

Debido a que todas las constantes son codificadas en formato IEEE754, cuando se programa al iPLC se utiliza la denominación 'VD' para acceder a variables reales, pues el tipo 'D' es un 'Double Word' que posee 32 bits. Sin embargo, si se desea monitorizar una variable real y se utiliza la denominación 'VD', el sistema de Monitorización no desplegará el valor de la variable real de 32 bits, sino que desplegará un valor erróneo: el valor decimal de la representación binaria del número en formato IEEE754, que es una representación binaria codificada (ver la sección D en el capítulo IV). Para solucionar este problema se implementó una función de conversión desde IEEE754 hacia el valor real decimal correspondiente, únicamente con fines de monitorización.

Las variables tipo real y las variables tipo double, utilizan ambas 4 bytes, de tal forma que se utiliza la denominación 'D' para indicar al sistema de monitorización que el valor a desplegar y actualizar en la página Mon1.htm se trata de una variable tipo Double. Con esto, el valor desplegado será precisamente el valor decimal de la representación binaria de 32 bits de la variable tipo double. En cambio, se utiliza la denominación 'R' para indicar al sistema de monitorización que la variable a monitorizar es de tipo Real. En este caso, el valor desplegado será el valor decimal de la representación binaria **decodificada** desde el formato IEEE754. La distinción entre variables tipo Real y variables tipo Double es una diferencia importante entre el entorno de programación y el entorno de monitorización.

d. Los *objetos* del sistema de monitorización. En el sistema de programación se utilizan tan solo 3 *objetos*: 'io6', 'io5' y 'com', en cambio en el sistema de monitorización se utilizan 137 *objetos*. La página Mon1.htm contiene 122 *objetos*, mientras que los 15 restantes se encuentran en la página Mods.htm.

1) El *objeto PLC_Status*. Cada vez que se recarga la página Mon1.htm, el estado de los LED's gráficos es actualizado según el valor del *objeto PLC_Status*. En

el sistema de archivos del servidor HTTP se incluyeron cuatro archivos gráficos, con los nombres 'E0.gif', 'E1.gif', 'Mode0.gif' y 'Mode1.gif', todos ellos almacenados en la raíz del servidor HTTP. Los gráficos E0 y E1 corresponden a un mismo LED, pero en estado encendido y apagado respectivamente. Los gráficos Mode0 y Mode1 también corresponden a un mismo LED, diferente al anterior, y también es estado encendido / apagado respectivamente. La técnica utilizada es la siguiente:

```
<IMG src=E^PLC_Status'0.gif align=left hspace=20 width=30 height=20>
```

Esta línea de código HTML corresponde con la creación del elemento gráfico para el LED del bit menos significativo en el Byte de Estado del iPLC. El funcionamiento de esta técnica es como sigue: El símbolo " ' " es interpretado por el servidor HTTP del módulo SitePlayer como un *modificador de objeto*, que cumple la función de extraer el valor de un bit de un *objeto*. El *objeto* se antepone al símbolo, y éste se antepone al número que indica la posición del bit dentro del byte del *objeto*. De esta forma, el texto "^PLC_Status'0" retorna el valor del bit menos significativo en el byte del *objeto PLC_Status*. Con esto, cuando el servidor HTTP envía el código HTML hacia el explorador de Internet, sustituye el texto "E^PLC_Status'0" con "E0" si el bit menos significativo del byte de estado posee un estado bajo, o con "E1" si el estado del mismo bit es alto. Así, el archivo gráfico cambiará entre "E0.gif" y "E1.gif" dependiendo del estado del bit menos significativo en el byte de estado del iPLC, con lo cual se logra el efecto de mostrar un LED encendido / apagado que corresponde con el LED real que se encuentra instalado en la tarjeta madre del iPLC.

2) El *objeto Trefresh.* Cuando el usuario presiona el botón "Monitorizar al iPLC", el sistema de monitorización escribe hacia 41 *objetos*. Primero escribe al *objeto* **Trefresh**, en el cual almacena el valor para el periodo de monitorización, que es utilizado como variable en el encabezado del código HTML para la propiedad que "refresca" el contenido de la página HTML, tal como se muestra a continuación:

```
<meta HTTP-equiv="refresh" content=^Trefresh>
```

Con esta línea de código HTML se provoca que la página de monitorización Mon1.htm sea recargada periódicamente, con periodo T_M, donde T_M es el valor ingresado por el usuario en el campo de texto correspondiente. Nótese la técnica de utilización del *objeto* **Trefresh**: Cada vez que el servidor HTTP envía la página hacia el explorador de Internet, en lugar de enviar el texto "^Trefresh" envía el contenido de la localidad de memoria RAM en la cual fue definido este *objeto.*

3) Los *objetos n1* hasta *n40* y *a1* hasta *a40.* Luego de escribir al *objeto* **Trefresh**, el sistema de monitorización escribe en los elementos tipo *oculto* de un formulario. En total son 40 elementos, a los cuales se les asignaron los nombres de los 40 *objetos n1* a *n40.* En estos *objetos* se escriben las cadenas de texto correspondientes con los nombres de las variables a monitorizar, ingresados por el usuario. Posteriormente el sistema de monitorización envía el formulario hacia el servidor HTTP, actualizando el valor de los *objetos n1* a *n40*, con lo cual quedan almacenados todos los nombres de las variables de monitorización. Estos nombres serán reutilizados cada vez que se recarga la página Mon1.htm, ya que de no existir los *objetos n1* a *n40*, al recargar la página se borrarían los nombres de las variables, con lo cual se cancelarían todas las solicitudes de monitorización. Para evitar eso, cada vez que se recarga la página se leen los *objetos n1* a *n40*, y se escribe el contenido de éstos en los campos de texto reservados para los nombres de variables, donde el usuario los ingresó la primera vez, antes de activar la monitorización.

Finalmente, el sistema de monitorización ejecuta la función de generación de direcciones modificada, para obtener las direcciones RAM de todas las variables a monitorizar. Estas direcciones son escritas en los elementos de un segundo formulario, el cual posee 40 elementos, cuyos nombres van desde a1 hasta a40, que corresponden con otros 40 *objetos* del servidor HTTP. Luego el sistema de monitorización envía el formulario hacia el servidor HTTP, de tal forma que en estos *objetos*, desde *a1* hasta *a40* quedan almacenadas las direcciones de todas las variables a monitorizar, que serán requeridas desde el iPLC para su procesamiento. En este punto finaliza la tarea de envío de información desde la página Mon1.htm.

4) Los *objetos d1* hasta *d40*. El procedimiento que continúa para ofrecer la monitorización de las variables tiene lugar en el iPLC, el cual adquiere una a una, desde el módulo SitePlayer, todas las direcciones de las variables a monitorizar (*Objetos a1* hasta *a40*) , con las cuales accede a la memoria RAM y extrae los valores correspondientes, que posteriormente envía hacia el servidor HTTP, escribiéndolos en los últimos 40 *objetos* definidos, *d1* a *d40*. Cada vez que la página Mon1.htm es recargada, llena los campos de texto para el valor monitorizado, con el valor que contengan los *objetos d1* a *d40*, actualizando así el contenido de la página con los nuevos valores para las variables monitorizadas, y finalizando el proceso de monitorización. Luego, cuando transcurre el periodo T_M, todo el proceso se reinicia.

2. La página "Mods.htm". Esta es la segunda página del entorno de monitorización del servidor HTTP. En ella se puede leer la información adquirida acerca de los módulos detectados en el bus de expansión durante la fase de inicialización del sistema.

La información ofrecida por esta página es de gran utilidad ya que se debe conocer la cantidad y ubicación de las entradas y salidas digitales / analógicas y / o especiales que se posean para ser tomadas en cuenta al escribir el programa de usuario. Esta página es bastante simple, pues no ofrece interacción con el usuario, ni actualización periódica.

a. Los *objetos emt1* a *emt15*. Esta página utiliza los quince últimos *objetos* incluidos como parte del servidor HTTP en el *archivo de definición*. Estos *objetos* consisten en bytes que almacenan el código de identificación de los módulos de expansión detectados durante la fase de inicialización del iPLC. Cuando los módulos de expansión envían sus códigos de identificación hacia el microcontrolador central, este los recibe, y retransmite hacia el módulo SitePlayer, donde son almacenados en la memoria RAM, como *objetos*.

En la página Mods.htm se dispone de una tabla con 15 filas, cada fila corresponde a una dirección en el bus de expansión. Cada fila posee dos columnas, en la primera se

despliega el número de módulo / dirección en el bus de expansión, y en la segunda se despliega el tipo de módulo detectado, en campos de texto correspondientes a un formulario *X*. También se creó un formulario *Y*, que contiene 15 elementos tipo *oculto*, que poseen los nombres 'em1' a 'em15', de tal forma que al cargar la página Mods.htm, los elementos del formulario *Y* adquieren el valor de los códigos de identificación de todos los módulos de expansión detectados. Se utilizó una función ejecutada al cargar la página que se encarga de evaluar los códigos de identificación y asociar una cadena de texto que define las características de cada uno de los módulos, según el código correspondiente. Esas cadenas de texto son escritas en los campos de texto del formulario *X* de tal forma que las descripciones acerca de los módulos de expansión aparecen en la segunda columna de la tabla en la página HTML.

b. Códigos de identificación y mensajes. En el código JavaScript de la página Mods.htm se posee una tabla para relacionar códigos de identificación con cadenas de texto descriptivas acerca de los módulos de expansión. Cada tipo de módulo de expansión existente debe ser incluido en esta tabla, proporcionando el código de identificación y la descripción del módulo.

El sistema del iPLC envía un código especial cuando determina que una dirección está libre, de tal forma que los mensajes de la segunda columna pueden presentar la descripción del tipo de módulo detectado, un mensaje de dirección libre, o un mensaje de módulo no reconocido cuando se recibe un código diferente al especial, pero que no se encuentra emparejado con algún texto descriptivo.

VII. LENGUAJE DE PROGRAMACIÓN DEL iPLC

El lenguaje de programación desarrollado para el iPLC es prácticamente igual al lenguaje STL (Statement List) de la empresa alemana Siemens. El lenguaje está desarrollado con funciones JavaScript, e incrustado en el código de la página del compilador del iPLC, Prog.htm, que constituye al entorno de programación.

En total se implementaron 106 instrucciones, con las cuales se puede utilizar toda la capacidad del hardware instalado en el iPLC, y se obtiene casi toda la funcionalidad de los sistemas S7-200, a excepción de las comunicaciones en redes.

En las secciones posteriores se presentan las características distintivas y fundamentales del lenguaje del iPLC. Se asume que el lector posee conocimientos de programación de PLC's en lenguaje STL.

A. Escribir programas para el iPLC

1. Estructura y características de un programa para el iPLC. Los programas para el iPLC poseen las siguientes características fundamentales:

- La primera línea puede iniciar con cualquier instrucción de programa. No hay restricciones para el inicio del programa.
- Se debe ingresar una sola instrucción de programa por cada línea de código fuente.
- Se pueden utilizar mayúsculas y minúsculas indistintamente.
- Todas las líneas del código fuente deben iniciar en la primera columna del área de texto.
- En cada línea debe existir un espacio en blanco de separación entre la instrucción y su primer parámetro.

- Si la instrucción posee varios parámetros, éstos deben ir separados entre sí por una coma. El último parámetro no lleva coma ni espacio en blanco a su derecha.
- Puede existir un programa principal y diversas subrutinas.
- El programa principal debe terminar con la instrucción "Mend".
- Todas las subrutinas deben ingresarse después de la finalización del programa principal, en la línea siguiente a la que contiene la instrucción "Mend".
- Las subrutinas deben iniciar con una línea que posea el texto "SBR", seguido por una cadena de texto que define el nombre de la subrutina. El tamaño de la cadena de texto NO está limitado por programa. Debe existir un espacio en blanco entre el nombre y el texto "SBR".
- Las subrutinas deben finalizar con la instrucción "RET", escrita en una línea adicional, después de la última instrucción de la subrutina.
- El control del programa puede ser transferido a cualquier localidad por medio de la instrucción "JMP", seguida de la cadena de texto que define el nombre de la etiqueta hacia la cual se trasladará el control del programa.
- Las etiquetas son definidas utilizando el texto "LABEL", seguido de la cadena de texto que define el nombre de la etiqueta. Debe existir un espacio en blanco entre el nombre de la etiqueta y el texto "LABEL". La definición de una etiqueta ocupa una línea del código fuente.

2. Tipos de memoria del iPLC. Como todo PLC, el iPLC posee diversos tipos de memoria, organizados y mapeados en la memoria RAM del sistema según la tabla 4.1. Los tipos de memoria son utilizados para la organización de todos los datos y dispositivos funcionales dentro del iPLC. A continuación se presenta la lista de identificadores para cada región de memoria en el iPLC:

- Todas las variables del programa de usuario pertenecen a la región de memoria *V*.
- Todas las entradas digitales pertenecen a la región de memoria *DI*.
- Todas las salidas digitales pertenecen a la región de memoria *DQ*.
- Todas las entradas analógicas pertenecen a la región de memoria *AI*.
- Todas las salidas analógicas pertenecen a la región de memoria *AQ*.

- Todas las marcas especiales del sistema pertenecen a la región de memoria *SM*.
- Todos los contadores del sistema pertenecen a la región de memoria *C*.
- Todos los temporizadores del sistema pertenecen a la región de memoria *T*.
- Todos los registros del reloj de tiempo real pertenecen a la región de memoria *R*.

3. Tipos de datos. Casi todas las localidades de memoria pueden ser accedidas como variables tipo Byte (B), Word (W), Double Word (D) o Real (R). Las excepciones son los registros de marcas especiales y los registros del reloj de tiempo real, los cuales pueden ser accedidos únicamente como variables tipo byte. Para seleccionar el tipo de dato a acceder basta con colocar la letra correspondiente después de la letra que identifica a la región de memoria.

Ejemplo # 1. Para acceder a la variable de usuario tipo Byte # 10 se debe escribir el siguiente enunciado: VB10

Ejemplo # 2. Para acceder a la variable de usuario tipo Word #1460 se debe escribir el siguiente enunciado: VW1460

Ejemplo # 3 Para acceder al temporizador # 25 se debe escribir el siguiente enunciado: T25

En el lenguaje de programación, las variables tipo Double y las variables tipo Real son accedidas por medio del mismo identificador de tipo 'D', pues ambos tipos poseen cuatro bytes. Estos dos tipos de datos son utilizados indistintamente al programar. Sin embargo, debe tenerse cuidado de no utilizar funciones matemáticas con variables tipo Double, que no se encuentren en el formato IEEE754, de lo contrario los resultados serán erróneos. Todos los datos deben ser convertidos al formato IEEE754 previo a su utilización como argumentos en funciones matemáticas.

El tipo 'Word' también es indistintamente llamado 'Entero' por compatibilidad con los sistemas S7-200 de Siemens. Sin embargo, debe tenerse cuidado al utilizar este tipo

de datos con diversas instrucciones, pues algunas instrucciones lo utilizan como un dato de 16 bits (estrictamente tipo 'Word'), mientras que otras lo utilizan como un dato de 15 bits más un bit de signo (estrictamente tipo 'Entero'). El nombre de cada instrucción define la forma en que utilizan este tipo de dato.

4. Direccionamiento de bits. Algunas instrucciones requieren parámetros tipo bit, por lo que se debe realizar el direccionamiento correspondiente. Para direccionar un bit se debe hacer referencia al byte que pertenece. La sintaxis es: *byte.bit*

Ejemplo # 4. Para acceder al bit # 3 del registro imagen de las entradas digitales en el módulo de expansión # 6 se debe escribir el siguiente enunciado: DI6.3

5. Sintaxis de las instrucciones del iPLC. Todas las instrucciones en el lenguaje del iPLC cumplen con la siguiente sintaxis general:

Nombre de la instrucción{espacio en blanco}[parámetro1],[parámetro2],[parámetro3]

En el campo *nombre de la instrucción* se coloca el nombre de cualquiera de las 106 instrucciones implementadas en el lenguaje del iPLC. El espacio en blanco entre el nombre de la instrucción y sus parámetros es un requisito indispensable para el compilador.

Los parámetros opcionales 1, 2 y 3 son incluidos dependiendo de la instrucción que se utilice. Algunas instrucciones no utilizan parámetros, mientras que otras utilizan los tres. Todas las instrucciones que incluyen los tres parámetros, utilizan al último como parámetro de salida, es decir, ofrecen la dirección de la variable que debe almacenar la salida generada al ejecutar la instrucción.

Los parámetros pueden ser variables o constantes. La mayoría de instrucciones únicamente aceptan un parámetro constante, precisamente el *parámetro1*. Existen dos instrucciones que aceptan dos parámetros constantes: La instrucción "TONR" y la

instrucción "CTU". La primera instrucción es utilizada para activar un temporizador, de tal forma que el *parámetro1* debe ser una constante que indique el número del temporizador a activar, mientras que el *parámetro2* puede ser otra constante que indique la cantidad de tiempo que se desea temporizar. La segunda instrucción es utilizada con los contadores, de tal forma que el *parámetro1* es una constante que indica el número de contador, y el *parámetro2* puede ser otra constante que indica la cantidad de conteos que deben realizarse antes de activar el bit C correspondiente.

a. Parámetros variables. Todos los parámetros de tipo *variable* deben cumplir con la sintaxis siguiente:

{Área de memoria}{Tipo de dato a acceder}{Número de elemento a acceder}[.bit]

Todos los datos dentro del sistema del iPLC pueden ser representados con esa sintaxis general. Los primeros tres elementos son obligatorios, mientras que el cuarto elemento (.bit) es opcional dependiendo de la instrucción utilizada.

b. Parámetros constantes. Al escribir el código fuente del programa, los parámetros constantes pueden ser introducidos con cuatro formatos distintos: Decimal, científico, octal o hexadecimal. A continuación se presentan algunos ejemplos:

Constante decimal: MovB **240**,DQ2
Constante octal: MovB **0125**,DQ2
Constante hexadecimal: MovB **0x3F**,DQ2
Constante decimal (real): MovR **0.089**, VD24
Constante en notación científica: MovR **1.97e-2**, VD78

6. Las instrucciones especiales 'CTU' y 'TONR'. Estas dos instrucciones son utilizadas para el manejo de los contadores y los temporizadores del sistema, respectivamente. Poseen tres características especiales con relación a todas las demás en lo que se refiere a su utilización en el lenguaje de programación del iPLC:

- Son las únicas instrucciones que pueden utilizar más de un parámetro constante. De hecho pueden utilizar dos parámetros constantes, uno para el número de contador y otro para el valor predeterminado (VP).
- En el primer argumento de la instrucción (número de contador o número de temporizador) no se utiliza identificador de región de memoria, únicamente se escribe el número del elemento. La región de memoria está implícita en la instrucción: región C para la instrucción 'CTU', y región T para la instrucción 'TONR'.
- Esta característica no es precisamente de las instrucciones 'CTU' y 'TONR', pero está relacionada con ellas. Cuando cualquier instrucción que opere con bits, como por ejemplo la instrucción 'LD', desee acceder al bit C de un contador o al bit T de un temporizador, no se utiliza la notación *Byte.bit*, pues las instrucciones automáticamente consideran que el bit que se desea acceder es el más significativo de la palabra (2 bytes) asociada con el contador o temporizador, en donde se encuentran los bits C y T respectivamente.

B. El conjunto de instrucciones del iPLC.

A continuación se listan todas las instrucciones implementadas en el lenguaje del iPLC. Se incluye el nombre de la instrucción (como se le conoce en idioma inglés), la descripción de su función, y la sintaxis requerida para su utilización en la creación de programas de usuario.

1. **Load.** Carga un bit en la pila lógica.
   Sintaxis: LD *byte.bit*

2. **And.** Hace una operación AND de un bit, con el tope de la pila lógica, dejando el resultado en el tope de la pila lógica.
   Sintaxis: A *byte.bit*

3. **Or.** Hace una operación OR de un bit, con el tope de la pila lógica, dejando el resultado en el tope de la pila lógica.

   Sintaxis: O *byte.bit*

4. **Load Not.** Ejecuta una instrucción Load con el complemento de un bit.

   Sintaxis: LDN *byte.bit*

5. **And Not.** Ejecuta una instrucción AND con el complemento de un bit.

   Sintaxis: AN *byte.bit*

6. **Or Not.** Ejecuta una instrucción OR con el complemento de un bit.

   Sintaxis: ON *byte.bit*

7. **Not.** Complementa el valor de un bit.

   Sintaxis: Not *byte.bit*

8. **Edge Up.** Carga un uno en la pila cuando la lógica que le precede realiza una transición de flanco positivo.

   Sintaxis: EU

9. **Edge Down.** Carga un uno en la pila cuando la lógica que le precede realiza una transición de flanco negativo.

   Sintaxis: ED

10. **And Load.** Efectúa una operación And con el primero y segundo valor de la pila lógica, dejando el resultado en el tope de ésta.

    Sintaxis: ALD

11. **Or Load.** Efectúa una operación OR con el primero y segundo valor de la pila lógica, dejando el resultado en el tope de ésta.

    Sintaxis: OLD

12. Logic Push. Duplica el tope de la pila, el último valor se pierde.

Sintaxis: LPS

13. Logic Read. Copia el segundo valor de la pila en el tope de ésta.

Sintaxis: LRD

14. Logic Pop. Extrae el tope de la pila.

Sintaxis: LPP

15. Output. Escribe el valor del tope de la pila en un bit.

Sintaxis: = *byte.bit*

16. **Set.** Coloca en alto n bits a partir del bit x.

Sintaxis: Set_Bits *n,byte.bitx*

17. **Reset.** Coloca en bajo n bits a partir del bit x.

Sintaxis: Reset_Bits *n,byte.bitx*

18. **Nothing.** No efectúa operación alguna. Simplemente ocupa 11.4 μS en tiempo de procesamiento.

Sintaxis: NOTH

19. **Load Byte Equal.** Compara dos bytes, si los dos bytes son iguales, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDB= *byte1,byte2*

20. **Load Word Equal.** Compara dos Words, si los dos son iguales, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDW= word*1,word2*

21. **Load Double Equal.** Compara dos Doubles, si los dos son iguales, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDD= double*1,double2*

22. **Load Real Equal.** Compara dos reales, si los dos son iguales, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDR= *real1,real2*

23. **Load Byte Greater.** Compara dos bytes, si el primero es mayor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDB> *byte1,byte2*

24. **Load Word Greater.** Compara dos Words, si el primero es mayor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDW> *word1,word2*

25. **Load Double Greater.** Compara dos Doubles, si el primero es mayor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDD> *double1,double2*

26. **Load Real Greater.** Compara dos Reales, si el primero es mayor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDR> *real1,real2*

27. **Load Byte Less than.** Compara dos Bytes, si el primero es menor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDB< *byte1,byte2*

28. **Load Word Less than.** Compara dos Words, si el primero es menor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDW< *word1,word2*

29. **Load Double Less than.** Compara dos Doubles, si el primero es menor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDD< *double1,double2*

30. **Load Real Less than.** Compara dos Reales, si el primero es menor que el segundo, carga un uno en la pila, de lo contrario carga un cero.

Sintaxis: LDR< *real1,real2*

31. **Byte to Integer.** Convierte un Byte en un Entero.

Sintaxis: BTI *byte,word*

32. **Byte to Double.** Convierte un Byte en un Double.

Sintaxis: BTD *byte,double*

33. **Byte to Real.** Convierte un Byte en un Real.

Sintaxis: BTR *byte,real*

34. **Integer to Double.** Convierte un entero en un Double.

Sintaxis: ITD *word,double*

35. **Integer to Real.** Convierte un Entero en un Real.

Sintaxis: ITR *word,real*

36. **Double to Integer.** Convierte un Double en un Entero.

Sintaxis: DTI *double,word*

37. **Double to Real.** Convierte un Double en un Real.

Sintaxis: DTR *double,real*

38. **Real to Byte.** Convierte un Real en un Byte.

Sintaxis: RTB *real,byte*

***39.* Real to Integer.** Convierte un Real en un Entero.

Sintaxis: RTI *real,word*

***40.* Real to Double.** Convierte un Real en un Double.

Sintaxis: RTD *real,double*

***41.* Count Up.** Instrucción para la utilización de los contadores del sistema. Utiliza dos parámetros y dos valores de la pila lógica. Hace que el *Contador X* incremente su conteo cuando se presenta un flanco positivo en el segundo valor de la pila lógica. Reinicia con cero el valor actual del *Contador X* cuando el tope de la pila lógica se hace igual a uno. Cuando la cuenta actual es mayor o igual que el *valor predeterminado*, se enciende el bit *Cx*.

Sintaxis: CTU *ContadorX,valor predetermindado*

***42.* Timer On Retentive.** Habilita / deshabilita el *Temporizador X* dependiendo del valor del tope de la pila lógica (0 = deshabilitado, 1 = habilitado). Cuando el valor del *Temporizador X* es mayor o igual que el *valor predeterminado*, se enciende el bit *TX*.

Sintaxis: TONR *TemporizadorX,valor predeterminado*

***43.* Timer Reset.** Asigna el valor cero al registro del *Temporizador X*.

Sintaxis: TRESET *TemporizadorX*

***44.* Invert Byte.** Complementa un byte.

Sintaxis: INVB *byte*

***45.* Invert Word.** Complementa un Word.

Sintaxis: INVW *word*

***46.* Invert Double.** Complementa un Double.

Sintaxis: INVD *double*

47. **And Byte.** Realiza una operación AND entre dos Bytes. El resultado es colocado en el tercer parámetro de la instrucción.

Sintaxis: ANDB *byte1,byte2,byte3*

48. **And Word.** Realiza una operación AND entre dos Words. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ANDW *word1,word2,word3*

49. **And Double.** Realiza una operación AND entre dos Doubles. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ANDD *double1,double2,double3*

50. **Or Byte.** Realiza una operación OR entre dos Bytes. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ORB *byte1,byte2,byte3*

51. **Or Word.** Realiza una operación OR entre dos Words. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ORW *word1,word2,word3*

52. **Or Double.** Realiza una operación OR entre dos Doubles. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ORD *double1,double2,double3*

53. **Xor Byte.** Realiza una operación XOR entre dos Bytes. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: XORB *byte1,byte2,byte3*

54. **Xor Word.** Realiza una operación XOR entre dos Words. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: XORW *word1,word2,word3*

55. **Xor Double.** Realiza una operación XOR entre dos Doubles. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: XORD *double1,double2,double3*

56. **Move Byte.** Mueve el valor del *byte1* hacia el *byte2.*

Sintaxis: MOVB *byte1,byte2*

57. **Move Word.** Mueve el valor del *word1* hacia el *word2.*

Sintaxis: MOVW *word1,word2*

58. **Move Double.** Mueve el valor del *double1* hacia el *double2.*

Sintaxis: MOVD *double1,double2*

59. **Move Real.** Mueve el valor del *real1* hacia el *real2.*

Sintaxis: MOVR *real1,real2*

60. **Shift Left Byte.** Desplaza hacia la izquierda n posiciones de un byte, llenando con ceros las posiciones de la derecha. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHLB *n,inbyte,outbyte*

61. **Shift Right Byte.** Desplaza hacia la derecha n posiciones de un Byte, llenando con ceros las posiciones de la izquierda. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHRB *n,inbyte,outbyte*

62. **Shift Left Word.** Desplaza hacia la izquierda n posiciones de un Word, llenando con ceros las posiciones de la derecha. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHLW *n,inword,outword*

63. **Shift Right Word.** Desplaza hacia la derecha n posiciones de un Word, llenando con ceros las posiciones de la izquierda. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHRW *n,inword,outword*

64. **Shift Left Double.** Desplaza hacia la izquierda n posiciones de un Double, llenando con ceros las posiciones de la derecha. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHLD *n,indouble,outdouble*

65. Shift Right Double. Desplaza hacia la derecha n posiciones de un double, llenando con ceros las posiciones de la izquierda. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: SHRD *n,indouble,outdouble*

66. Rotate Left Byte. Rota hacia la izquierda n posiciones de un Byte. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RLB *n,inbyte,outbyte*

67. Rotate Right Byte. Rota hacia la derecha n posiciones de un byte. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RRB *n,inbyte,outbyte*

68. Rotate Left Word. Rota hacia la izquierda n posiciones de un Word. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RLW *n,inword,outword*

69. Rotate Right Word. Rota hacia la derecha n posiciones de un Word. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RRW *n,inword,outword*

70. **Rotate Left Double.** Rota hacia la izquierda n posiciones de un Double. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RLD *n,indouble,outdouble*

71. **Rotate Right Double.** Rota hacia la derecha n posiciones de un Double. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: RRD *n,indouble,outdouble*

72. **Increment Byte.** Incrementa un Byte. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: INCB *byte,outbyte*

73. **Decrement Byte.** Decrementa un Byte. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: DECB *byte,outbyte*

74. **Increment Word.** Incrementa un Word. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: INCW *word,outword*

75. **Decrement Word.** Decrementa un Word. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: DECW *word,outword*

76. **Increment Double.** Incrementa un Double. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: INCD *double,outdouble*

77. **Decrement Double.** Decrementa un Double. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: DECD *double,outdouble*

78. **Add.** Suma dos números reales. Coloca el resultado en el tercer parámetro de la instrucción.

    Sintaxis: ADD *real1,real2,outreal*

79. **Subtract.** Resta el *real2* del *real1*. Coloca el resultado en el tercer parámetro de la instrucción.

    Sintaxis: SUB *real1,real2,outreal*

80. **Multiply.** Multiplica dos números reales. Coloca el resultado en el tercer parámetro de la instrucción.

    Sintaxis: MULT *real1,real2,outreal*

81. **Divide.** Divide al *real1* entre el *real2*. Coloca el resultado en el tercer parámetro de la instrucción.

    Sintaxis: DIV *real1,real2,outreal*

82. **Change Sign.** Cambia el signo de un número real. Coloca el resultado en el segundo parámetro de la instrucción.

    Sintaxis: CHS *real1,outreal*

83. **Absolute Value.** Calcula el valor absoluto de un número real. Coloca el resultado en el segundo parámetro de la instrucción.

    Sintaxis: ABS *real1,outreal*

84. **Square Root.** Calcula la raíz cuadrada de un número real. Coloca el resultado en el segundo parámetro de la instrucción.

    Sintaxis: SQRT *real1,outreal*

85. **Natural Logarithm.** Calcula el logaritmo natural de un número real. Coloca el resultado en el segundo parámetro de la instrucción.

    Sintaxis: LOG *real1,outreal*

86. Base 10 Logarithm. Calcula el logaritmo base 10 de un número real. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: LOG10 *real1,outreal*

***87.* Exponential.** Calcula e^X, donde x es un número real. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: EXP *x,outreal*

88. Base 10 Exponential. Calcula 10^X, donde x es un número real. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: EXP10 *x,outreal*

89. Power. Calcula X^Y, donde *X* y *Y* son dos números reales. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: POW *x,y,outreal*

90. Root. Calcula $X^{1/Y}$, donde *X* y *Y* son dos números reales. Coloca el resultado en el tercer parámetro de la instrucción.

Sintaxis: ROOT *x,y,outreal*

91. Reciprocal. Calcula 1/x, donde x es un número real. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: RECIP *x,outreal*

92. Sine. Calcula seno(x), donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: SIN *x,outreal*

93. Arcsine. Calcula $\text{seno}^{-1}(x)$, donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: ASIN *x,outreal*

94. Cosine. Calcula cos(x), donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: COS *x,outreal*

95. Arcosine. Calcula $\cos^{-1}(x)$, donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: ACOS *x,outreal*

96. Tangent. Calcula tan(x), donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: TAN *x,outreal*

97. Arctangent. Calcula $\tan^{-1}(x)$, donde x es un número real cuya unidad es el radián. Coloca el resultado en el segundo parámetro de la instrucción.

Sintaxis: ATAN *x,outreal*

98. Jump. Traslada el control del programa hacia una *etiqueta.*

Sintaxis: JMP *etiqueta*

99. Call Subroutine. Llama a una subrutina.

Sintaxis: CALLSBR *etiqueta*

100. Return. Retorna de una subrutina.

Sintaxis: RET

101. Conditional Return. Retorna condicionalmente de una subrutina, dependiendo de la lógica precedente. Retorna si el tope de la pila lógica es igual a uno.

Sintaxis: CRET

102. End Program. Finaliza la Ejecución del Programa en el Ciclo actual.

Sintaxis: ENDP

103. Stop. Activa el modo detenido del iPLC. El modo detenido se activa hasta que finaliza la ejecución del Ciclo actual.

Sintaxis: STOP

104. Watchdog Timer Reset. Reinicializa el Reloj de vigilancia con el valor cero (0).

Sintaxis: WDTR

105. Read Real Time Clock. Realiza la lectura de los trece registros internos en el reloj de tiempo real, correspondientes a la fecha y hora actuales, y los escribe en los trece registros de la memoria R.

Sintaxis: RTCRD

106. Write Real Time Clock. Escribe el contenido de la región de memoria R en los trece registros internos del reloj de tiempo real. Esta instrucción es utilizada para establecer la fecha y hora en el reloj de tiempo real, por ejemplo debido a cambios en la hora local.

Sintaxis: RTCWR

VIII. RESULTADOS OBTENIDOS

En este capítulo se presentarán las características del PLC desarrollado y se discutirán las diferencias con relación al PLC S7-200 con CPU 212 versión 0.91 de la empresa alemana Siemens, con el propósito de ofrecer una forma de comparación y evaluación del proyecto implementado. Además se mencionará lo más relevante en cuanto a la habilitación de la Internet en el iPLC y se incluye una tabla de costos del iPLC.

A. Características del sistema desarrollado

Para ofrecer una presentación eficiente, se utilizan tablas comparativas entre las características del iPLC y del PLC S7-200 con CPU 212 v.91. De esta forma se pueden apreciar clara y rápidamente todas las diferencias.

En la figura 8.1 se muestra la tarjeta madre del iPLC creado en este proyecto. Se indican todos sus componentes principales, a excepción del bus de expansión que está desconectado.

1. Terminal de alimentación +24 Vdc
2. Terminal de salida de +5V dc
3. Terminal negativa de la alimentación (Gnd)
4. Microcontrolador central PIC16F877
5. Módulo SitePlayer
6. Memoria flash
7. Memoria RAM
8. Coprocesador matemático PAK-IX
9. Banco de Flip-Flops
10. Transformador / filtro Ethernet
11. Reloj de tiempo real
12. Jack RJ-45
13. Baterías de apoyo para el RTC
14. Regulador de voltaje general
15. LED's de estado del iPLC
16. Reloj TTL
17. Regulador de voltaje para grabación en memoria flash.

Figura 8.1: "Tarjeta madre del iPLC"

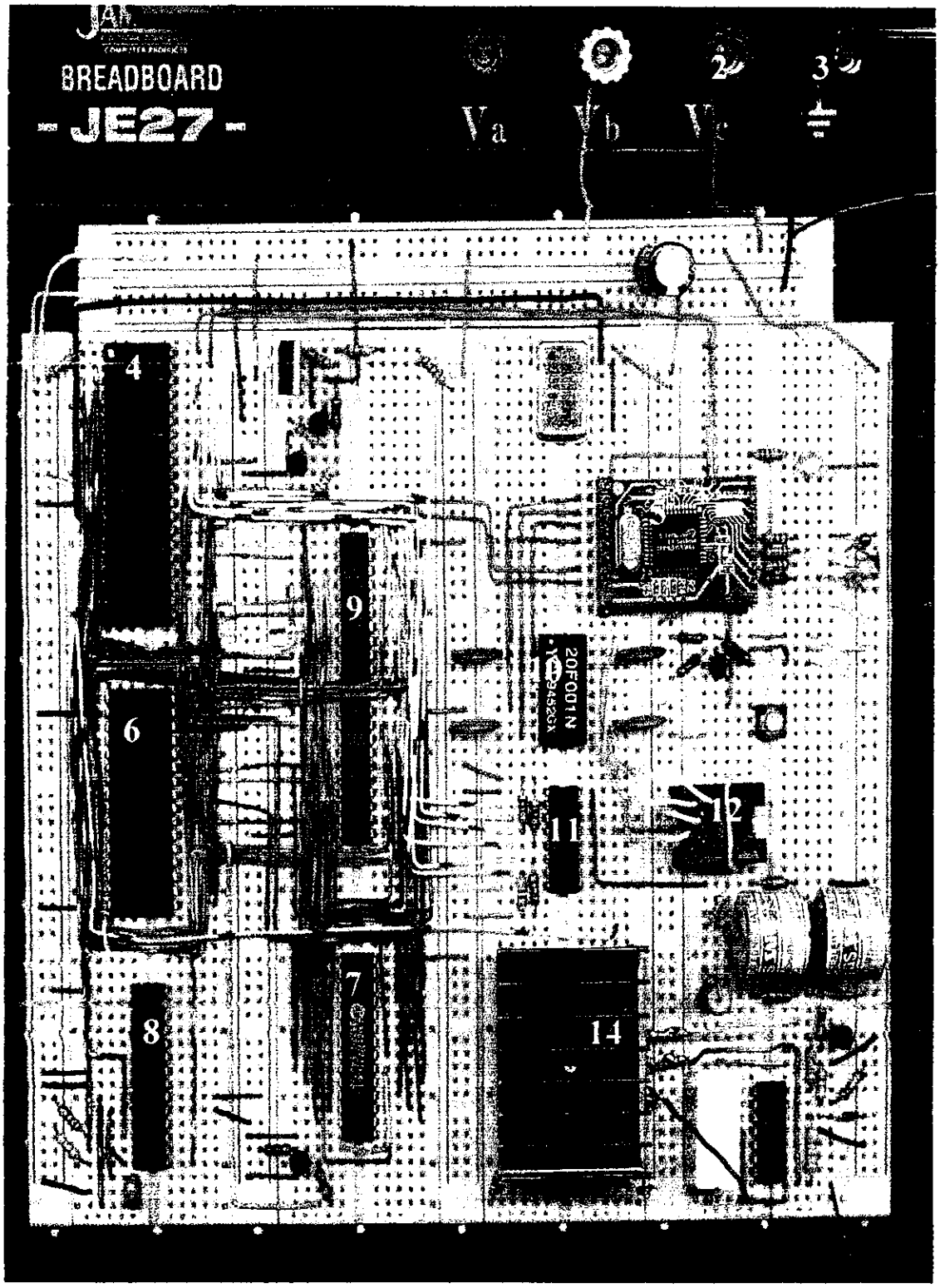


Tabla 8.1: "Especificaciones del iPLC y del PLC Siemens S7-200-212"

Característica	S7-200-212	iPLC
Alimentación	+24Vdc	+24 Vdc
Potencia disipada	5 W	5.40 W
Máximo tamaño del programa de usuario	1 KB	64 KB
Memoria de usuario (Memoria V)	1 KB	28 KB
Número máximo de módulos de expansión	2	15
Máxima cantidad de puntos digitales de E / S	64 / 64	120 / 120
Máxima cantidad de puntos analógicos de E / S	16 / 16	240 / 240
Temporizadores	64	96
Contadores	64	40
Contadores rápidos	1	0
Eventos de interrupción	6	0
Reloj de tiempo real	NO	SI
Máxima cantidad de instrucciones de salto (JMP)	64	Sin límite
Máxima cantidad de subrutinas	16	Sin límite
Tiempo de ejecución booleana	1.2 μS	5.6 μS
Funciones matemáticas	16/32 bits, enteros	32 bits, punto flotante
Protocolos de comunicación compatibles	PPI, FreePort	HTTP,TCP, IP, DHCP
Interfaz de comunicación	RS-485	Ethernet 10BASE-T

Los protocolos e interfaz de comunicación no pueden compararse, pues simplemente se trata de cosas distintas para cada PLC (el iPLC y el S7-200). Sin embargo, se debe destacar lo siguiente:

- La versión actual del iPLC no posee capacidad para conectarse en redes de PLC's.
- Los protocolos y la interfaz de comunicación en el S7-200 pueden ser utilizados para establecer comunicación con un dispositivo maestro, o para conectarse en una red de PLC's (Intranet).
- Los protocolos y la interfaz de comunicación en el iPLC son utilizados para conectarse en una sola red: La Internet.

1. Desventajas del iPLC con relación al PLC S7-200 con CPU 212 de la empresa alemana Siemens. Como puede verse en la tabla 8.1, el iPLC posee cuatro desventajas con respecto a las características fundamentales de los PLC's S7-200 con CPU 212.

a. Contadores. El iPLC posee cuarenta contadores mientras que el S7-200 posee 64. No cabe duda que es una diferencia apreciable, sin embargo, cuarenta contadores es una cantidad que podría ser insuficiente únicamente en aplicaciones bastante grandes. Se considera que la cantidad de contadores implementada en el iPLC puede ser suficiente para aplicaciones pequeñas / medianas.

b. Contadores rápidos. El iPLC no posee contadores rápidos. Esto es una limitante únicamente en las aplicaciones de conteo a muy alta velocidad, que probablemente no representan la mayoría de las aplicaciones pequeñas / medianas.

c. Eventos de interrupción. El iPLC no posee eventos de interrupción. Nuevamente, esto es una limitante para aplicaciones de muy alta velocidad, en las cuales el tiempo de ciclo debe ser tan pequeño, que se requiere de interrupciones para la atención / detección de ciertos eventos. En las aplicaciones que no sean tan rápidas la función de las interrupciones puede ser prescindible, ya que se pueden incluir

instrucciones que revisen la ocurrencia de los eventos, de forma periódica, con cada ciclo del iPLC.

d. Velocidad de ejecución. El iPLC es más lento que el S7-200. Esta es una cuestión que nuevamente puede ser problema para la detección de eventos en aplicaciones muy rápidas. La velocidad de ejecución booleana del iPLC fue calculada a partir del tiempo de ejecución de la instrucción ALD, que ejecuta una operación AND con los dos primeros valores de la pila lógica. Esa instrucción del iPLC está implementada con veintiocho instrucciones internas del microcontrolador central PIC16F877. Cada instrucción del PIC16F877 utiliza 200 nS en tiempo de ejecución, de tal forma que el tiempo de la instrucción ALD es igual a 5.6 µS.

2. Ventajas del iPLC con relación al PLC S7-200 con CPU 212 de la empresa alemana Siemens. Como puede verse en la tabla 8.1, el iPLC posee diversas ventajas con respecto a las características fundamentales de los PLC's S7-200 con CPU 212 v 0.91.

a. Memoria de programa. El iPLC posee una capacidad de almacenamiento de programa bastante superior a la del S7-200. Con esto prácticamente se garantiza que el usuario no tendrá problemas en cuanto al tamaño del programa que desee cargar al iPLC, ya que la capacidad de almacenamiento es en realidad grande (comparada con la capacidad de almacenamiento de PLC's comerciales de rango medio).

b. Memoria de usuario (RAM). La capacidad de almacenamiento en RAM del iPLC también es bastante superior a la del S7-200. Con esto prácticamente se garantiza que el programa de usuario no tendrá problemas de almacenamiento temporal para el procesamiento de información durante la fase de ejecución del programa.

c. Módulos de expansión y puntos de E / S digitales y analógicos. El iPLC posee mayor capacidad de ampliación que el S7-200. Esto provoca directamente que pueda poseer más puntos de entrada / salida, no sólo digitales o analógicos, sino de

cualquier otro tipo especial que se encuentre implementado en un módulo de expansión. Esto es importante, ya que al soportar más módulos de expansión, se pueden instalar más funciones especiales.

d. Temporizadores. El iPLC posee 32 temporizadores más que el S7-200. Aunque el S7-200 ya posee bastantes temporizadores (64), esto no deja de ser una ventaja que pueda pesar en ciertas aplicaciones.

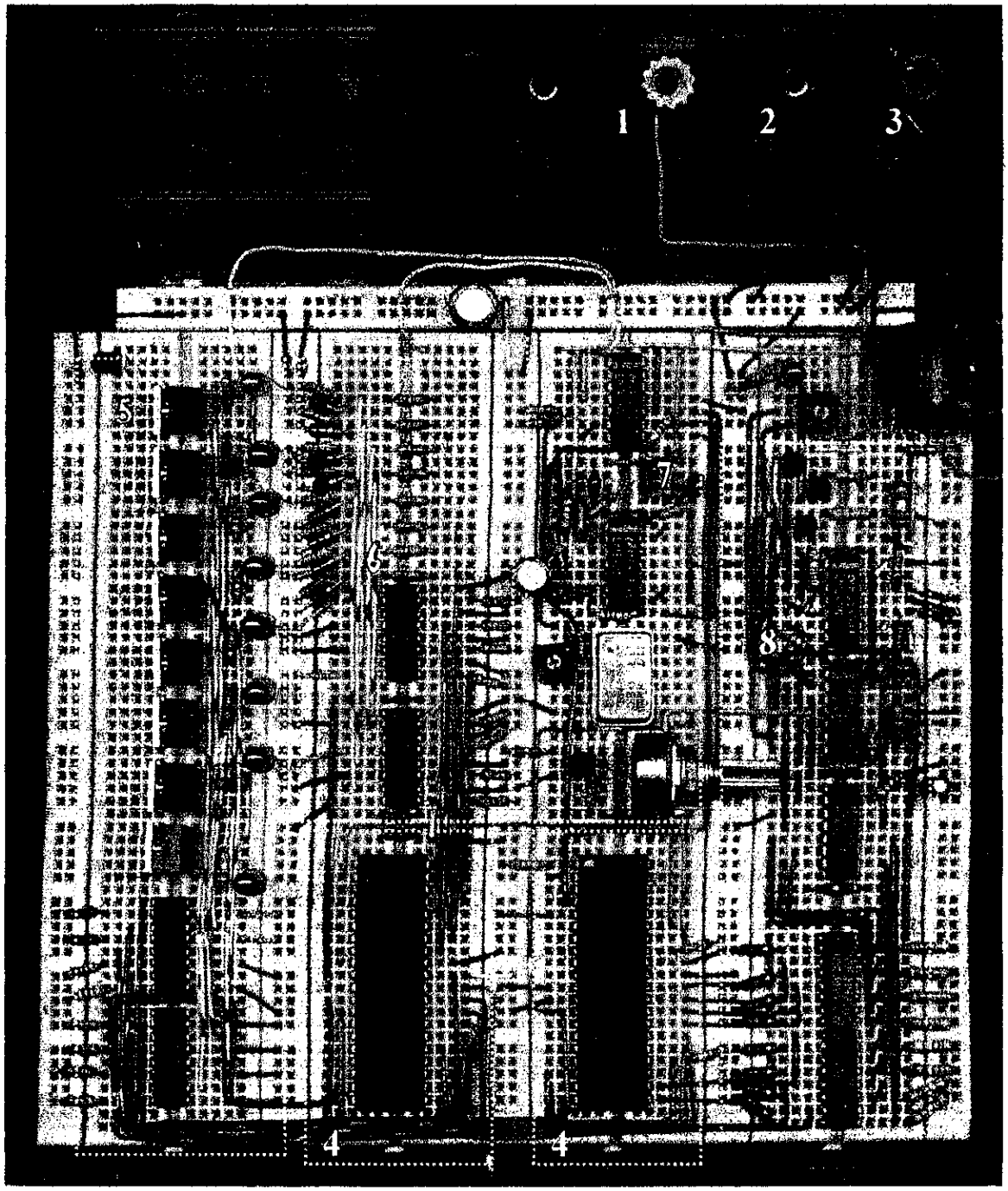
e. Reloj de tiempo real. El S7-200 con CPU 212 no posee reloj de tiempo real. Es claro que poseer este dispositivo es una ventaja interesante, pues con él, todas las instrucciones del iPLC pueden tomar en cuenta la hora y fecha actuales, de tal forma que en una planta se puedan controlar tareas dependientes del horario real. Los modelos superiores de la serie S7-200 sí incluyen este dispositivo.

f. Instrucciones de salto y subrutinas. El software no impone límites, el único límite es la capacidad de almacenamiento para el programa de usuario (64KB). En el programa de usuario se pueden incluir tantas instrucciones de salto de y subrutinas como se deseen, siempre que el programa completo no sobrepase la capacidad de la memoria flash del iPLC. Por lo tanto, fácilmente pueden incluirse cientos de instrucciones de salto y subrutinas.

g. Funciones matemáticas. El potencial matemático del iPLC es muy superior al del S7-200 con CPU 212. El S7-200 con CPU 212 únicamente puede sumar / restar números enteros de 32 bits, y multiplicar / dividir números enteros de 16 bits. En cambio, el iPLC posee un extenso conjunto de funciones matemáticas de 32 bits con punto flotante (ver el capítulo VII, sección B).

A continuación se presentan las tablas de datos acerca de los módulos de expansión desarrollados. En cada tabla se incluye la información acerca de un módulo de expansión Siemens compatible con la serie S7-200, únicamente con el objetivo de poder tener una referencia para las características fundamentales de todos los módulos de expansión.

Figura 8.2: "Módulos de expansión"



1. Terminal de alimentación +24 Vdc
2. Terminal de salida +5 Vdc
3. Terminal negativa de la alimentación (Gnd).
4. Módulo de expansión modelo
5. Electrónica específica para las entradas digitales
6. Electrónica específica para las salidas digitales
7. Electrónica específica para las entradas analógicas
8. Electrónica para la salida analógica

Tabla 8.2: "Especificaciones del módulo de expansión digital"

Característica	iMxD1 (iPLC)	EM221	EM222
Alimentación	+24Vdc	+24Vdc	+24Vdc
Disipación de potencia	1.5 W	2 W	4W
Entradas	Tipo sumidero	Tipo sumidero	-----
Puntos de entrada	8	8	-----
Mínimo voltaje de entrada (ON)	15Vdc, a un mínimo de 4.2 mA	15Vdc, a un mínimo de 4 mA	-----
Voltaje nominal de entrada (ON)	24Vdc	24Vdc	-----
Voltaje máximo de entrada (OFF)	5Vdc	5Vdc	-----
Aislamiento eléctrico (acoplado ópticamente)	5KV pico	500 Vac, 1 min	-----
Salidas	Transistor tipo fuente	-----	Transistor tipo fuente
Puntos de salida	8	-----	8
Voltaje nominal de salida	24Vdc	-----	24Vdc
Máxima corriente de carga	4 A en un solo punto, 10 A en todos los puntos	-----	0.75 A en un solo punto, 4 A entre todos los puntos
Aislamiento eléctrico (acoplado ópticamente)	5KV pico	-----	500 Vac, 1 min

Tabla 8.3: "Especificaciones del módulo de expansión analógico"

Característica	iMxA1 (iPLC)	EM231	EM232
Alimentación	+24Vdc	+24Vdc	+24Vdc
Disipación de potencia	3.2 W	2 W	2 W
Entradas	Simples: 1 Diferenciales: 1	Voltaje: 3 diferenciales, Corriente: 1	-----
Filtro de entrada	-3 dB @ 3.1 KHz	-3 dB @ 3.1 KHz	-----
Máximo voltaje de entrada	12.8 V (rango 0 V a 10 V)	30 V (rango de 0V a 10 V)	-----
Resolución	8 bits	12 bits	-----
Conversión A/D	< 43 μS	< 250 μS	-----
Salidas	Voltaje: 0 a 10 V Corriente: 0 a 20mA	-----	Voltaje: -10V a +10V Corriente: 0 a 20mA
Resolución	8 bits	-----	Voltaje: 12 bits Corriente: 11 bits
Carga mínima para la salida de voltaje	1 KΩ	-----	5 KΩ
Carga máxima para la salida de corriente	700 Ω	-----	500 Ω

B. Las instrucciones y funcionalidad del iPLC

En total se implementaron 106 instrucciones al iPLC. Con estas instrucciones, y exceptuando las instrucciones relacionadas con las comunicaciones en redes y atención de interrupciones que ofrecen los sistemas S7-200, el iPLC posee casi toda la funcionalidad de dichos sistemas. Más aún, el conjunto de instrucciones del iPLC supera

en algunas características a diversos modelos de la serie S7-200 de la empresa alemana Siemens.

Con propósitos de comparación, se habla de "funcionalidad" y no de "instrucciones" puesto que los PLC's S7-200 poseen más cantidad de instrucciones que el iPLC, pero el iPLC ofrece la misma funcionalidad con un número reducido de instrucciones. De esta forma, para poder evaluar la funcionalidad del iPLC, se presentará la clasificación de todas las instrucciones de los sistemas S7-200, mencionando las diferencias de funcionalidad con respecto al iPLC.

Se debe mencionar una característica importante acerca de las instrucciones del iPLC: Todas sus instrucciones que realizan algún tipo de cálculo u operación a partir de sus argumentos, incluyen un parámetro de salida, en el cual se almacena el resultado y se evita la "destrucción" de uno de los argumentos de entrada. En cambio, utilizando el lenguaje STL de los sistemas S7-200, no es posible indicar un parámetro de salida, de tal forma que todas las funciones "destruyen" uno de los parámetros de entrada, obligando a crear una copia del dato en caso se necesite posteriormente.

A continuación se presenta la clasificación del conjunto de instrucciones de los sistemas S7-200 de la empresa alemana Siemens. Se utiliza la categorización que el fabricante ofrece en la documentación de los sistemas S7-200.

Cada PLC indicará por medio de una "x", si posee implementada la instrucción correspondiente. Si el PLC no posee implementada una instrucción, simplemente presentará en blanco la casilla correspondiente. En las tablas se compara al iPLC contra en el S7-200 con CPU 212 versión 0.91 de Siemens.

Como puede verse en la tabla 8.4, el iPLC no posee las versiones de instrucciones de *operación inmediata*, distinguidas por la letra 'I' al final del nombre de la instrucción. La característica de estas instrucciones es la lectura / escritura inmediata de sus parámetros, sin esperar a las fases de lectura / escritura en el Ciclo de PLC.

Tabla 8.4: "Instrucciones de lógica de bits"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
LD	X	X
A	X	X
O	X	X
LDI	X	
AI	X	
OI	X	
LDNI	X	
ANI	X	
ONI	X	
NOT	X	X
EU,ED	X	X
ALD.OLD	X	X
LPS	X	X
LRD	X	X
LPP	X	X
=	X	X
=I	X	
S,R	X	X
SI,RI	X	
NOP	X	X

Tabla 8.5: "Instrucciones de comunicación e interrupción"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
XMT	X	---
INT	X	---
CRETI	X	---
RETI	X	---
ENI,DISI	X	---
ATCH,DTCH	X	---

Como se ha mencionado anteriormente, el iPLC no posee capacidad de comunicación en redes de PLC's, ni de servicio a interrupciones. Por ello no posee ninguna de las instrucciones de la tabla 8.5.

Tabla 8.6: "Instrucciones de contadores"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
CTU	X	X
CTUD	X	
HDEF,HSC	X	

El iPLC únicamente posee la instrucción de conteo para contadores ascendentes. La serie S7-200 posee otra instrucción para contadores ascendentes / descendentes, y para el control del contador rápido (HDEF,HSC).

Tabla 8.7: "Instrucciones de temporizadores"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
TON	X	
TONR	X	X
TRESET	X	X

El iPLC únicamente posee *temporizadores retentivos*, mientras que la serie s7-200 posee además *temporizadores no retentivos*. Los de tipo *retentivo* conservan su valor actual aún después de ser deshabilitados (aunque ya no se incrementan), mientras que los *no retentivos* se borran al ser deshabilitados.

Tabla 8.8: "Instrucciones de rotación y desplazamiento"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
SHLB, SHRB		X
SHLW, SHRW	X	X
SHLD, SHRD	X	X
RLB, RRB		X
RLW, RRW	X	X
RLD, RRD	X	X

El iPLC posee instrucciones para rotar y desplazar Bytes, Words y Doubles, mientras que el S7-200 no puede realizar estas operaciones con Bytes.

Tabla 8.9: "Instrucciones de control de programa"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
JMP, LABEL	X	X
CALLSBR, SBR	X	X
RET	X	X
CRET	X	X
END	X	X
MEND	X	X
STOP	X	X

El iPLC posee todas las instrucciones de control de programa incluidas en los sistemas S7-200-212 v0.91.

Tabla 8.10: "Instrucciones de comparación"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
LD[B|W|D]=	X	X
A[B|W|D]=	X	
O[B|W|D]=	X	
LD[B|W|D]>=	X	
A[B|W|D]>=	X	
O[B|W|D]>=	X	
LD[B|W|D]<=	X	
A[B|W|D]<=	X	
O[B|W|D]<=	X	
LD[B|W|D]>		X
A[B|W|D]>		X
O[B|W|D]>		X
LD[B|W|D]<		X
A[B|W|D]<		X
O[B|W|D]<		X
LDR=		X
LDR>		X
LDR<		X

La tabla 8.10 muestra que el iPLC es superior en cuanto a las instrucciones de comparación, pues además de implementar toda la funcionalidad de comparación para Bytes, Words y Doubles, posee la capacidad de realizar comparaciones con números reales (32 bits, punto flotante).

Tabla 8.11: "Instrucciones de conversión"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
BCDI, IBCD	X	
DECO	X	
ENCO	X	
ATH, HTA	X	
SEG	X	
BTI		X
BTD		X
BTR, RTB		X
ITD, DTI		X
ITR, RTI		X
RTD, DTR		X

Las primeras cinco instrucciones de la tabla 8.11 básicamente están implementadas para el manejo y conversión de datos durante comunicaciones con otros dispositivos, utilizando redes.

Como ya se indicó, el iPLC no posee capacidad para redes de dispositivos o PLC's, por lo que no posee esas primeras cinco instrucciones. Las restantes seis instrucciones están implementadas únicamente en el iPLC, y representan una ventaja sobre la serie S7-200 ya que los PLC's de esa serie no son capaces de realizar conversiones entre Bytes, Words (enteros), Doubles y Reales.

Tabla 8.12: "Instrucciones de operaciones lógicas"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
INVB	X	X
INVW	X	X
INVD	X	X
ANDB	X	X
ANDW	X	X
ANDD	X	X
ORB	X	X
ORW	X	X
ORD	X	X
XORB	X	X
XORW	X	X
XORD	X	X

Tabla 8.13: "Instrucciones de movimiento / asignación"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
MOVB	X	X
MOVW	X	X
MOVD	X	X
MOVR		X
SWAP	X	
BMB	X	
BMW	X	

El iPLC no posee instrucciones para el movimiento de bloques de Bytes/Words, tampoco posee la instrucción SWAP, que intercambia los dos Bytes de un Word. Sin embargo, la serie S7-200 no posee la capacidad de mover variables reales.

Tabla 8.14: "Instrucciones de reloj de tiempo real"

Instrucción	S7-200 cpu 212 v 0.91	iPLC
RTCRD		X
RTCWR		X

Solo las versiones superiores de la CPU212 poseen instalado un reloj de tiempo real.

Tabla 8.15: "Instrucciones de funciones matemáticas"

Instrucción	S7-200 cpu 212 v 0.91	IPLC
INCB		X
DECB		X
INCW	X	X
DECW	X	X
INCD	X	X
DECD	X	X
ADD	* X	X
SUB	* X	X
CHS	NE	X
ABS	NE	X
MUL	** X	X
DIV	** X	X
SQRT		X
LOG		X
LOG10		X
EXP		X
EXP10		X
POW	NE	X
ROOT	NE	X
RECIP	NE	X
SIN		X
COS		X
TAN		X
ASIN	NE	X
ACOS	NE	X
ATAN	NE	X

* Estas instrucciones están implementadas únicamente para números enteros de 32 bits.

** Estas instrucciones están implementadas únicamente para números enteros de 16 bits.

NE: Estas instrucciones no existen en NINGÚN PLC de la serie S7-200 de Siemens.

C. Conexión a la Internet habilitada en el iPLC

El iPLC fue satisfactoriamente habilitado para la conexión con Internet por medio de la utilización del módulo SitePlayer, producido por la empresa estadounidense Netmedia

Inc. Sobre este módulo se desarrolló todo el entorno de interacción con el usuario remoto. Los métodos de interacción con el usuario remoto fueron implementados por medio del software desarrollado en los lenguajes HTML y JavaScript.

La habilitación de la Internet realizada en el iPLC presenta dos esquemas. El primer esquema consiste en la tecnología y los medios de habilitación para cualquier sistema microcontrolado. El segundo esquema consiste en el software que permite la interacción entre el PLC desarrollado y un usuario remoto en un explorador de Internet.

A continuación se listan las características principales de la habilitación de la Internet efectuada en el iPLC, con el esquema de la tecnología de habilitación:

- El iPLC maneja los protocolos ARP, TCP, IP, ICMP, DHCP y HTTP.
- El iPLC posee un servidor HTTP incrustado en la tarjeta madre del sistema.
- El iPLC posee un circuito integrado controlador de Ethernet, un transformador / filtro de Ethernet y un conector Jack RJ-45 que le permiten conectarse directamente a una red Ethernet.

A continuación se listan las características principales de la habilitación de la Internet efectuada en el iPLC, con el esquema del software de interacción con el usuario:

- El servidor HTTP del iPLC ofrece la página HTML "***Prog.htm***", en la cual se puede seleccionar el modo de operación y reprogramar al iPLC, de forma remota, desde un explorador de Internet. Esta página se muestra en la figura 8.3.
- El servidor HTTP del iPLC ofrece la página HTML "***Mon1.htm***", en la cual se puede monitorizar a todas las variables del iPLC, de forma remota, desde un explorador de Internet. Esta página se muestra en la figura 8.4.
- El servidor HTTP del iPLC ofrece la página HTML "***Mods.htm***", en la cual se muestra la descripción de todos los módulos de expansión conectados al iPLC. Esta página HTML sirve como una herramienta más para la monitorización del sistema. Esta página se muestra en la figura 8.5.

Figura 8.3: "Página del entorno de programación del iPLC"

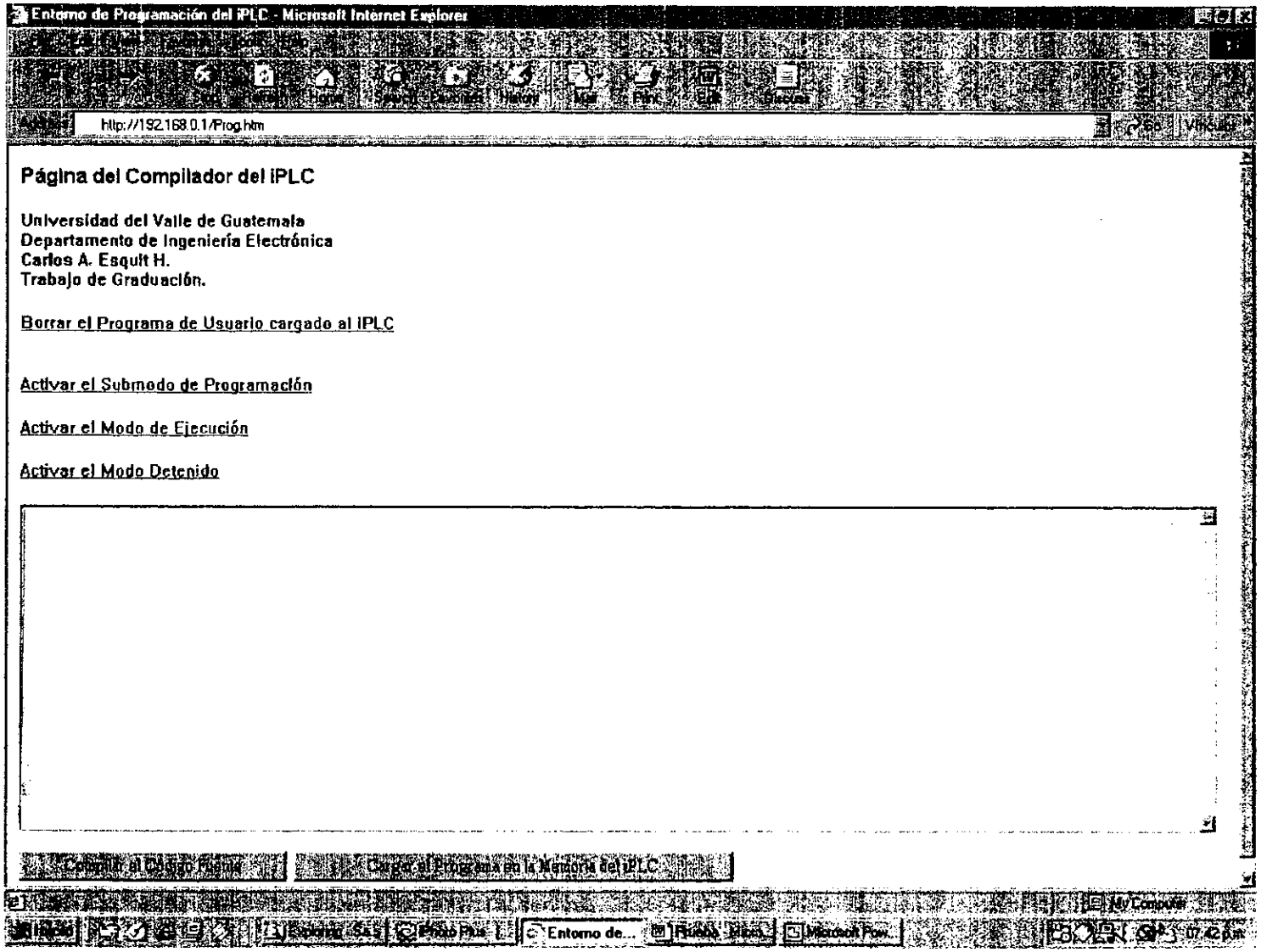


Figura 8.4: "Página de monitorización de variables (Mon1.htm)"

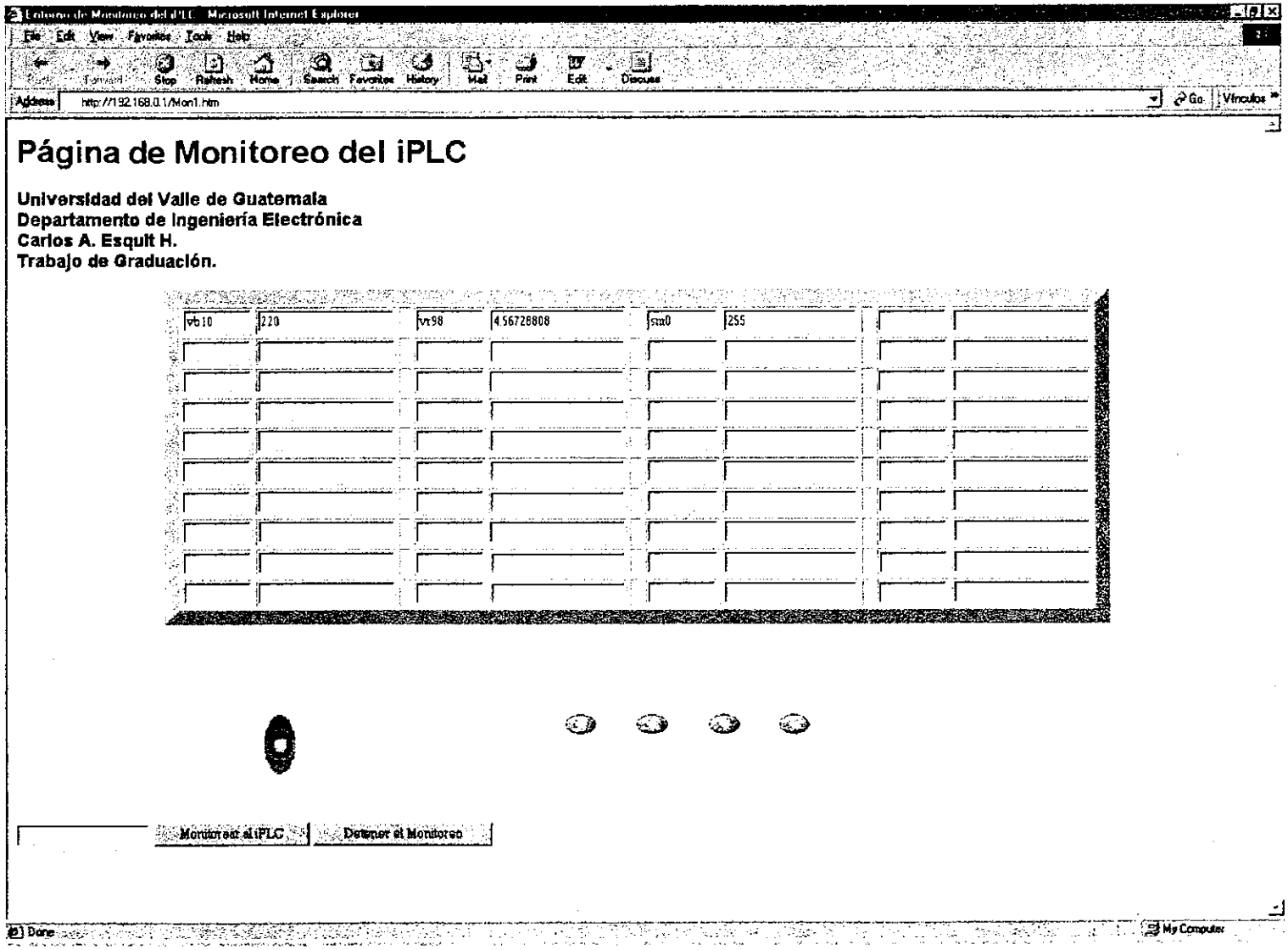
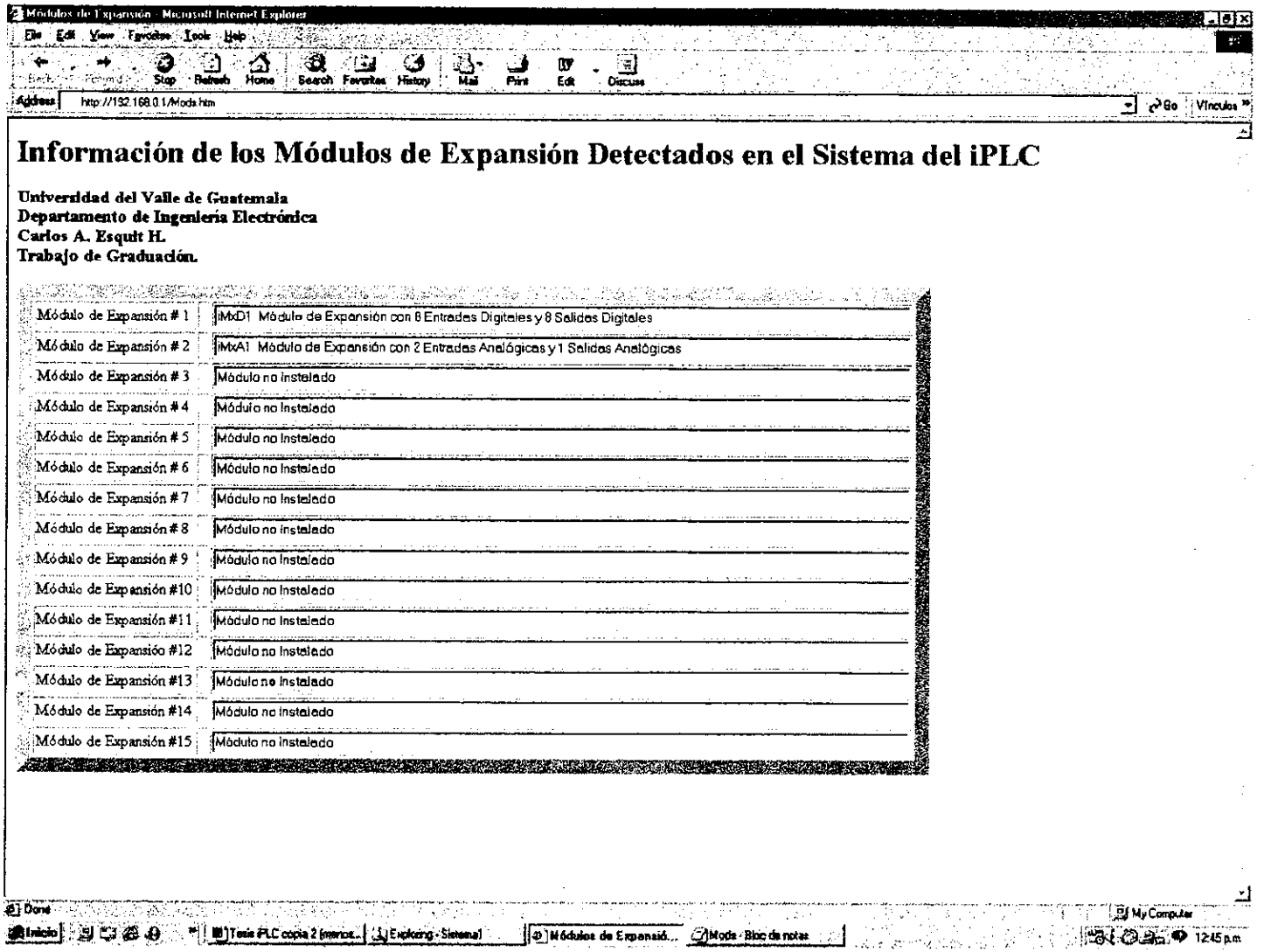


Figura 8.5: "Página de información acerca de los módulos de expansión detectados"



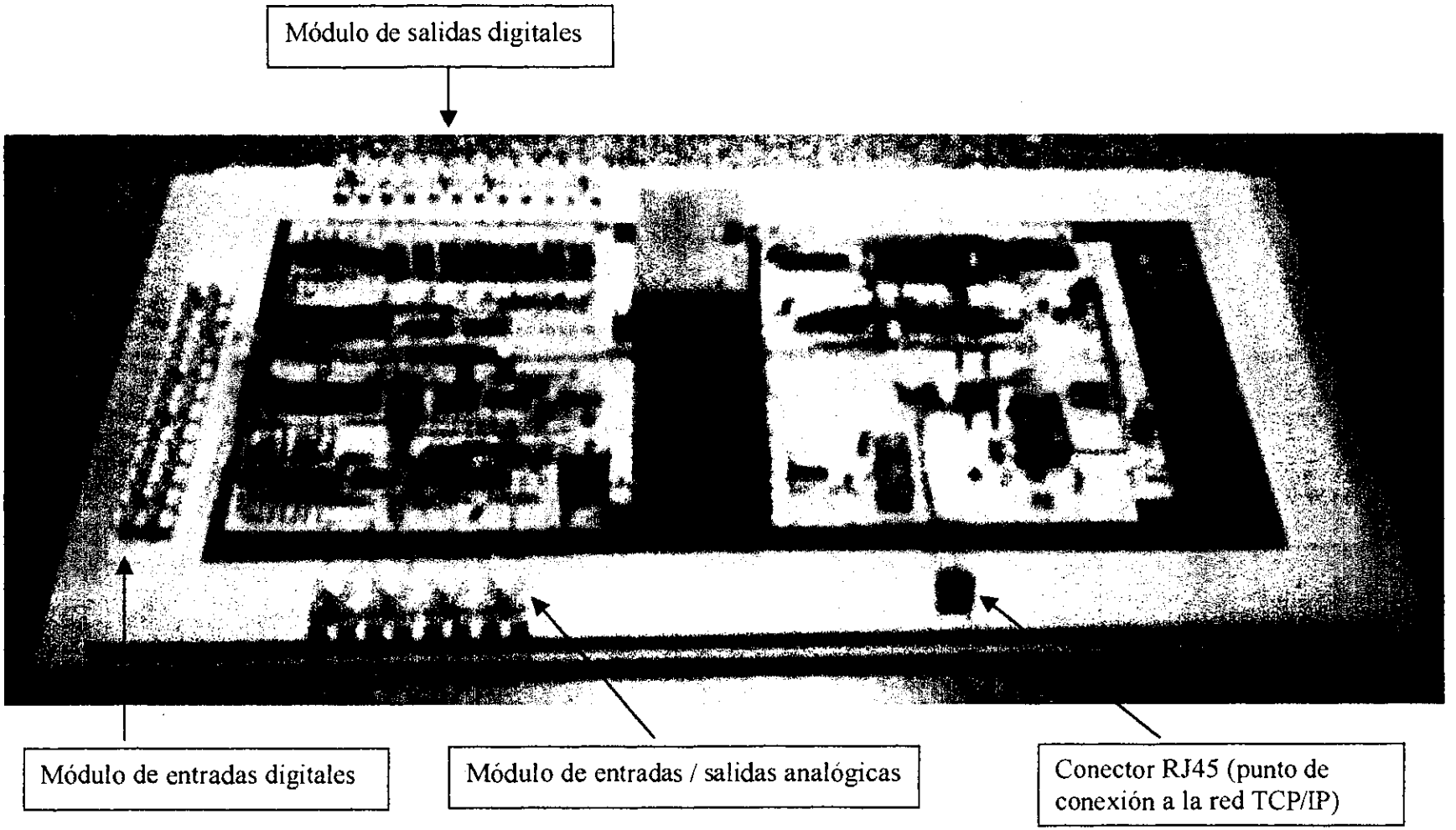
D. Costos del iPLC

En la tabla 8.16 se presentan los costos del iPLC, únicamente se incluyen los costos de todos los componentes electrónicos. En esa tabla se incluyen los costos de la tarjeta madre y de los dos módulos de expansión desarrollados.

Tabla 8.16: "Lista de costos del iPLC"

Componente	Costo (Dólares estadounidenses)
Microcontrolador PIC16F877 x 3	28.47
Coprocesador matemático PAK-IX	29.95
Memoria flash CAT28F020	7.95
Memoria RAM CY7C199	3.75
Reloj de tiempo real RTC58321	6.95
Módulo SitePlayer	29.95
Acopladores ópticos x 6	8.95
Flip-Flops ECG74LS374 x 2	0.78
Reloj de 20 MHz x 3	3.87
Diodos 1N4001 x 8	0.32
Transistores TIP125 x 8	4.72
Transistores ECG123AP x 8	0.48
Op-Amps LM348N x 4	1.96
Baterías recargables x 2	6.95
Resistores x 120	2.75
Total	**$ 137.80**

Figura 8.6: "Prototipo implementado del iPLC"



IX. CONCLUSIONES Y RECOMENDACIONES

A. Conclusiones

- Se logró construir un PLC que posee las funciones básicas de los PLC's de la serie S7-200 de la empresa alemana Siemens.
- Las desventajas más notables del iPLC con relación a los PLC's de la serie S7-200 son la carencia de interfaces de comunicación con otros PLC's, la carencia de eventos de interrupción y la menor velocidad de procesamiento de instrucciones.
- Las ventajas más notables del iPLC con relación a los PLC's de la serie S7-200 son la capacidad de almacenamiento para el programa y datos de usuario, la capacidad de expansión, el potencial matemático y la capacidad para ser programado y monitorizado a través de la Internet.
- El módulo SitePlayer permite habilitar la Internet en un sistema microcontrolado de forma fácil, eficiente y económica.

B. Recomendaciones

- Investigar la viabilidad de sustituir el microcontrolador central PIC16F877 por otro más veloz, como los producidos por Scenix, que son hasta 10 veces más veloces, con lo cual se podría desarrollar un PLC mucho más rápido.
- Otra buena opción sería utilizar un microcontrolador Microchip de la serie 17C7xx ó de la serie 18Cxxx ya que con estos se dispondría de un segundo puerto serial, que podría ser utilizado para establecer una red simple de iPLC's y / o establecer comunicación con otros dispositivos. Además se dispondría de más bits de entrada / salida para interactuar con más periféricos y desarrollar un sistema de interrupciones.

X. BIBLIOGRAFÍA

December, John y Ginsburg, Mark. 1996. *HTML3.2 & CGI.* Indianapolis, Sams.net Publishing. 1321 págs.

Romero, Luis Fernando. 1997. *Publicar en Internet: guía práctica para la creación de documentos HTML.* Santander, Servicio de Publicaciones de la Universidad de Cantabria. 360 págs.

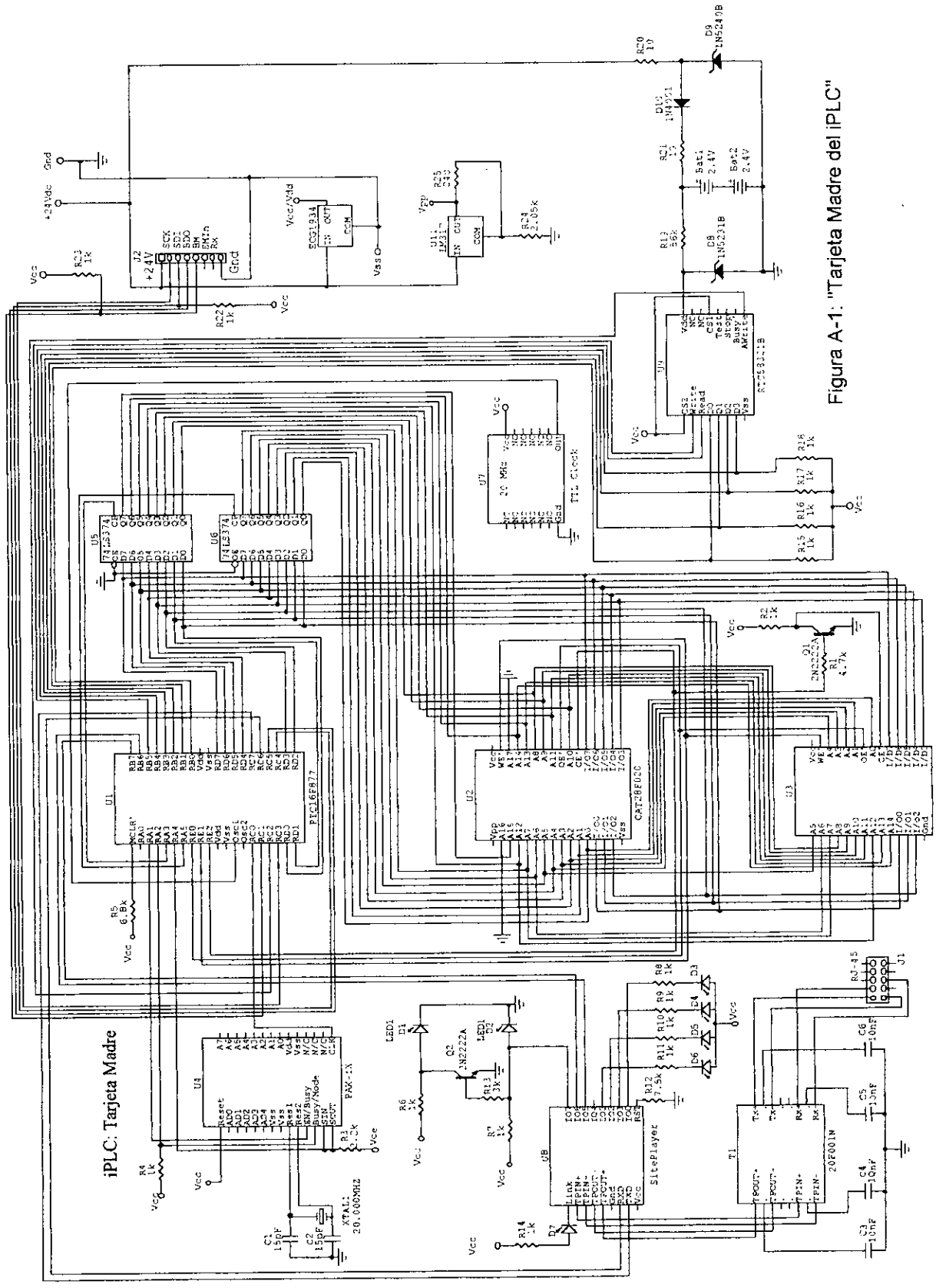
Savitch, Walter J. 1999. *Java, an Introduction to Computer Science and Programming.* New Jersey, Prentice Hall. 726 págs.

Sistema de automatización S7-200. 1998. Siemens AG. Nuernberg. 478 págs.

XI. APÉNDICES

APÉNDICE A

Diagramas de los circuitos del iPLC


iPLC: Tarjeta Madre
PIC16F877
CAT28F020
SitePlayer
20F001N
PAK-IN
Figura A-1: "Tarjeta Madre del iPLC"

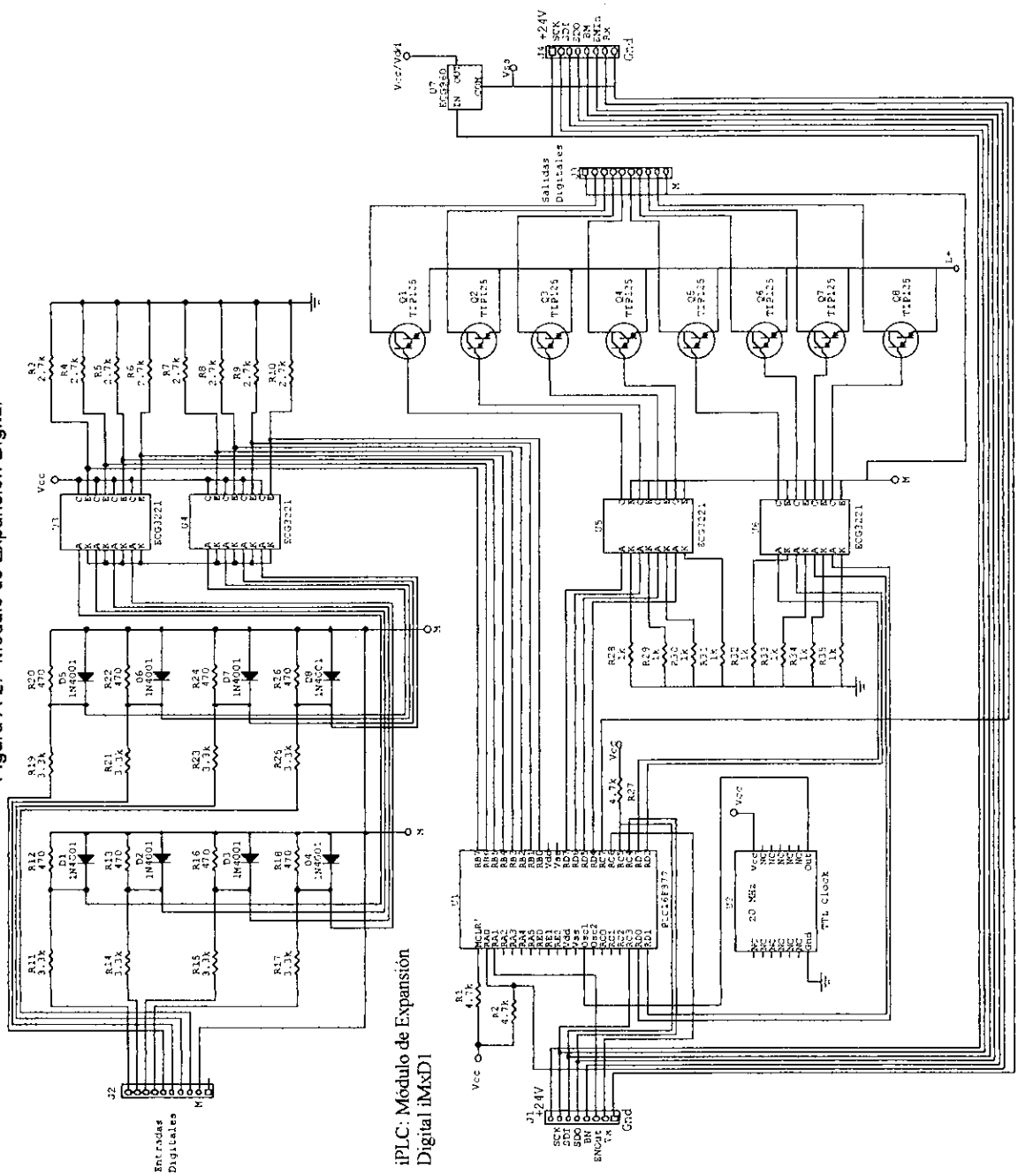


Figura A-2: "Módulo de Expansión Digital"

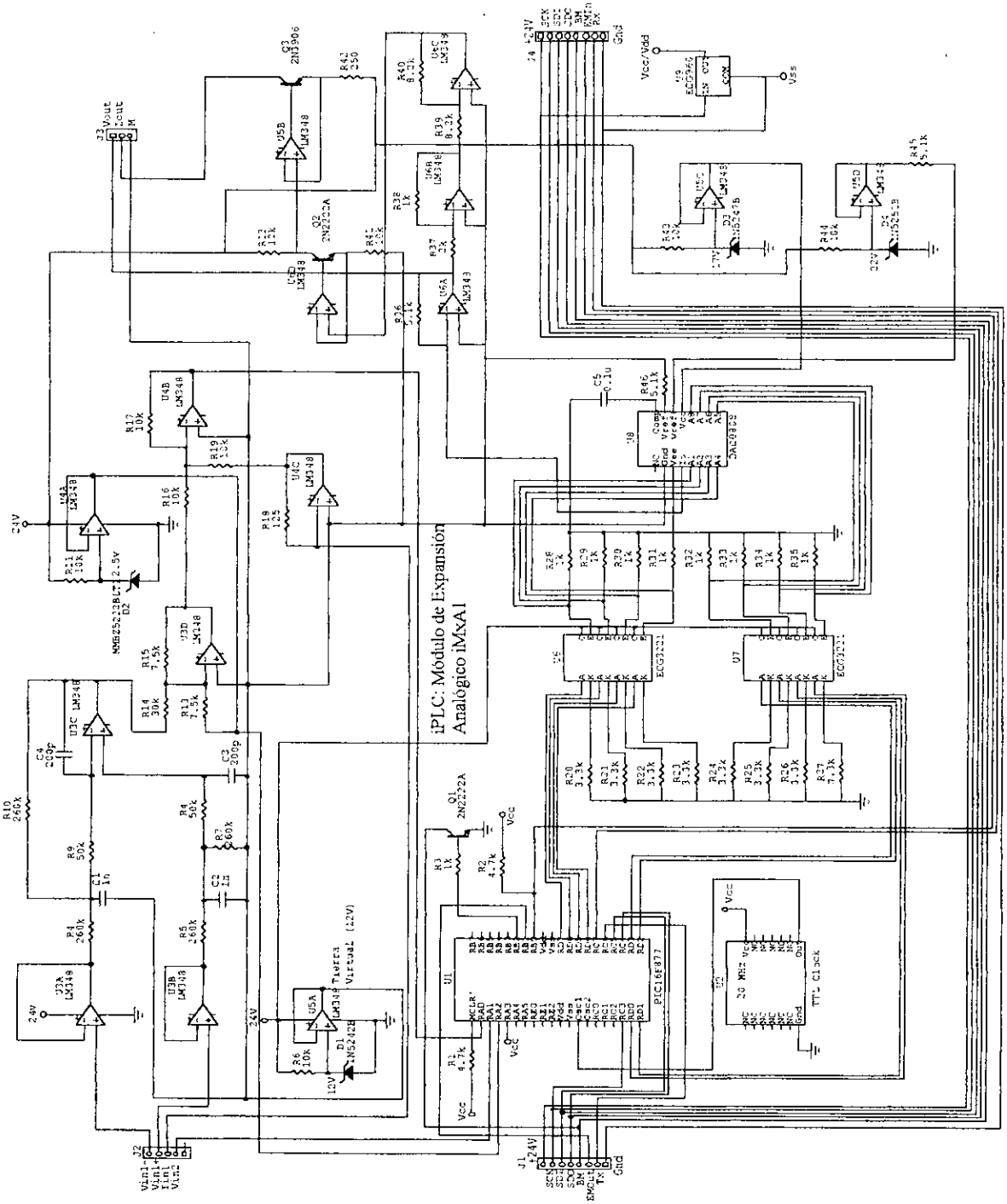


Figura A-3: "Módulo de Expansión Analógico"

APÉNDICE B

Archivo de definición del servidor HTTP

```
;******** Archivo de Definición para el Servidor HTTP del iPLC ********


;*** Sección de Definición ***

$DHCP on

$DeviceName "iPLC"

$DownloadPassword "Esquit"

$Include "C:\Archivos de Programa\SitePlayer\pcadefv2.inc"

$InitialIP "192.168.0.1"

$PostIRQ off

$SiteFile "C:\Mis Documentos\Carlos\TesisSP\Sys1.spb"

$SitePath "C:\Mis Documentos\Carlos\TesisSP\Sistema1"


;*** Sección de Objetos ***

                org 0h

PLC_Status      db 0
Trefresh dw ffffh


                org 010h

; Objetos de direcciones para las variables a monitorear.

a1              dd 0000ffffh    ; Se utilizan 16 bits para la dirección de la variable a monitorizar.
a2              dd 0000ffffh    ; Se utiliza un byte adicional para el tipo de dato a monitorizar.
a3              dd 0000ffffh
a4              dd 0000ffffh
a5              dd 0000ffffh
a6              dd 0000ffffh
a7              dd 0000ffffh
a8              dd 0000ffffh
a9              dd 0000ffffh
a10             dd 0000ffffh
a11             dd 0000ffffh
a12             dd 0000ffffh
a13             dd 0000ffffh
```

```
a14             dd 0000ffffh
a15             dd 0000ffffh
a16             dd 0000ffffh
a17             dd 0000ffffh
a18             dd 0000ffffh
a19             dd 0000ffffh
a20             dd 0000ffffh
a21             dd 0000ffffh
a22             dd 0000ffffh
a23             dd 0000ffffh
a24             dd 0000ffffh
a25             dd 0000ffffh
a26             dd 0000ffffh
a27             dd 0000ffffh
a28             dd 0000ffffh
a29             dd 0000ffffh
a30             dd 0000ffffh
a31             dd 0000ffffh
a32             dd 0000ffffh
a33             dd 0000ffffh
a34             dd 0000ffffh
a35             dd 0000ffffh
a36             dd 0000ffffh
a37             dd 0000ffffh
a38             dd 0000ffffh
a39             dd 0000ffffh
a40             dd 0000ffffh


                org 110h

; Objetos de datos para las variables a monitorear.

d1              dd 0h    ; Se utilizan 4 bytes para almacenar el dato de la variable de monitoreo.
d2              dd 0h
d3              dd 0h
d4              dd 0h
d5              dd 0h
d6              dd 0h
d7              dd 0h
d8              dd 0h
d9              dd 0h
d10             dd 0h
d11             dd 0h
d12             dd 0h
d13             dd 0h
d14             dd 0h
d15             dd 0h
```

```
d16             dd 0h
d17             dd 0h
d18             dd 0h
d19             dd 0h
d20             dd 0h
d21             dd 0h
d22             dd 0h
d23             dd 0h
d24             dd 0h
d25             dd 0h
d26             dd 0h
d27             dd 0h
d28             dd 0h
d29             dd 0h
d30             dd 0h
d31             dd 0h
d32             dd 0h
d33             dd 0h
d34             dd 0h
d35             dd 0h
d36             dd 0h
d37             dd 0h
d38             dd 0h
d39             dd 0h
d40             dd 0h


                org 1B0h

; Objetos de nombres para las variables a monitorear.

n1              ds 7    ; Todas las variables del Sistema pueden ser representadas con un máximo
n2              ds 7    ; de 7 caracteres.
n3              ds 7
n4              ds 7
n5              ds 7
n6              ds 7
n7              ds 7
n8              ds 7
n9              ds 7
n10             ds 7
n11             ds 7
n12             ds 7
n13             ds 7
n14             ds 7
n15             ds 7
n16             ds 7
n17             ds 7
```

```
n18             ds 7
n19             ds 7
n20             ds 7
n21             ds 7
n22             ds 7
n23             ds 7
n24             ds 7
n25             ds 7
n26             ds 7
n27             ds 7
n28             ds 7
n29             ds 7
n30             ds 7
n31             ds 7
n32             ds 7
n33             ds 7
n34             ds 7
n35             ds 7
n36             ds 7
n37             ds 7
n38             ds 7
n39             ds 7
n40             ds 7


                org 2D0h
; Tipos de módulos de expansión

emt1            db 0    ; Los Objetospara almacenar el código de identificación de los módulos
emt2            db 0    ; de expansión son de un byte.
emt3            db 0
emt4            db 0
emt5            db 0
emt6            db 0
emt7            db 0
emt8            db 0
emt9            db 0
emt10           db 0
emt11           db 0
emt12           db 0
emt13           db 0
emt14           db 0
emt15           db 0
```

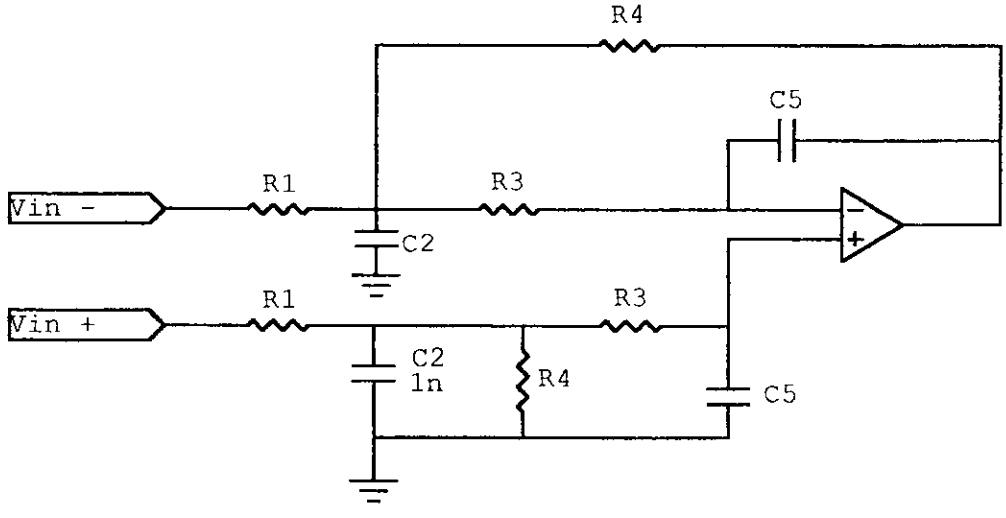
APÉNDICE C

Memoria de cálculos

A. Amplificador Diferencial, con filtro pasabajas, utilizado para la Entrada Diferencial del Módulo de Expansión Analógico

Considérese el circuito de la figura C-1, el cual se refiere a un amplificador diferencial, con filtro de entrada, propuesto como entrada diferencial para un DAC, según la nota de aplicación AN48 publicada por la empresa Cirrus Logic en marzo de 2003.

Figura C-1: "Filtro pasa bajas de entrada diferencial con dos polos".



Además, considérense las siguientes ecuaciones que describen el comportamiento del circuito de la figura C-1:

$$\xi = \alpha / \sqrt{(\alpha^2 + \beta^2)} \qquad \omega_0 = 2\pi F_c * \sqrt{(\alpha^2 + \beta^2)} \qquad K = C_5 / C_2$$

$R_1 = R_4 / (-H_0)$ $\qquad R_3 = \dfrac{1}{\omega_0 C_2 [\xi + \sqrt{(\xi^2 - K(1 - H_0))}]}$

$R_4 = \dfrac{\xi + \sqrt{(\xi^2 - K(1 - H_0))}}{\omega_0 C_5}$ donde H_0 es la ganancia de la banda pasante.

Entonces, eligiendo $H_0 = -1$, debido a la inversión del amplificador operacional, eligiendo $F_c = 3.1$ KHz, $C_2 = 1$nf, $C_5 = 200$pf y eligiendo un filtro tipo butterworth, la ubicación de los polos queda determinada por:

$\alpha = 0.7071$ y $\beta = 0.7071$

con lo que se tiene

$\xi = 1 / \sqrt{2}$,

$\omega_0 = 19477.87$,

$K = 0.2$

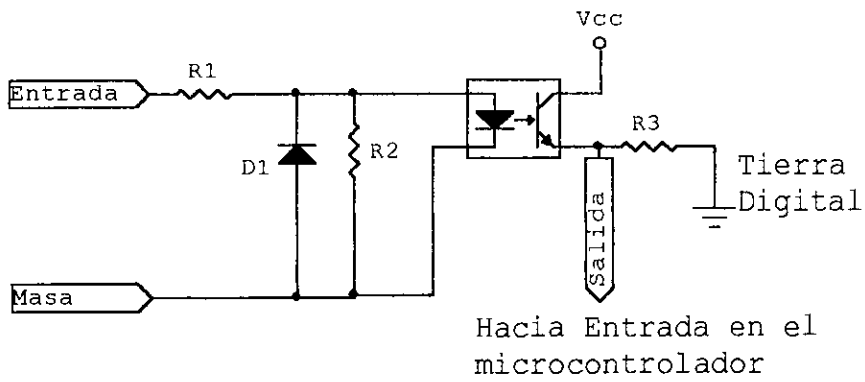
$R_4 = R_1 = 262$ KΩ

$R_3 = 50$ KΩ

B. Acoplamiento Óptico para las Entradas en el Módulo de Expansión Digital

Para el acoplamiento óptico se utilizó el ECG3221, que posee cuatro LED's y fototransistores apareados. Considérese el circuito de la figura C-2, utilizado para la conexión del acoplador óptico entre las terminales de entrada del iPLC y los bits de entrada del microcontrolador del módulo de expansión digital.

Figura C-2: " Circuito de Acoplamiento Óptico para las Entradas Digitales".



El voltaje nominal de entrada es de 24 V. El LED tiene una caída de 1.1 V, y la tensión $V_{CE\ (SAT)} = 0.3$ V máximo. Se desea calcular R_2 tal que el voltaje mínimo de entrada para el estado alto sea 15 V. Para el cálculo de este voltaje mínimo, se considerará que el voltaje de entrada en el microcontrolador no debe ser menor que 4.7 V, es decir, que el fototransistor debe estar saturado siempre que la entrada esté en alto.

Sean $R_1 = 3.3\ K\Omega$ y $R_3 = 2.7\ K\Omega$, entonces se tiene que:

$I_{R1} = (15V - 1.1V) / 3.3\ K\Omega = 4.21$ mA,

Puesto que la proporción de transferencia de corriente del ECG3221 es el 100%, para satisfacer $V_{R3} >= 4.7$ V, se necesita que $I_E = I_{LED} >= 4.7V / 2.7\ K\Omega = 1.74$ mA, así que:

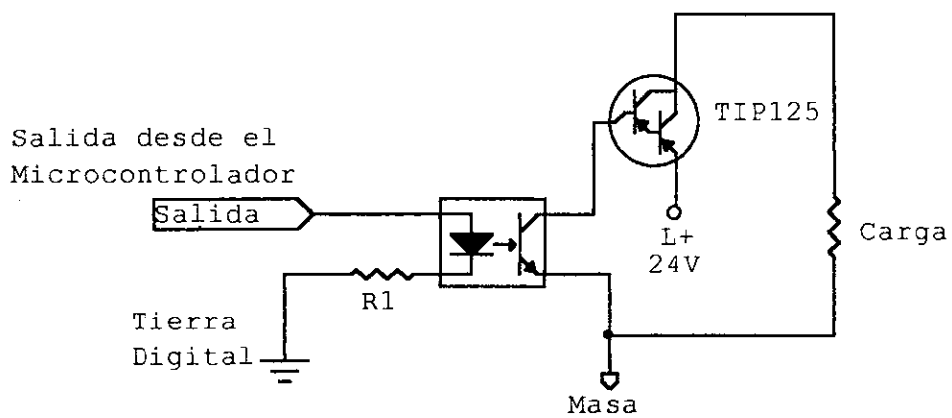
$I_{R2} >= 4.21$ mA – 1.74 mA = 2.47 mA,

De donde se tiene que $R_2 >= 1.1V / 2.47$ mA = 445.34 Ω (Se elige $R_2 = 470\ \Omega$).

C. Acoplamiento Óptico para las Salidas en el Módulo de Expansión Digital

Para este acoplamiento también se utilizó el ECG3221. Ahora considérese el circuito de la figura C-3, utilizado para acoplar las salidas del microcontrolador del módulo de expansión, con los transistores de salida de éste:

Figura C-3: "Circuito de Acoplamiento Óptico para las Salidas Digitales".



Se desea tener garantizar la capacidad de brindar hasta 4 A para una carga a 24 Vdc. El ECG3221 tiene una transferencia de corriente del 100%, y el voltaje mínimo de salida en alto ofrecido por el microcontrolador es de $V_{H\,min} = V_{CC} - 0.7V = 4.3\ V$. Además, el TIP125 posee una $hfe_{min} = 1000$, por lo que se tiene:

$I_C = I_{R1} >= 4\ A / 1000 = 4\ mA$,

Por lo que $R_1 <= 4.3\ V / 4\ mA = 1075\ \Omega$ (Se elige $R_1 = 1\ K\Omega$).

APÉNDICE D

Código fuente HTML y JavaScript de las páginas del servidor HTTP

****

Sistema de Software de alto nivel. Firmware del Entorno de Programación / Monitorización.

****

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Entorno de Programación del iPLC</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<META content="MSHTML 5.50.4134.600" name=GENERATOR></HEAD>
<BODY>
<SCRIPT language=JavaScript>

    <!-- ocultamiento para los browsers no compatibles

    var output_buffer=new Array
    var label_names=new Array //Pareja A
    var label_dirs=new Array     //Pareja A
    var label_request=new Array       //Pareja B
    var label_fill_dir=new Array  //Pareja B
    var i1,i2
    var k,j,e,f
    var int_const,float_const
    var b1,b2,b3,b4
    var sx,cx,ar1,ar2,ar3
    var id=new Array


    function Floats(in_const,float_const)
    {
        S=float_const<0?2:1

        int_const=Math.abs(float_const)
        float_const=parseInt(float_const)

        int_const=float_const<1?0:int_const

        nint=0
        flag=0
        e=-1

        if (int_const!=0)
        {
            nint=parseInt((Math.log(int_const)/Math.log(2))+1)
            e=nint-1
            flag=1
        }

        res=float_const-int_const+3

        M=int_const

        for (i=1; i<27-nint; i++)
        {
```

```
                if (res==0)
                {
                        resbit=0
                }
                else
                {
                        resbit=parseInt(res*2)
                        res=res*2-resbit
                )

                if (resbit==1)
                {
                        flag=1
                )

                if (flag==0)
                {
                        e--
                        i=0
                )

                M=M<<1|resbit
        }

        E=e+127

        b1=M&255
        b2=M>>>8&255
        b3=M>>>16&127|E<<7&64
        b4=E>>>1&127|S<<4

        output_buffer[k++]=b1
        output_buffer[k++]=b2
        output_buffer[k++]=b3
        output_buffer[k++]=b4

}


function Extract_Address(argx,str,inst)

{
        var
area_str,memory_area,data_str,data_element,memory_offset1,memory_offset2,memory_location,address_l
ow,address_hi
        var analog_element

        if (argx=='')
        {
                Instruction_Error(3,str)
                return
        }

        memory_offset1=0
```

```
memory_offset2=0

area_str=argx.substring(0,1)

data_str=argx.substring(1,2)

data_element=parseInt(argx.substring(2,argx.length))

switch(area_str)
{
      case 'D':
            if (data_element>15)
            {
                  Instruction_Error(4,str)
                  return
            }
            if (data_element==0)
            {
                  Instruction_Error(6,str)
                  return
            }
            if (data_str=='I')
            {
                  memory_offset1=31
                  break
            }
            if (data_str=='Q')
            {
                  memory_offset1=38
                  break
            }
            else
            {
                  Instruction_Error(4,str)
                  return
            )

      case 'C':
            data_str='W'
            data_element=parseInt(argx.substring(1,argx.length))

            if (data_element<40)
            {
                  memory_offset1=160

                  if ( (inst=='LD') || (inst=='LDN') || (inst=='A') || (inst=='AN') || (inst=='O') ||
(inst=='ON') )
                  {
                        memory_offset1++
                  }

                  break
            }
            else
            {
```

```
                    Instruction_Error(4,str)
                    return
                }

            case 'T':
                data_str='W'
                data_element=parseInt(argx.substring(1,argx.length))

                if (data_element<48)
                {
                    memory_offset1=272

                    if ( (inst=='LD') || (inst=='LDN') || (inst=='A') || (inst=='AN') || (inst=='O') ||
(inst=='ON') )
                    {
                        memory_offset1++
                    }

                    break
                }
                if (data_element>47 && data_element<96)
                {
                    memory_offset1=400

                    if ( (inst=='LD') || (inst=='LDN') || (inst=='A') || (inst=='AN') || (inst=='O') ||
(inst=='ON') )
                    {
                        memory_offset1++
                    }

                    data_element=data_element-48
                    break
                }
                else
                {
                    Instruction_Error(4,str)
                    return
                }

            case 'A':
                analog_element=parseInt(argx.substring(4,argx.length))
                if ( (isNaN(analog_element)) || (analog_element>7) )
                {
                    Instruction_Error(4,str)
                    return
                }

                if (data_str=='I')
                {
                    if (data_element==0)
                    {
                        Instruction_Error(6,str)
                        return
                    }
```

```
                if (data_element>15)
                {
                        Instruction_Error(4,str)
                        return
                }

                memory_offset1=512
                data_element=data_element*16+analog_element;
                break
        }

        if (data_str=='Q')
        {
                if (data_element==0)
                {
                        Instruction_Error(6,str)
                        return
                }
                if (data_element>15)
                {
                        Instruction_Error(4,str)
                        return
                }

                memory_offset1=768
                data_element=data_element*16+analog_element;
                break
        }
        else
        {
                Instruction_Error(4,str)
                return
        }

    case 'V':
        memory_offset1=1024
        break
    case 'S':
        memory_offset1=106
        break
    case 'R':
        memory_offset1=32211
        data_str='B'
        data_element=parseInt(argx.substring(1,argx.length))
        if (data_element>12)
        {
                Instruction_Error(4,str)
        }
        break
    default:
        Instruction_Error(4,str)
        return
}
```

```
        switch(data_str)
        {
              case 'I':
                    memory_offset2=data_element*1
                    break
              case 'Q':
                    memory_offset2=data_element*1
                    break
              case 'B':
                    memory_offset2=data_element*1
                    break
              case 'W':
                    memory_offset2=data_element*2
                    break
              case 'D':
                    memory_offset2=data_element*4
                    break
              case 'M':
                    memory_offset2=data_element*1
                    break
              default:
                    Instruction_Error(5,str)
                    return
        }

        if (memory_offset2>28000)
        {
              Instruction_Error(4,str)
              return
        }

        memory_location=memory_offset1+memory_offset2

        address_hi=parseInt(memory_location/64)
        address_low=((memory_location/64)-address_hi)*127

        output_buffer[k++]=address_low
        output_buffer[k++]=address_hi

}


function Extract_Bit(argx,str)
{
        var bit_index,bit_num,find_char

        if ( (argx.substring(0,1)=='C') || (argx.substring(0,1)=='T') )
        {
              bit_num=7
              output_buffer[k++]=bit_num
              return
        }


        find_char=String.fromCharCode(46)
```

```
        bit_index=argx.indexOf(find_char,0)

        if (bit_index<0)
        {
              Instruction_Error(3,str)
              return
        )

        bit_num=parseInt(argx.substring(bit_index+1,argx.length))

        if (bit_num>7)
        (
              Instruction_Error(4,str)
              return
        }

        output_buffer[k++]=bit_num


}       // Fin de la función "Extract_Bit".


function Detect_Constant(st,argx,type,coded,f)
{
        int_const=parseInt(argx.toLowerCase())
        float_const=parseFloat(argx.toLowerCase())

        val=Math.abs(int_const)


        if ( !isNaN(int_const) && !isNaN(float_const) )
        {

              if ((float_const!=int_const)&&type!=4&&(argx.substring(0,2)!='0X'))
              {
                    Instruction_Error(5,st)
                    return 4
              }

              switch(type)
              (
                    case 0:
                          Instruction_Error(5,st)

                    case 1:
                          if (val>255)
                                Instruction_Error(5,st)

                    case 2:
                          if (val>65535)
                                Instruction_Error(5,st)

                    case 3:
                          if (val>4294967295)
```

```
                        Instruction_Error(5,st)
        default:
            break
}

if ( (float_const==int_const||argx.substring(0,2)=='0X') && type!=4)
{
    switch(type)
    {
        case 1:
            b1=int_const
            output_buffer[k++]=coded+128
            output_buffer[k++]=b1
            return 1
        case 2:
            b2=parseInt(int_const/256)
            b1=(int_const/256-b2)*256
            if (int_const>=0) b2|=128
            else b2&=127

            if (cx=='CTU'||cx=='TONR')
            {
                b2&=127
                output_buffer[k++]=coded+128
                output_buffer[k++]=b1
                output_buffer[k++]=b2
                return 2
            }

            output_buffer[k++]=coded+128
            output_buffer[k++]=b1
            output_buffer[k++]=b2
            return 2
        case 3:
            b3=parseInt(int_const/65536)
            b2=parseInt( (int_const-b3*65536)/256)
            b1=int_const-b3*65536-b2*256
            if (int_const>=0) b3|=128
            else b3&=127

            output_buffer[k++]=coded+128
            output_buffer[k++]=b1
            output_buffer[k++]=b2
            output_buffer[k++]=b3
            output_buffer[k++]=0
            return 3
        default:
            return 5
    }
}
else
{   // Se generan los bytes para la constante real.

    output_buffer[k++]=coded+128
```

```
                    Floats(int_const,float_const)

                    return 4
                }

            }

            output_buffer[k++]=coded

            Extract_Address(argx,st,f)

            if ( (f=='LD') || (f=='LDN') || (f=='A') || (f=='AN') || (f=='O') || (f=='ON') || (f=='=') || (f=='SET_BITS') ||
(f=='RESET_BITS') )
            {
                Extract_Bit(argx,st)
            }

            return 0

        }


function Encode_Instruction(s,c,a1,a2,a3)
{
        sx=s
        cx=c
        ar1=a1
        ar2=a2
        ar3=a3

switch(c)
{
case '.':
output_buffer[k++]=0
break
case 'LD':
X(1,1,0)
break
case 'A':
X(2,1,0)
break
case 'O':
X(3,1,0)
break
case 'LDN':
X(4,1,0)
break
case 'AN':
X(5,1,0)
break
case 'ON':
X(6,1,0)
break
case 'NOT':
```

```
X(7,0,0)
break
case 'EU':
X(8,0,0)
break
case 'ED':
X(9,0,0)
break
case 'ALD':
X(10,0,0)
break
case 'OLD':
X(11,0,0)
break
case 'LPS':
X(12,0,0)
break
case 'LRD':
X(13,0,0)
break
case 'LPP':
X(14,0,0)
break
case '=':
X(15,1,0)
break
case 'SET_BITS':
X(16,2,1)
break
case 'RESET_BITS':
X(17,2,1)
break
case 'NOTH':
X(18,0,0)
break
case 'LDB=':
X(19,2,1)
break
case 'LDW=':
X(20,2,2)
break
case 'LDD=':
X(21,2,3)
break
case 'LDR=':
X(22,2,4)
break
case 'LDB>':
X(23,2,1)
break
case 'LDW>':
X(24,2,2)
Break
```

```
case 'LDD>':
X(25,2,3)
break
case 'LDR>':
X(26,2,4)
break
case 'LDB<':
X(27,2,1)
break
case 'LDW<':
X(28,2,2)
break
case 'LDD<':
X(29,2,3)
break
case 'LDR<':
X(30,2,4)
break
case 'BTI':
X(31,2,0)
break
case 'BTD':
X(32,2,0)
break
case 'BTR':
X(33,2,0)
break
case 'ITD':
X(34,2,0)
break
case 'ITR':
X(35,2,0)
break
case 'DTI':
X(36,2,0)
break
case 'DTR':
X(37,2,0)
break
case 'RTB':
X(38,2,0)
break
case 'RTI':
X(39,2,0)
break
case 'RTD':
X(40,2,0)
break
case 'CTU':
X(41,2,2)
break
case 'TONR':
X(42,2,2)
Break
```

```
case 'TRESET':
X(43,1,1)
break
case 'INVB':
X(44,2,0)
break
case 'INVW':
X(45,2,0)
break
case 'INVD':
X(46,2,0)
break
case 'ANDB':
X(47,3,1)
break
case 'ANDW':
X(48,3,2)
break
case 'ANDD':
X(49,3,3)
break
case 'ORB':
X(50,3,1)
break
case 'ORW':
X(51,3,2)
break
case 'ORD':
X(52,3,3)
break
case 'XORB':
X(53,3,1)
break
case 'XORW':
X(54,3,2)
break
case 'XORD':
X(55,3,3)
break
case 'MOVB':
X(56,2,1)
break
case 'MOVW':
X(57,2,2)
break
case 'MOVD':
X(58,2,3)
break
case 'MOVR':
X(59,2,4)
break
case 'SHLB':
X(60,3,1)
Break
```

```
case 'SHRB':
X(61,3,1)
break
case 'SHLW':
X(62,3,1)
break
case 'SHRW':
X(63,3,1)
break
case 'SHLD':
X(64,3,1)
break
case 'SHRD':
X(65,3,1)
break
case 'RLB':
X(66,3,1)
break
case 'RR8':
X(67,3,1)
break
case 'RLW':
X(68,3,1)
break
case 'RRW':
X(69,3,1)
break
case 'RLD':
X(70,3,1)
break
case 'RRD':
X(71,3,1)
break
case 'INCB':
X(72,1,0)
break
case 'DECB':
X(73,1,0)
break
case 'INCW':
X(74,1,0)
break
case 'DECW':
X(75,1,0)
break
case 'INCD':
X(76,1,0)
break
case 'DECD':
X(77,1,0)
break
case 'ADD':
X(78,3,4)
```

```
 break
 case 'SUB':
 X(79,3,4)
 break
 case 'MULT':
 X(80,3,4)
 break
 case 'DIV':
 X(81,3,4)
 break
 case 'CHS':
 X(82,2,4)
 break
 case 'ABS':
 X(83,2,4)
 break
 case 'SQRT':
 X(84,2,4)
 break
 case 'LOG':
 X(85,2,4)
 break
 case 'LOG10':
 X(86,2,4)
 break
 case 'EXP':
 X(87,2,4)
 break
 case 'EXP10':
 X(88,2,4)
 break
 case 'POW':
 X(89,3,4)
 break
 case 'ROOT':
 X(90,3,4)
 break
 case 'RECIP':
 X(91,2,4)
 break
 case 'SIN':
 X(92,2,4)
 break
 case 'ASIN':
 X(93,2,4)
 break
 case 'COS':
 X(94,2,4)
 break
 case 'ACOS':
 X(95,2,4)
 break
 case 'TAN':
 X(96,2,4)
```

```
break
case 'ATAN':
X(97,2,4)
break
case 'JMP':
output_buffer[k++]=98
label_request[e]=a1
label_fill_dir[e++]=k
k+=2
break
case 'CALLSBR':
output_buffer[k++]=99
label_request[e]=a1
label_fill_dir[e++]=k
k+=2
break
case 'LABEL':
label_names[f]=a1
label_dirs[f++]=k
break
case 'SBR':
label_names[f]=a1
label_dirs[f++]=k
break
case 'RET':
output_buffer[k++]=100
break
case 'CRET':
output_buffer[k++]=101
break
case 'ENDP':
output_buffer[k++]=102
break
case 'STOP':
output_buffer[k++]=103
break
case 'WDTR':
output_buffer[k++]=104
break
case 'MEND':
output_buffer[k++]=126
break
case 'RTCRD':
output_buffer[k++]=105
break
case 'RTCWR':
output_buffer[k++]=106
break

default:
      Instruction_Error(1,c)
      return
}
```

```
}

    function X(xparam,code,const_type)
    {
        switch(xparam)
        {
            case 0:
                if (ar1!=''||ar2!=''||ar3!='')
                {
                    Instruction_Error(2,sx)
                    return
                }
                else
                {
                    output_buffer[k++]=code
                    retum
                }


            case 1:
                if (ar2!=''||ar3!='')
                {
                    Instruction_Error(2,sx)
                    return
                }
                else
                {
                    arg_type=Detect_Constant(sx,ar1,const_type,code,cx)
                    return
                }


            case 2:
                if (ar3!='')
                (
                    Instruction_Error(2,sx)
                    return
                }
                else
                {
                    if (cx=='CTU'||cx=='TONR')
                    {
                        arg_type=Detect_Constant(sx,ar2,const_type,code,cx)

                        tcnum=parseInt(ar1)
                        output_buffer[k++]=tcnum

                        if (cx=='CTU'&&tcnum>39)
                        {
                            Instruction_Error(4,sx)
                            return
                        }

                        if (cx=='TONR'&&tcnum>95)
```

```
                {
                    Instruction_Error(4,sx)
                    return
                }

                return
            }

            arg_type=Detect_Constant(sx,ar1,const_type,code,cx)

            Extract_Address(ar2,sx,cx)

            if ( (cx=='SET_BITS') || (cx=='RESET_BITS') )
            {
                Extract_Bit(ar2,sx)
            }

            return
        }


    case 3:
        arg_type=Detect_Constant(sx,ar1,const_type,code,cx)

        Extract_Address(ar2,sx,cx)

        Extract_Address(ar3,sx,cx)

        return


    default:
        break
  )

}



function Analize(form)
{
    var instructions_strs=new Array
    var p
    var i,Cadena,Clength,start_position,end_position,instruction_str_index,instruction_index,find_str
    var instruction_str,instruction_code_str,arg1_str,arg2_str,arg3_str,arg1,arg2,arg3
    var arg1_index,arg2_index,arg3_index
    var args_qty

    label_names.length=0
    label_dirs.length=0
    label_request.length=0
    label_fill_dir.length=0
```

```
cadena=form.source_code.value
cadena=cadena.toUpperCase()

k=0
p=0
e=0
f=0

start_position=0
end_position=0

form.valid_program.value=1

clength=cadena.length

instruction_str='x'
i=0

top_index=1

while (top_index<clength-1)
{
      find_str=String.fromCharCode(13)

      top_index=cadena.indexOf(find_str,start_position)

      if (top_index<0)
      {
            break
      }

      instruction_str=cadena.substring(start_position,top_index)

      start_position=top_index+2

      find_str=String.fromCharCode(32)

      top_index=instruction_str.indexOf(find_str,0)

      if (top_index<0)
      {
            instruction_code_str=instruction_str
            args_qty=0
            instructions_strs[p++]=instruction_code_str

            Encode_Instruction(instruction_str,instruction_code_str,",",")

            continue
      }

      else
      {
            instruction_code_str=instruction_str.substring(0,top_index)
            instructions_strs[p++]=instruction_code_str
```

```
}

find_str=String.fromCharCode(44)

arg1_index=instruction_str.indexOf(find_str,top_index)

if (arg1_index<0)
{
      arg1_str=instruction_str.substring(top_index+1,instruction_str.length)
      args_qty=1
      instructions_strs[p++]=arg1_str

      Encode_Instruction(instruction_str,instruction_code_str,arg1_str,",")

      continue
}

else
{
      arg1_str=instruction_str.substring(top_index+1,arg1_index)
      args_qty=2
      instructions_strs[p++]=arg1_str

      arg2_index=instruction_str.indexOf(find_str,arg1_index+1)

      if (arg2_index<0)
      {
            arg2_str=instruction_str.substring(arg1_index+1,instruction_str.length)
            args_qty=2
            instructions_strs[p++]=arg2_str

            Encode_Instruction(instruction_str,instruction_code_str,arg1_str,arg2_str,")

            continue
      }

      else
      {
            arg2_str=instruction_str.substring(arg1_index+1,arg2_index)
            args_qty=3
            instructions_strs[p++]=arg2_str

            arg3_index=instruction_str.indexOf(find_str,arg2_index+1)

            if (arg3_index<0)
            {
                  arg3_str=instruction_str.substring(arg2_index+1,instruction_str.length)
                  args_qty=3
                  instructions_strs[p++]=arg3_str

Encode_Instruction(instruction_str,instruction_code_str,arg1_str,arg2_str,arg3_str)

                  Continue
```

```
                        }
                        else
                        {
                                form.valid_program.value=0
                                Instruction_Error(2,instruction_str)
                                return
                        }


                }


        }


)       //Fin del ciclo While.

// Aquí va el código de las instrucciones JMP y CALLSBR.

for (xt=0; xt<label_request.length; xt++)
{
        lbl_flag=0

        for (yt=0; yt<label_names.length; yt++)
        {
                if (label_names[yt]==label_request[xt])
                {
                        lbl_flag=1
                        ad=label_dirs[xt] //NO se resta uno debido a que el contenido de la posición 0
del buffer de salida queda
                        ad_hi=parseInt(yt/256)// grabado en la posición 1 del IC FLASH.
                        ad_low=((yt/256)-ad_hi)*256

                        output_buffer[label_fill_dir[xt]]=ad_low
                        output_buffer[label_fill_dir[xt]+1]=ad_hi

                        if (label_names[yt]=='S1')
                        {
                                alert('K='+label_fill_dir[xt]+' : '+ad_low)
                                alert('K='+label_fill_dir[xt]+1+' : '+ad_hi)
                        }

                }
        }

        if (lbl_flag==0)
        {
                Instruction_Error(7,label_request[xt])
        }

}

}
```

```
function Instruction_Error(error_number,inst_txt)
{
	document.form1.valid_program.value=0

	switch(error_number)
	{
		case 1:
			alert("Instrucción NO Válida: \n"+inst_txt); break
		case 2:
			alert("Demasiados Parámetros en la Instrucción: \n"+inst_txt); break
		case 3:
			alert("Faltan Parámetros en la Instrucción: \n"+inst_txt); break
		case 4:
			alert("Área de Memoria NO Válida: \n"+inst_txt); break
		case 5:
			alert("Tipo de Dato NO Válido: \n"+inst_txt); break
		case 6:
			alert("Área de Memoria Reservada: \n"+inst_txt); break
		case 7:
			alert("No se encontró la etiqueta '"+inst_txt+"'"); break
		default:
			alert("Error NO Definido.")
	}

}


function Txt_Erase(form)
{
	form.source_code.value=""
}


function PROGRAM(form)
{
	if (form.valid_program.value==0)
	{
		alert("El Programa Ingresado Tiene Errores.  No es Posible Programar el iPLC.")
		return
	}
	else
	{
		if (k==0)
		{
			alert("No se Dispone de un Código Compilado para ser Cargado en el iPLC.")
			return
		}
		j=0
		last_rx_ack=document.ack_form.rx_ack.value+1
		Send_data_to_the_iPLC()
		return
	}

}
```

```
function ERASE(form)
{
	document.all['ack'].value=1024;

}


function Send_data_to_the_iPLC()
{

	document.out_form.com.value=output_buffer[j++]
	document.out_form.submit()
	document.ack_form.num.value=j-1


	if (j<k)
	{
		id[k-2]=setTimeout("Send_data_to_the_iPLC();",document.ack_form.delay.value)

	}
	else
	{
		alert("¡El iPLC ha sido programado satisfactoriamente!")
		return
	}

}


// -- Fin del ocultamiento para los browsers no compatibles con JavaScript-->


</SCRIPT>


<H2>Página del Compilador del iPLC </H2> Carlos A. Esquit H.
<A href="Sys1spi.spi?io6=1&io5=0"> <H3>Activar el modo ERASE</H3></A>
<A href="Sys1spi.spi?io6=1&io5=1"> <H3>Activar el modo PROG</H3></A>
<A href="Sys1spi.spi?io6=0&io5=0"> <H3>Activar el modo RUN</H3></A>
<A href="Sys1spi.spi?io6=0&io5=1"> <H3>Activar el modo STOP</H3></A>

<FORM name=out_form action=Sys1spi.spi method=get>
	<INPUT type=text name=com>
</FORM>

<FORM name=ack_form action=Sys1spi.spi method=get>
	<INPUT type=text name=rx_ack value=^rx_ack id=ack>
	<INPUT type=text name=num value=0>
	Delay del buffer de salida: <INPUT name=delay value=50>
</FORM>

<FORM name=form1 action=Sys1spi.spi method=get>
	<TEXTAREA name=source_code rows=20 cols=120> </TEXTAREA> <BR><BR>
	<INPUT onclick=Analize(this.form) type=button value="Compilar">
	<INPUT onclick=Txt_Erase(this.form) type=button value="Borrar el Código ">
```

```
      <INPUT type=hidden name=valid_program>
      <INPUT onclick=PROGRAM(this.form) type=button value="Programar el iPLC">
      <INPUT onclick=ERASE(this.form) type=button value="Borrar el iPLC">
</FORM>

</BODY>

</HTML>



<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Entorno de Monitoreo del iPLC</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<META content="MSHTML 5.50.4134.600" name=GENERATOR>
<meta http-equiv="refresh" content=^Trefresh>

</HEAD>

<BODY>
<SCRIPT language=JavaScript>

      var k,flag


      function Extract_Address(argx)    // Esta función extrae la dirección de el parámetro recibido, según el área de memoria
                                        // y el byte al que pertenece.
      {
            var area_str,memory_area,data_str,data_element,memory_offset1,memory_offset2,memory_location,address_low,address_hi
            var analog_element

            argx=argx.toUpperCase()

            if (argx=='')
            {
                  document.addresses.elements[k++].value=65535  // Se carga la localidad de memoria a monitorear en el objeto SP.
                  return
            }

            memory_offset1=0
            memory_offset2=0

            area_str=argx.substring(0,1)// Se extrae el código de área de memoria.

            data_str=argx.substring(1,2)// Se extrae la cadena del número de elemento a acceder en el área de memoria.

            data_element=parseInt(argx.substring(2,argx.length))
```

```
switch(area_str) // Se asignará el primer offset según el área de memoria a acceder.
{
      case 'D':
            if (data_element>15)
            {
                  Instruction_Error(4,argx)     // Área de memoria NO válida.
                  return
            }
            if (data_element==0)
            {
                  Instruction_Error(6,argx)     // Área de memoria reservada.
                  return
            }
            if (data_str=='I')
            {
                  memory_offset1=31    // offset de 0x1F para el área de módulos de entradas
digitales (empiezan en 1).
                  break
            }
            if (data_str=='Q')
            {
                  memory_offset1=38    // offset de 0x26 para el área de módulos de salidas
digitales (empiezan en 1).
                  break
            }
            else
            {
                  Instruction_Error(4,argx)     // El área de memoria NO es válida.
                  return
            }

      case 'C':
            data_str='W'      // Los Contadores serán accedidos únicamente como Words.
            data_element=parsefloat(argx.substring(1,argx.length))

            if (data_element<40)
            {
                  memory_offset1=160  // offset de 0xA0 para el área de Contadores (empiezan
en 0).
                  break
            }
            else
            {
                  Instruction_Error(4,argx)     // Área de memoria NO válida.
                  return
            }

      case 'T':
            data_str='W'      // Los Temporizadores serán accedidos únicamente como Words.
            data_element=parseInt(argx.substring(1,argx.length))

            if (data_element<48)
            {
```

```
                        memory_offset1=272  // offset de 0x110 para el área de Temporizadores (del
T0 al T47).
                        break
                }
                if (data_element>47 && data_element<96)
                {
                        memory_offset1=400  // offset de 0x190 para el área de Temporizadores (del
T48 al T95).
                        data_element=data_element-48   // Se debe restar la cantidad de
Temporizadores que hay en el bloque #1.
                        break
                }
                else
                {
                        Instruction_Error(4,argx) // Área de memoria NO válida.
                        return
                )

        case 'A':
                analog_element=parseInt(argx.substring(4,argx.length))
                if ( (isNaN(analog_element)) || (analog_element>7) )      // Se determina si el
elemento analógico es válido.
                {
                        Instruction_Error(4,argx)
                        return
                }

                if (data_str=='I')
                {
                        if (data_element==0)
                        {
                                Instruction_Error(6,argx)
                                return
                        )
                        if (data_element>15)
                        {
                                Instruction_Error(4,argx)
                                return
                        }

                        memory_offset1=512  // offset de 0x200 para el área de entradas analógicas
(empiezan en 1).
                        data_element=data_element*16+analog_element;// El # de módulo de exp.
está dado por el 2do. dígito hexadecimal.
                        break
                }

                if (data_str=='Q')
                {
                        if (data_element==0)
                        {
                                Instruction_Error(6,argx)
                                return
                        }
```

```
                        if (data_element>15)
                        {
                            Instruction_Error(4,argx)
                            return
                        }

                        memory_offset1=768  // offset de 0x300 para el área de salidas analógicas
(empiezan en 1).
                        data_element=data_element*16+analog_element;// El # de módulo de exp.
está dado por el 2do. dígito hexadecimal.
                        break
                    }
                    else
                    {
                        Instruction_Error(4,argx)     // El área de memoria NO es válida.
                        return
                    }

                case 'V':
                    memory_offset1=1024// offset de 0x400 para el área de memoria V.
                    break

                case 'S':
                    memory_offset1=106  // offset de 0x6A para el área de memoria de marcas
especiales.
                    break
                case 'R':
                    memory_offset1=32211
                    data_str='B'
                    data_element=parseInt(argx.substring(1,argx.length))
                    if (data_element>12)
                    {
                        Instruction_Error(4,argx)
                    }
                    break
                default:
                    Instruction_Error(4,argx)     // Área de memoria NO válida.
                    return
            }


            switch(data_str) // Se asignará un segundo offset según el # de elemento a acceder en el área
de memoria seleccionada.
            {
                case 'I':
                    off2=data_element*1
                    memory_offset2=data_element*1+6536// El multiplicador es uno (1) para los
elementos tipo Byte.
                    break                         // y el código del byte # 3 es 6536 (1).
                case 'Q':
                    off2=data_element*1
                    memory_offset2=data_element*1+6536// El multiplicador es uno (1) para los
elementos tipo Byte.
                    Break
```

```
                case 'B':
                    off2=data_element*1
                    memory_offset2=data_element*1+6536// El multiplicador es uno (1) para los
elementos tipo Byte.
                    break
                case 'W':
                    off2=data_element*2
                    memory_offset2=data_element*2+11072     // El multiplicador es dos (2) para los
elementos tipo Word.
                    break                       // y el código del byte # 3 es 11072 (2).
                case 'D':
                    off2=data_element*4
                    memory_offset2=data_element*4+22144     // El multiplicador es cuatro (4) para los
elementos tipo Double.
                    break                       // y el código del byte # 3 es 22144 (4).
                case 'R':
                    off2=data_element*4
                    memory_offset2=data_element*4+22144     // El multiplicador es cuatro (4) para los
elementos tipo Real.
                    break                       // y el código del byte # 3 es 22144 (4).
                case 'M':
                    off2=data_element*1
                    memory_offset2=data_element*1+6536
                    break
                default:
                    Instruction_Error(5,argx)   // Tipo de dato NO válido.
                    return
            }

            if (off2>28000)
            {
                Instruction_Error(4,argx)   // Área de memoria NO válida.
                return
            }

            memory_location=memory_offset1+memory_offset2     // Localidad de memoria a acceder
(está en valor decimal, entre 0 y 29,024).

            document.addresses.elements[k].value=memory_location     // Se carga la localidad de
memoria a monitorear en el objeto SP.
                                                // (dentro del formulario de la página Web).

        } // Fin de la función "Extract_Address".



        function Instruction_Error(error_number,inst_txt)   // Función que notifica los errores de sintaxis.
        {
            flag=1
            switch(error_number)
            {
                case 1:
```

```
                alert("Instrucción NO Válida: \n"+inst_txt); break
            case 2:
                alert("Demasiados Parámetros en la Instrucción: \n"+inst_txt); break
            case 3:
                alert("Faltan Parámetros en la Instrucción: \n"+inst_txt); break
            case 4:
                alert("Área de Memoria NO Válida: \n"+inst_txt); break
            case 5:
                alert("Tipo de Dato NO Válido: \n"+inst_txt); break
            case 6:
                alert("Área de Memoria Reservada: \n"+inst_txt); break
            default:
                alert("Error NO Definido.")
        }

)       // Fin de la función "Instruction_Error".



function Monitor()
{
        k=0   // Se inicializa el contador de objetos de direcciones de variables a monitorizar.
        flag=0

        tm=parseInt(document.controls.Tmonitor.value)
        if ( isNaN(tm) || tm=='' || tm==0)
        {
                alert("Debe Ingresar un dato Válido para el Periodo de Monitorización")
                retum
        )

        for (i=0; i<document.names.length; i+=2)
        {
                Extract_Address(document.names.elements[i].value)
        }

        if (flag==0)
        {
                document.addresses.Trefresh.value=tm
                document.names.submit()

                id=setTimeout("Send_Addresses_to_the_iPLC();",100)
        )
        else
        {
                alert("No se han Ingresado Variables Válidas para la Monitorización.")
        }

}       // Fin de la función "Monitor".
```

```
function Stop_Monitor()
{
	document.stop.Trefresh.value=65000
	document.stop.submit()
}



function Send_Addresses_to_the_iPLC()
{
	document.addresses.submit()

}	// Fin de la función "Send Addresses to the iPLC".



function Reals(str)
{
	var min=0.5/(Math.pow(2,22))

	for (j=0; j<str.length; j+=2)
	{
		var M=0

		if (str.elements[j].value.substring(1,2).toUpperCase()=='R')
		(
			IEEE754=parseInt(str.elements[j+1].value)
			S=IEEE754>>>31
			E=(IEEE754>>>23)&255

			for (h=0; h<23; h++)
			{
				xbit=(IEEE754>>>h)&1
				M=M+min*Math.pow(2,h)*xbit
			}
			val=Math.pow(-1,S)*Math.pow(2,(E-127))*(1+M)

			str.elements[j+1].value=val
		}

		if (str.elements[j].value=='')
		{
			str.elements[j+1].value=0
		}
	}

}	// Fin de la función "Reals".

</SCRIPT>
```

```
<H1>Página de Monitoreo del iPLC</H1><H3> <br> Carlos A. Esquit H. <br> </H3>

<div align=center>

<FORM name=names action=mon1spi.spi method=get>
<table border="15" cellpadding="1" cellspacing="3" width="50">

      <tr><td><INPUT size=9 name=n1 value=^n1></td><td><INPUT
value=^d1></td><td>  </td><td><INPUT size=9 name=n2 value=^n2></td><td><INPUT
value=^d2></td>
      <td>  </td><td><INPUT size=9 name=n3 value=^n3></td><td><INPUT
value=^d3></td><td>  </td><td><INPUT size=9 name=n4 value=^n4></td>
      <td><INPUT value=^d4></td></tr>
      <tr><td><INPUT size=9 name=n5 value=^n5></td><td><INPUT
value=^d5></td><td>  </td><td><INPUT size=9 name=n6 value=^n6></td><td><INPUT
value=^d6></td>
      <td>  </td><td><INPUT size=9 name=n7 value=^n7></td><td><INPUT
value=^d7></td><td>  </td><td><INPUT size=9 name=n8 value=^n8></td>
      <td><INPUT value=^d8></td></tr>
      <tr><td><INPUT size=9 name=n9 value=^n9></td><td><INPUT
value=^d9></td><td>  </td><td><INPUT size=9 name=n10 value=^n10></td><td><INPUT
value=^d10></td>
      <td>  </td><td><INPUT size=9 name=n11 value=^n11></td><td><INPUT
value=^d11></td><td>  </td><td><INPUT size=9 name=n12 value=^n12></td>
      <td><INPUT value=^d12></td></tr>
      <tr><td><INPUT size=9 name=n13 value=^n13></td><td><INPUT
value=^d13></td><td>  </td><td><INPUT size=9 name=n14 value=^n14></td><td><INPUT
value=^d14></td>
      <td>  </td><td><INPUT size=9 name=n15 value=^n15></td><td><INPUT
value=^d15></td><td>  </td><td><INPUT size=9 name=n16 value=^n16></td>
      <td><INPUT value=^d16></td></tr>
      <tr><td><INPUT size=9 name=n17 value=^n17></td><td><INPUT
value=^d17></td><td>  </td><td><INPUT size=9 name=n18 value=^n18></td><td><INPUT
value=^d18></td>
      <td>  </td><td><INPUT size=9 name=n19 value=^n19></td><td><INPUT
value=^d19></td><td>  </td><td><INPUT size=9 name=n20 value=^n20></td>
      <td><INPUT value=^d20></td></tr>
      <tr><td><INPUT size=9 name=n21 value=^n21></td><td><INPUT
value=^d21></td><td>  </td><td><INPUT size=9 name=n22 value=^n22></td><td><INPUT
value=^d22></td>
      <td>  </td><td><INPUT size=9 name=n23 value=^n23></td><td><INPUT
value=^d23></td><td>  </td><td><INPUT size=9 name=n24 value=^n24></td>
      <td><INPUT value=^d24></td></tr>
      <tr><td><INPUT size=9 name=n25 value=^n25></td><td><INPUT
value=^d25></td><td>  </td><td><INPUT size=9 name=n26 value=^n26></td><td><INPUT
value=^d26></td>
      <td>  </td><td><INPUT size=9 name=n27 value=^n27></td><td><INPUT
value=^d27></td><td>  </td><td><INPUT size=9 name=n28 value=^n28></td>
      <td><INPUT value=^d28></td></tr>
      <tr><td><INPUT size=9 name=n29 value=^n29></td><td><INPUT
value=^d29></td><td>  </td><td><INPUT size=9 name=n30 value=^n30></td><td><INPUT
value=^d30></td>
      <td>  </td><td><INPUT size=9 name=n31 value=^n31></td><td><INPUT
value=^d31></td><td>  </td><td><INPUT size=9 name=n32 value=^n32></td>
```

```
      <td><INPUT value=^d32></td></tr>
      <tr><td><INPUT size=9 name=n33 value=^n33></td><td><INPUT
value=^d33></td><td>  </td><td><INPUT size=9 name=n34 value=^n34></td><td><INPUT
value=^d34></td>
      <td>  </td><td><INPUT size=9 name=n35 value=^n35></td><td><INPUT
value=^d35></td><td>  </td><td><INPUT size=9 name=n36 value=^n36></td>
      <td><INPUT value=^d36></td></tr>
      <tr><td><INPUT size=9 name=n37 value=^n37></td><td><INPUT
value=^d37></td><td>  </td><td><INPUT size=9 name=n38 value=^n38></td><td><INPUT
value=^d38></td>
      <td>  </td><td><INPUT size=9 name=n39 value=^n39></td><td><INPUT
value=^d39></td><td>  </td><td><INPUT size=9 name=n40 value=^n40></td>
      <td><INPUT value=^d40></td></tr>

</table>
</div>
</FORM>


<FORM name=addresses action=mon1spi.spi method=get>
      <INPUT type=hidden name=a1><INPUT type=hidden name=a2><INPUT type=hidden
name=a3><INPUT type=hidden name=a4><INPUT type=hidden name=a5>
      <INPUT type=hidden name=a6><INPUT type=hidden name=a7><INPUT type=hidden
name=a8><INPUT type=hidden name=a9><INPUT type=hidden name=a10>
      <INPUT type=hidden name=a11><INPUT type=hidden name=a12><INPUT type=hidden
name=a13><INPUT type=hidden name=a14><INPUT type=hidden name=a15>
      <INPUT type=hidden name=a16><INPUT type=hidden name=a17><INPUT type=hidden
name=a18><INPUT type=hidden name=a19><INPUT type=hidden name=a20>
      <INPUT type=hidden name=a21><INPUT type=hidden name=a22><INPUT type=hidden
name=a23><INPUT type=hidden name=a24><INPUT type=hidden name=a25>
      <INPUT type=hidden name=a26><INPUT type=hidden name=a27><INPUT type=hidden
name=a28><INPUT type=hidden name=a29><INPUT type=hidden name=a30>
      <INPUT type=hidden name=a31><INPUT type=hidden name=a32><INPUT type=hidden
name=a33><INPUT type=hidden name=a34><INPUT type=hidden name=a35>
      <INPUT type=hidden name=a36><INPUT type=hidden name=a37><INPUT type=hidden
name=a38><INPUT type=hidden name=a39><INPUT type=hidden name=a40>
      <INPUT type=hidden name=Trefrash>
</FORM>

      <br><br><br><br>

<p>
      <IMG src=Mode^PLC_Status'7.gif align="left" hspace="250" width="30" height="60">
</p>

<p>
      <IMG src=E^PLC_Status'3.gif align=left hspace=20 width=30 height=20>
      <IMG src=E^PLC_Status'2.gif align=left hspace=20 width=30 height=20>
      <IMG src=E^PLC_Status'1.gif align=left hspace=20 width=30 height=20>
      <IMG src=E^PLC_Status'0.gif align=left hspace=20 width=30 height=20>
</p>

      <br><br><br><br><br><br>
```

```
<FORM name=controls>
      <INPUT type=text name=Tmonitor>
      <INPUT type=button value='Monitorear al iPLC' onClick=Monitor()>
      <INPUT type=button value='Detener el Monitoreo' onClick=Stop_Monitor()>
</FORM>

<FORM name=stop action=mon1spi.spi method=get>
      <INPUT type=hidden name=Trefresh>
</FORM>


</BODY>

</HTML>


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Módulos de Expansión</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<META content="MSHTML 5.50.4134.600" name=GENERATOR>

</HEAD>

<BODY onLoad=setTimeout('Display_Modules();',100)>

<script language=javascript>

      var st

      function Display_Modules()
      {
            for (i=0; i<document.objs.elements.length; i++)
            {
                  switch(document.objs.elements[i].value)
                  {
                        case '255':
                              st='Módulo no Instalado'
                              break
                        case '1':
                              st='iMxD1  Módulo de Expansión con 8 Entradas Digitales y 8 Salidas Digitales'
                              break
                        case '2':
                              st='iMxA1  Módulo de Expansión con 8 Entradas Analógicas y 3 Salidas
Analógicas'
                              break
                        default:
                              st='¡El Módulo de Expansión Instalado no fue Reconocido!'
                  }

                  document.modules.elements[i].value=st
            }
      }

</script>
```

```
<H1>Información de los Módulos de Expansión Detectados en el Sistema del iPLC</H1> <br> Carlos A. Esquit H. <br> </H3>

<table border="15" cellpadding="1" cellspacing="3" width="350">
	<FORM name=modules>
	<tr><td>Módulo&nbspde&nbspExpansión #&nbsp1 </td><td>  </td><td><input size=100 name=m1></td></tr>
	<tr><td>Módulo de Expansión # 2</td><td>  </td><td><input size=100 name=m2></td></tr>
	<tr><td>Módulo de Expansión # 3</td><td>  </td><td><input size=100 name=m3></td></tr>
	<tr><td>Módulo de Expansión # 4</td><td>  </td><td><input size=100 name=m4></td></tr>
	<tr><td>Módulo de Expansión # 5</td><td>  </td><td><input size=100 name=m5></td></tr>
	<tr><td>Módulo de Expansión # 6</td><td>  </td><td><input size=100 name=m6></td></tr>
	<tr><td>Módulo de Expansión # 7</td><td>  </td><td><input size=100 name=m7></td></tr>
	<tr><td>Módulo de Expansión # 8</td><td>  </td><td><input size=100 name=m8></td></tr>
	<tr><td>Módulo de Expansión # 9</td><td>  </td><td><input size=100 name=m9></td></tr>
	<tr><td>Módulo de Expansión #10</td><td>  </td><td><input size=100 name=m10></td></tr>
	<tr><td>Módulo de Expansión #11</td><td>  </td><td><input size=100 name=m11></td></tr>
	<tr><td>Módulo de Expansión #12</td><td>  </td><td><input size=100 name=m12></td></tr>
	<tr><td>Módulo de Expansión #13</td><td>  </td><td><input size=100 name=m13></td></tr>
	<tr><td>Módulo de Expansión #14</td><td>  </td><td><input size=100 name=m14></td></tr>
	<tr><td>Módulo de Expansión #15</td><td>  </td><td><input size=100 name=m15></td></tr>
	</FORM>
</table>

<FORM name=objs>
	<input type=hidden name=emt1 value=^emt1><input type=hidden name=emt2 value=^emt2><input type=hidden name=emt3 value=^emt3>
	<input type=hidden name=emt4 value=^emt4><input type=hidden name=emt5 value=^emt5><input type=hidden name=emt6 value=^emt6>
	<input type=hidden name=emt7 value=^emt7><input type=hidden name=emt8 value=^emt8><input type=hidden name=emt9 value=^emt9>
	<input type=hidden name=emt10 value=^emt10><input type=hidden name=emt11 value=^emt11><input type=hidden name=emt12 value=^emt12>
	<input type=hidden name=emt13 value=^emt13><input type=hidden name=emt14 value=^emt14><input type=hidden name=emt15 value=^emt15>
</FORM>

</BODY>

</HTML>
```

APÉNDICE E

Código fuente de los programas cargados al microcontrolador central y a los módulos de expansión

```
;
;
;
;
;
;
;
;
;
;
```

Este es el FirmWare de la cpu central. Ofrece acceso a FLASH, RAM, coprocesador matemático, Sistema de timers, Sistema de Comunicación para módulos de expansión, tanto analógicos, como digitales, incorpora el Sistema de ejecución de instrucciones, implementa el set de instrucciones del PLC (TODAS las instrucciones) y además incluye el Sistema de Transmisión para el Monitoreo remoto, así como la implementación de marcas especiales y el byte de estado del Sistema (para el monitoreo). Finalmente se han agregado las dos intrucciones para el manejo del reloj de tiempo real ya incorporado (leer y escribir).
Carlos A. Esquit H.
Diseñado para un Cristal de 20 MHz.
Guatemala, 14 de abril de 2003.

```
LIST P=16F877
INCLUDE P16F877.INC

__CONFIG _WDT_OFF & _HS_OSC & _CP_OFF & _LVP_OFF & _PWRTE_ON & _BODEN_ON


#DEFINE     IORD                PORTE,1
#DEFINE     IOWR                PORTE,0
#DEFINE     CS                  PORTE,2
#DEFINE     Vpp                 PORTA,0
#DEFINE     Pak_Rst             PORTA,2
#DEFINE     Clk_Hi              PORTA,3
#DEFINE     Pak_IO              PORTA,4
#DEFINE     Clk_Low             PORTA,1
#DEFINE     Pak_Clk             PORTC,1
#DEFINE     Busy_Module         PORTC,0
#DEFINE     System_Is_Stopped   Mode_Flags,1
#DEFINE     In_System_Monitor   Mode_Flags,2
#DEFINE     Flow_On             Flow_Test,7
#DEFINE     PLC_Mode_LED        Mode_Flags,3
#DEFINE     RTC_Rd              PORTB,4
#DEFINE     RTC_Wr              PORTB,5
#DEFINE     RTC_AWr             PORTC,2

CBLOCK 0X20

        Digital_Inputs_Image1   ; Estas variables deben estar declaradas en estas direcciones, de forma fija,
        Digital_Inputs_Image2   ; ya que se escribirá/leerá en ellas utilizando direccionamiento indirecto.
        Digital_Inputs_Image3   ;
        Digital_Inputs_Image4   ;
```

```
Digital_Inputs_Image5     ;
Digital_Inputs_Image6     ;
Digital_Inputs_Image7     ; 0x26
Digital_Outputs_Image1    ; 0x27
Digital_Outputs_Image2    ;
Digital_Outputs_Image3    ;
Digital_Outputs_Image4    ;
Digital_Outputs_Image5    ;
Digital_Outputs_Image6    ;
Digital_Outputs_Image7    ; 0x2D
Addr_Bus_Low
Addr_Bus_Hi
Erase_Count
Write_Count
IO_Byte_Out
USART_Byte_In
Mode
Mode_Flags
Send_Pak
Receive_Pak
T_Area_Start
T_Area_End
T_1m_Count
SPI_Command
SPI_Byte_In
Module_Address
Instruction_Full
Instruction_Code
ByteX
ByteY
ByteY2
ByteZ
Byte1
Byte2
Byte3
Byte4
Byte5
Byte6
Byte7
Byte8
Byte9
```

```
Byte10
Temp
Tempx
Temp2
Ac0_Temp
Ac1_Temp
Ac2_Temp
PC_Hi
PC_Low
Logic_Stack
Logic_Stack_Copy
Edge_Up_Counter
Edge_Down_Counter
S_R_Bit_Counter
Data_To_RAM
Real_Sign
RTX_Code
Flow_Test
BOOLE_Code
Shift_Rotate_Code
Direction_Code
Inc_Dec_Code
Math_Code
LSBckUp
Stack_Depth             ; Utilizada para la introducción/Extracción de direcciones en el Stack del PC.
                        ; Sin esta variable, el retorno desde las subrutinas sería más lento.
Watchdog_Counter
Object_Address          ; equ 0x67.  Variable # 72.
Module_Type
Module_Type_Obj_Addr
SM0                     ; SM0, siempre es 0xFF.  Equ 0x6A.
SM1                     ; SM1, es 0xFF en el ciclo #1, 0x00 en todos los demás.
Scan_Time               ; SM2
Min_Scan_Time           ; SM3
Max_Scan_Time           ; SM4
PLC_Status              ; SM5
Scan_Clock              ; SM6

ENDC
```

```
        CBLOCK      0X72

                W_Temp
                STATUS_Temp
                FSR_Temp
                INDFx
                Pointer
                Ac0
                Ac1
                Ac2
                PCLATH_Temp
                Logic_Stack_Temp

        ENDC


        ORG 0
    goto        SYSTEM_INIT


        ORG 4                   ; Rutina de servicio de interrupciones del sistema.

Copy:                   ; Segmento utilizado para almacenar los registros de estado antes de ejecutar el servicio
                        ; de las interrupciones del sistema.
                        ; No se puede utilizar una subrutina para realizar esta tarea, ya que el programa ocupa
                        ; más de una página de memoria de programa, por lo que no es posible llamar a una
                        ; subrutina "Copy" sin modificar previamente el registro PCLATH (indeseable).
        movwf   W_Temp
        swapf   STATUS,W
        movwf   STATUS_Temp

            movf    FSR,W
            movwf   FSR_Temp

            banksel Tempx
            movf    Temp,W
```

```
        movwf   Tempx

        movf    Ac0,W
        movwf   Ac0_Temp
        movf    Ac1,W
        movwf   Ac1_Temp
        movf    Ac2,W
        movwf   Ac2_Temp

        movf    PCLATH,W
        movwf   PCLATH_Temp

        movf    Logic_Stack,W
        movwf   Logic_Stack_Temp

                        ; Fin del segmento "Copy".


;;; ********************************************** ¡CUIDADO! ***************************************************

;;; Debido a las instrucciones del PLC, es imposible utilizar subrutinas de interrupción que provoquen el uso
;;; del Stack en más de tres niveles, por lo que el recurso disponible del Stack para las interrupciones queda de la
;;; siguiente forma:
;;;                         1 nivel para la interrupción propiamente (evento de saltar a la dirección 0x0004).
;;;                         1 nivel para la realización de las tareas necesarias como servicio de interrupción
;;;                           de cualquier evento que la haya provocado, nivel que ya es utilizado para
;;;                           la determinación de la fuente de interrupción (ej. "call USART_Rx").

;;; Como ejemplo, las interrupciones provocadas por los timers YA ESTÁN UTILIZANDO AL MÁXIMO LA CAPACIDAD DEL STACK,
;;; Y POR NINGÚN MOTIVO SE PUEDE ANIDAR UNA SUBRUTINA DENTRO DE LA RUTINA DE SERVICIO DE INTERRUPCIÓN DE
TIMERS.

;;; ********************************************** ¡CUIDADO! ***************************************************



        bcf     PCLATH,4        ; Se selecciona la página 0.
        bcf     PCLATH,3

        btfsc   INTCON,T0IF
```

```
        goto    Timers_1_10_mS_Update ; Se utiliza 'goto' para las dos rutinas de actualización de los
                                      ; Temporizadores debido a la escasés de niveles en el Stack del
Returned_From_T_1_10_mS_Update:       ; PIC16F877.

        banksel PIR1
        btfsc   PIR1,TMR1IF
        goto    Timers_100mS_Update

Returned_From_T_100_mS_Update:

        banksel PIE1
        btfss   PIE1,RCIE
        goto    Interrupt_Service_End

        call    USART_Rx

Interrupt_Service_End:

        call    Paste           ; Se reestablecen los registros de estado.

        retfie
                                ; Fin de la rutina de servicio de interrupciones del sistema.  No altera el # de Banco.



Paste                           ; Rutina utilizada para reestablecer los registros de estado luego de haber atendido
                                ; las interrupciones del sistema.
        banksel Tempx
        movf    Tempx,W
        movwf   Temp

        movf    FSR_Temp,W
        movwf   FSR

        movf    Ac0_Temp,W
        movwf   Ac0

        movf    Ac1_Temp,W
        movwf   Ac1
```

```
        movf    Ac2_Temp,W
        movwf   Ac2

        movf    PCLATH_Temp,W
        movwf   PCLATH

        movf    Logic_Stack_Temp,W
        movwf   Logic_Stack

    swapf       STATUS_Temp,W
    movwf       STATUS
    swapf       W_Temp
    swapf       W_Temp,W

    return                          ;                       Selecciona el Banco que estaba activo
                                ; Fin de la rutina "Paste".     cuando se atendieron las interrupciones.



Timers_1_10_mS_Update       ; Rutina que actualiza los timers de 1 y 10 milisegundos.
                            ; Esta rutina es llamada desde el servicio de INTERRUPCIONES.

        bcf     INTCON,T0IF ; Se "limpia" la bandera de interrupción de desborde en el timer0.

        banksel TMR0
        movlw   .100        ; Se carga el registro del timer0 con el valor 236 para lograr que el desborde se
        movwf   TMR0        ; provoque cada 998.4 microsegundos (con prescaler 1:32).

        banksel T_Area_Start
        movlw   0x0F
        movwf   T_Area_Start    ; Se cargan las direcciones de inicio y fin en el área de memoria para los timers de 1 mS.
        movlw   0x17
        movwf   T_Area_End

        call    Timers_Update   ; Se realizan las actualizaciones para los timers de 1 milisegundo (4 Timers).

        btfss   System_Is_Stopped; Si el Sistema está detenido (modo 'STOP'), entonces ya NO se debe incrementar
```

```
        incf    Scan_Time       ; el contador del tiempo de ciclo.

        incf    T_1m_Count
        movf    T_1m_Count,W
        sublw   .10             ; Se verifica si ya se cumplió el periodo para la actualización de los timers de 10 mS.
        btfss   STATUS,Z
        goto    No_10m_Update
                                ; Si se cumplió el periodo para actualizar los timers de 10 milisegundos.
        movlw   0x17            ; Se cargan las direcciones de inicio y fin en el área de memoria para los timers de 10ms.
        movwf   T_Area_Start
        movlw   0x37
        movwf   T_Area_End

        call    Timers_Update   ; Se realizan las actualizaciones para los timers de 10 milisegundos (16 Timers).
        clrf    T_1m_Count      ; Se reinicia el periodo para la siguiente actualización de los timers de 10ms.

No_10m_Update:

        goto    Retumed_From_T_1_10_mS_Update

                                ; Fin de la rutina "Timers_1_10_mS_Update".     Banco 0.



Timers_100mS_Update             ; Rutina que actualiza los timers de 100 milisegundos.
                                ; Esta rutina es llamada desde el servicio de INTERRUPCIONES.

        banksel PIR1
        bcf     PIR1,TMR1IF     ; Se "limpia" la bandera de interrupción de desborde en el timer1.

        banksel TMR1L
        movlw   0xDC            ; Se cargan los registros del timer1 con el valor 3,036 para lograr que el desborde
        movwf   TMR1L           ; se provoque cada 100.0000 milisegundos (con prescaler 1:8).
        movlw   0x0B
        movwf   TMR1H

        banksel T1CON
        bsf     T1CON,T1CKPS1       ; Se selecciona el prescaler con la relación 1:8.
        bsf     T1CON,T1CKPS0
```

```
        bsf     T1CON,TMR1ON; Se enciende el timer1.

        banksel T_Area_Start
        movlw   0x37
        movwf   T_Area_Start    ; Se cargan los límites del área de memoria para los timers de 100 mS.
        movlw   0x6F
        movwf   T_Area_End

        call    Timers_Update   ; Se realizan las actualizaciones para los timers de 100 milisegundos (28 Timers).

        movlw   0x8F
        movwf   T_Area_Start    ; Se cargan los límites del área de memoria para los timers de 100 mS.
        movlw   0xEF
        movwf   T_Area_End

        call    Timers_Update   ; Se realizan las actualizaciones para los timers de 100 milisegundos (48 Timers).

        incf    Watchdog_Counter; Se incrementa el contador de vigilancia de tiempo de ciclo.
        movlw   .5
        subwf   Watchdog_Counter,W
        btfsc   STATUS,Z
        goto    Watchdog_Time_Out       ; El tiempo de ciclo excede de los 500 mS.

        goto    Returned_From_T_100_mS_Update

                                ; Fin de la rutina "Timers_100mS_Update".  Banco 0.



Timers_Update                   ; Rutina que actualiza los timers que se encuentren habilitados.
                                ; Esta rutina es ejecutada como consecuencia del servicio de INTERRUPCIONES.

        bsf     STATUS,IRP      ; Se direccionan indirectamente los bancos 2 y 3.

        banksel T_Area_Start
        movf    T_Area_Start,W
        movwf   Pointer         ; Se hace una copia del puntero del byte alto del timer actual (en este momento ninguno).

Next_Timer:
```

```
        movf    Pointer,W       ; Se verifica si ya se actualizaron todos los timers de 1 milisegundo, de ser así, se
        subwf   T_Area_End,W    ; finaliza esta rutina de actualización.
        btfsc   STATUS,Z
        return

        incf    Pointer         ; Se direcciona el byte alto del siguiente timer.
        incf    Pointer

        movf    Pointer,W
        movwf   FSR             ; Se carga el puntero del byte alto del timer actual en el registro FSR.
        movf    INDF,W
        movwf   INDFx
        btfss   INDFx,6         ; El timer no se actualizará si no está habilitado.
        goto    Next_Timer
        andlw   0x3F            ; Se verificará si el timer ha llegado a su valor máximo (0x3FFF), pues de ser así,
        sublw   0x3F            ; ya no se incrementará.
        btfss   STATUS,Z
        goto    Tmr_Inc         ; Se incrementará el timer, pues aún no ha llegado a su valor máximo.
        decf    FSR             ; Ahora se verificará si el byte bajo ya llegó a su valor máximo.
        movf    INDF,W
        sublw   0xFF
        btfss   STATUS,Z
        goto    Tmr_Inc         ; El timer aún no ha llegado a su valor máximo, razón por la cual será incrementado.
        goto    Next_Timer      ; El timer ya llegó a su valor máximo, razón por la cual ya NO será incrementado.
Tmr_Inc:
        movf    Pointer,W       ; Se carga el puntero del byte alto del timer actual.
        movwf   FSR
        decf    FSR             ; Se direcciona el byte bajo del timer actual.
        incf    INDF
        btfsc   STATUS,Z
        goto    Tmr_Inc_Hi_Byte ; Se desbordó el byte bajo del timer actual, por lo que debe incrementarse el byte alto.
        goto    Next_Timer

Tmr_Inc_Hi_Byte:
        incf    FSR
        incf    INDF
        goto    Next_Timer
                                ; Fin de la rutina "Timers_Update".          Banco 0.
```

```
USART_Rx                    ; Esta rutina lee el Comando/Dato recibido en el módulo serial y lo procesa según
                            ; el modo activo en el sistema (Pines RB7:RB6).
                            ; Esta rutina es llamada desde el servicio de INTERRUPCIONES.
        banksel RCREG
        movf    RCREG,W             ; Se lee el byte recibido, con lo cual también se borra la bandera de interrupción.
        movwf   USART_Byte_In ; Se almacena el byte recibido.


        movf    PORTB,W             ; Se lee el puerto B y se realiza un enmascaramiento para determinar el modo de
        andlw   b'11000000'   ; operación activo.
        movwf   Mode          ; El modo activo es almacenado en el registro 'Mode'.


Mode0_Verify:
        movlw   b'00000000'   ; Se verifica si el modo activo es 'RUN'.
        subwf   Mode,W
        btfss   STATUS,Z
        goto    Mode1_Verify

        banksel PIE1
        bcf     PIE1,RCIE     ; Se Deshabilita la interrupción de recepción USART para evitar conflictos con las
                              ; recepciones provocadas por solicitudes de instrucciones o rutinas dentro del ciclo
                              ; normal del PLC.

        banksel Mode_Flags    ; Selección del banco estándar (banco 0).

        return


Mode1_Verify:
        movlw   b'01000000'   ; Se verifica si el modo activo es 'STOP'.
        subwf   Mode,W
        btfss   STATUS,Z
        goto    Mode2_Verify

        return
```

```
Mode2_Verify:
        movlw   b'10000000'     ; Se verifica si el modo activo es 'ERASE'.
        subwf   Mode,W
        btfss   STATUS,Z
        goto    Mode3_Verify
        goto    FLASH_Erase

Mode3_Verify:
        movlw   b'11000000'     ; Se verifica si el modo activo es 'PROG'.
        subwf   Mode,W
        btfss   STATUS,Z
        return
        goto    FLASH_Write

                                ; Fin de la rutina "USART_Rx".          Banco 0.


FLASH_Write                     ; Rutina que escribe un byte de datos en la memoria FLASH P28F020.
                                ; Se asume que el byte de datos a escribir se encuentra almacenado en "USART_Byte_In",
                                ; y que los dos bytes de direccionamiento son 'Addr_Bus_Low' y 'Addr_Bus_Hi'.
        banksel IO_Byte_Out
        bsf     IORD            ; Primero se debe liberar el bus de datos.
        bsf     IOWR            ;
        movf    USART_Byte_In,W
        movwf   IO_Byte_Out     ; Se almacena el byte que se desea escribir.
        bsf     CS              ; Se selecciona el IC FLASH.
        bcf     Vpp             ; Se entra al modo de programación/Borrado del IC FLASH.
        movlw   .25
        movwf   Write_Count     ; Se inicializa la cuenta de ciclos de grabación (25 en total).
        incf    Addr_Bus_Low    ; Se incrementa la dirección en el bus de direcciones.
        btfsc   STATUS,Z        ; Se verifica si se debe incrementar el byte alto de la dirección (Addr_Bus_Hi).
        incf    Addr_Bus_Hi
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address     ; Se coloca la dirección en el bus de direcciones.
        bcf     PCLATH,3        ; Se selecciona la página 0.
Write_Loop:
```

```
        decfsz  Write_Count
        goto    No_Error

        bsf     Vpp             ; Se finaliza el modo de programación/borrado del IC FLASH.
        bsf     CS              ; Se deshabilita el IC FLASH (con lo cual, a la vez se habilita el IC RAM).
        clrf    Addr_Bus_Low    ; Se inicializa con cero el bus de direcciones.
        clrf    Addr_Bus_Hi
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address
        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   0x2D            ; Código de error para la escritura en FLASH.
        goto    Show_System_Error

No_Error:
        movlw   0x40
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write; Se escribe el comando de grabación FLASH.
        movf    IO_Byte_Out,W   ; Se escribe el dato en la FLASH, en la posición de memoria actualmente direccionada,
        call    IO_Write; que ha sido cargada a los registros de direccionamiento 'Addr_Bus_Hi' y 'Addr_Bus_Low'
                                ; previo a la ejecución de esta rutina.
        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   .17
        movwf   Ac0
Wait_10_u_secs:
        decfsz  Ac0
        goto    Wait_10_u_secs

        movlw   0xC0            ; Se escribe el comando de verificación de grabación.
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write
        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   .10
        movwf   Ac0
Wait_6_u_sec:
        decfsz  Ac0
        goto    Wait_6_u_sec

        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Read").
```

```
        call    IO_Read                 ; Se verifica si el byte fue grabado correctamente.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        subwf   IO_Byte_Out,W
        btfss   STATUS,Z
        goto    Write_Loop

        movlw   0x00            ; Se escribe el comando 'READ' en el IC FLASH.
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write
        bcf     PCLATH,3        ; Se selecciona la página 0.
        bsf     Vpp             ; Se finaliza el modo de programación/borrado del IC FLASH.
        bsf     CS              ; Se deshabilita el IC FLASH (con lo cual, a la vez se habilita el IC RAM).

;       call    Update_Rx_Ack   ; Se envía la señal de "acknowledge" de recepción USART.

        goto    Stopped_Loop    ; Luego de una escritura/borrado de FLASH no se retorna al ciclo del PLC (SYSTEM_Cycle),
                                ; sino que se debe cambiar directamente al modo 'STOP'.

                                ; Fin de la rutina "FLASH_Write".       Banco 0.



FLASH_Erase                     ; Rutina que borra la memoria FLASH.

        banksel PORTE
        bsf     IORD            ; Primero se debe liberar el bus de datos.
        bsf     IOWR            ;
        bsf     CS              ; Se selecciona el IC FLASH.
        bcf     Vpp             ; Se entra al modo de programación/Borrado del IC FLASH.
        clrf    Erase_Count     ; Se borra la cuenta de ciclos de borrado.
        clrf    Addr_Bus_Low    ; Se inicializa con cero el bus de direcciones.
        clrf    Addr_Bus_Hi
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address
        bcf     PCLATH,3        ; Se selecciona la página 0.

Erase_Loop:
        decfsz  Erase_Count
        goto    No_Err
```

```
        bsf     Vpp             ; Se finaliza el modo de programación/borrado del IC FLASH.
        bsf     CS              ; Se deshabilita el IC FLASH (con lo cual, a la vez se habilita el IC RAM).

        movlw   0x2E            ; Código de error para el borrado de la memoria FLASH.
        goto    Show_System_Error

No_Err:
        movlw   0x20            ; Se escribe el comando 'ERASE' en el IC FLASH (primera vez).
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write
        call    IO_Write; Se escribe el comando 'ERASE' en el IC FLASH (segunda vez).
        bcf     PCLATH,3        ; Se selecciona la página 0.

        clrf    Ac0
        movlw   .22
        movwf   Ac1
Wait_10_m_secs:                 ; Este ciclo de espera dura aproximadamente 10.098 milisegundos.
        decfsz  Ac0
        goto    Wait_10_m_secs
        decfsz  Ac1
        goto    Wait_10_m_secs ; Fin del ciclo de espera de 10 milisegundos.

Verify:
        movlw   0xA0            ; Se escribe el comando 'ERASE VERIFY' en el IC FLASH.
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write
        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   .10
        movwf   Ac0
Wait_6_u_secs:
        decfsz  Ac0
        goto    Wait_6_u_secs

        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Read").
        call    IO_Read
        bcf     PCLATH,3        ; Se selecciona la página 0.

        sublw   0xFF            ; Se verifica si el byte actual ya está borrado (valor=0xFF).
        btfss   STATUS,Z
```

```
        goto    Erase_Loop
        incf    Addr_Bus_Low    ; Se incrementa la dirección en el bus de direcciones.
        btfsc   STATUS,Z        ; Se verifica si se debe incrementar el byte alto de la dirección (Addr_Bus_Hi).
        incf    Addr_Bus_Hi
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address     ; Se escribe la nueva dirección a verificar en el bus de direcciones del sistema.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        movf    Addr_Bus_Hi,W
        sublw   .255            ; Se verifica si ya se borraron los primeros 64K Bytes del IC FLASH.
        btfss   STATUS,Z
        goto    Verify

        movlw   0x00            ; Se escribe el comando 'READ' en el IC FLASH.
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "IO_Write").
        call    IO_Write
        bcf     PCLATH,3        ; Se selecciona la página 0.
        bsf     Vpp             ; Se finaliza el modo de programación/borrado del IC FLASH.
        bsf     CS              ; Se deshabilita el IC FLASH (con lo cual, a la vez se habilita el IC RAM).

        clrf    Addr_Bus_Hi     ; Se inicializa el bus de direcciones con el valor 0000h.
        clrf    Addr_Bus_Low
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address
        bcf     PCLATH,3        ; Se selecciona la página 0.

        call    FLASH_Write_OK

;       call    Update_Rx_Ack   ; Se envía la señal de "acknowledge" de recepción USART.

        goto    Stopped_Loop    ; Luego de una escritura/borrado de FLASH no se retorna al ciclo del PLC (SYSTEM_Cycle),
                                ; sino que se debe cambiar directamente al modo 'STOP'.

                                ; Fin de la rutina "FLASH_Erase".       Banco 0.



FLASH_Write_OK

        banksel TXREG
```

```
        movlw   0x90            ; Comando WriteX.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x14
        movwf   TXREG           ; lsB de la dirección de IO3 en el Módulo SP.
        call    Poll_Tx
        movlw   0xFF
        movwf   TXREG           ; msB de la dirección de IO3 en el Módulo SP.
        call    Poll_Tx
        movlw   .0
        movwf   TXREG           ; Se enciende el LED msb.
        call    Poll_Tx

        return
                                ; Fin de la rutina "FLASH_Write_OK".        Banco 0.



Poll_Rx                         ; Rutina que "polea" la bandera de recepción USART y retorna hasta que se ha recibido
        banksel PIR1            ; un byte.
    btfss       PIR1,RCIF
    goto        Poll_Rx

    return
                                ; Fin de la rutina "Poll_RX".               Banco 0.



Poll_Tx                         ; Rutina que "polea" la bandera de transmisión USART y retorna hasta que se ha
        banksel TXSTA           ; transmitido el byte.
    btfss       TXSTA,TRMT
    goto        Poll_Tx

        banksel TXREG
    return
                                ; Fin de la rutina "Poll_Tx".               Banco 0.
```

```
Bytes_Delay                 ; La ejecución de esta rutina provoca una espera de 1 uS.  Esta rutina es utilizada
                            ; por las rutinas de comunicación con el Pak-IX.
        nop

        return
                            ; Fin de la rutina "Bytes_Delay".          No Altera el Banco.


Busy_Pak                    ; Rutina que espera hasta que los resultados del Pak-IX estén listos para ser leídos
                            ; por el CPU central (PIC16F877).
        banksel  PORTA
        movf     PORTA,W
        movwf    Temp
        btfsc    Temp,2     ; Se verifica si el Pak ya finalizó los cálculos y está listo para enviar los resultados.
        goto     Busy_Pak

        return
                            ; Fin de la rutina "Busy_Pak".                Banco 0.


Pak_Tx                      ; Rutina que Transmite un byte (el contenido de W) hacia el PAK-IX.

        banksel  TRISA
        bcf      TRISA,2
        banksel  Send_Pak
        movwf    Send_Pak

        movlw    .8         ; Cantidad de bits que se enviarán.
        movwf    Ac2
Bit_Tx:
        rlf      Send_Pak
```

```
        bcf     Pak_IO          ; Se asume que el primer bit (MSb) es uno (0).
        btfss   STATUS,C
        bsf     Pak_IO          ; Si no era uno, entonces se asigna un cero (1).
        call    Clk_Pulse       ; Se da el pulso.
        decfsz  Ac2
        goto    Bit_Tx

        call    Bytes_Delay
        call    Busy_Pak

        return
                                ; Fin de la rutina "Pak_Tx".            Banco 0.



Pak_Rx                          ; Rutina que Recibe un byte desde el PAK-IX y lo almacena en Receive_Pak.

        banksel TRISA
        bsf     TRISA,4         ; El pin RA4 será utilizado como entrada de datos seriales desde el Pak-IX.

        movlw   .8              ; Cantidad de bits que se recibirán.
        movwf   Ac2
Bit_Rx:
        banksel PORTA
        movf    PORTA,w                 ; Se lee el Puerto A para cargar el Latch con los datos actuales reales.
        movwf   Temp
        bsf     STATUS,C        ; Se asume que el primer bit (MSb) es uno (1).
        btfss   Temp,4
        bcf     STATUS,C        ; Si no era uno, entonces se asigna un cero (0).
        rlf     Receive_Pak
        call    Clk_Pulse       ; Se da el pulso.
        decfsz  Ac2
        goto    Bit_Rx

        call    Bytes_Delay
        call    Busy_Pak

        return
                                ; Fin de la rutina "Pak_Rx".            Banco 0.
```

```
Clk_Pulse                   ; Rutina que genera un pulso de reloj para el PAK-IX.

        banksel PORTC
        bsf     Pak_Clk     ; Flanco de subida.

        movlw   .2
        movwf   Ac0
THigh:
        decfsz  Ac0
        goto    THigh

        bcf     Pak_Clk     ; Flanco de bajada.

        movlw   .3
        movwf   Ac0
TLow:
        decfsz  Ac0
        goto    TLow

        return
                            ; Fin de la rutina "Clk_Pulse".          Banco 0.


Math_Error                  ; Rutina que realiza la notificación cuando ha ocurrido un error en el cálculo de una
                            ; operación matemática.  El sistema pasa al modo 'STOP'.

        movwf   Byte1       ; Se almacena el código original del error producido en el coprocesador matemático.

        btfsc   Byte1,0
        movlw   0x28        ; overflow de entero.
        btfsc   Byte1,1
        movlw   0x27        ; fp overflow.
        btfsc   Byte1,2
```

```
        movlw   0x26            ; fp underflow.
        btfsc   Byte1,3
        movlw   0x25            ; División entre cero.
        btfsc   Byte1,4
        movlw   0x24            ; No es un número.
        btfsc   Byte1,5
        movlw   0x23            ; Error de dominio.

        goto    Show_System_Error
                                ; Fin de la rutina "Math_Error".        Banco 0.



Watchdog_Time_Out               ; Rutina que realiza la notificación cuando el tiempo del ciclo del PLC ha excedido
                                ; del máximo permitido (500 mS).
        movlw   0x29
        goto    Show_System_Error
                                ; Fin de la rutina "Watchdog_Time_Out".     Banco 0.



EEPROM_Wr                       ; Rutina que escribe un byte (W) en la EEPROM.  Por ahora, la dirección es 0x00.

        banksel EEADR
        clrf    EEADR           ; Dirección EEPROM en la que se escribirá.

        banksel EEDATA
    movwf       EEDATA                  ; Dato que se escribirá.

        banksel EECON1
        bcf     EECON1,EEPGD
    bsf EECON1,WREN
    movlw       0x55
    movwf       EECON2
    movlw       0xAA
    movwf       EECON2
    bsf EECON1,WR
```

```
        banksel Mode_Flags

        return
                                ; Fin de la Rutina "EEPROMWr".          Banco 0.



SPI_Poll
        banksel SSPSTAT
        btfss   SSPSTAT,BF
        goto    SPI_Poll
        banksel SSPBUF
        movf    SSPBUF,W        ; Se lee el búfer de transmisión para borrar la bandera de recepción.
        movwf   SPI_Byte_In     ; Se almacena el byte recibido.
        movlw   .6
        movwf   Ac0
W_Loop:
        decfsz  Ac0             ; Ciclo de espera adicional para asegurar que el módulo tenga tiempo suficiente para
        goto    W_Loop          ; colocar el estado adecuado de la señal 'Busy_Module' (espera = 4.0 uS).

        return
                                ; Fin de la rutina "SPI_Poll".          Banco 0.



Test_Busy_Module                ; Rutina de espera mientras el módulo direccionado no se encuentre listo para realizar
        clrf    Ac0             ; la transacción con la cpu central.
        banksel PORTC
Busy_Loop:
        decf    Ac0
        btfsc   STATUS,Z        ; Se venció el periodo Tm=357 microsegundos para que el módulo responda si se encuentra
        goto    Module_Item_Access_Error; presente, razón por la cual se notificará que ha sucedido un error de E/S.
        movf    PORTC,W                 ; Se lee el puerto C para cargar los estados en el latch.
        btfsc   Busy_Module
        goto    Busy_Loop
```

```
        return
                                ; Fin de la rutina "Test_Busy_Module".          Banco 0.


Module_First_Byte_Wait          ; Esta rutina espera 7.6 microsegundos para dar tiempo a que el módulo lea y procese el
                                ; byte de comando y direccionamiento (Byte #1), y esté listo para continuar con la
        movlw   .12             ; transacción con la cpu central.
        movwf   Ac0
mfbw:
        decfsz  Ac0
        goto    mfbw
        return
                                ; Fin de la rutina "Module_First_Byte_Wait". No altera la selección del banco.


Module_Last_Byte_Wait           ; Esta rutina espera 3.4 microsegundos para dar tiempo a que el módulo ejecute las
                                ; tareas finales provocadas por el comando, retorne a su ciclo principal, y esté listo
        movlw   .5              ; para iniciar una nueva transacción con la cpu central.
        movwf   Ac0
mlbw:
        decfsz  Ac0
        goto    mlbw
        return
                                ; Fin de la rutina "Module_Last_Byte_Wait". No altera la selección del banco.


Module_Item_Access_Error        ; Rutina que realiza la notificación de error cuando la cpu central intenta acceder a
                                ; un elemento no existente en el módulo seleccionado. El sistema pasa al modo 'STOP'.
        banksel TXREG

        btfss   In_System_Monitor       ; Si el error fue provocado por una solicitud del Sistema de monitoreo,
        goto    Program_Source          ; simplemente se finaliza la solicitud, de lo contrario, se realiza la
```

```
                                        ; notificación de error y se pasa al modo 'STOP'.
Monitor_Source:                         ; El error fue provocado por una solicitud de monitoreo.
        movlw   0x00
        return

Program_Source:                 ; El error fue provocado por una instrucción de programa.
        movlw   0x2C
        goto    Show_System_Error

                                        ; Fin de la rutina "Module_Item_Access_Error".      Banco 0.



Read_Digital_Inputs_Phase       ; Rutina que realiza la lectura de las entradas digitales en todos los módulos de
                                ; expansión y genera las imágenes correspondientes a partir de ello.

        banksel SPI_Command
        movlw   0x11
        movwf   SPI_Command     ; Se carga el comando de lectura de entradas digitales con la dirección del módulo #1.

        bcf     STATUS,IRP      ; Se direccionará indirectamente el banco 0 (área de registros de imagen de E/S).

        movlw   0x20            ; Se inicializa el puntero con la dirección del primer registro de imagen de las
        movwf   FSR             ; entradas digitales.
Digital_Inputs_Loop:
        movf    SSPBUF,W        ; Se lee el buffer SPI para borrar la bandera de recepción.

        movf    SPI_Command,W
        movwf   SSPBUF                  ; Se envía el byte de comando de lectura de entradas digitales para el módulo i.
        call    SPI_Poll ; (Byte # 1 del protocolo de la comunicación SPI).

        call    Module_First_Byte_Wait

        movlw   .0              ; Se envía el byte del # de elemento a acceder (irrelevante para este comando).
        movwf   SSPBUF                  ; (Byte # 2 del protocolo de la comunicación SPI).
        call    SPI_Poll

        movwf   SSPBUF                  ; Lectura:  Se recibe el byte de las entradas digitales del módulo direccionado.
```

```
        call    SPI_Poll ; (Byte # 3 del protocolo de la comunicación SPI).
        movf    SPI_Byte_In,W
        movwf   INDF

        call    Module_Last_Byte_Wait

        incf    SPI_Command     ; Se direcciona el siguiente módulo de expansión.

        incf    FSR
        movf    FSR,W
        sublw   0x27            ; Se verifica si ya se escribieron todas las imagenes de las entradas digitales.
        btfss   STATUS,Z
        goto    Digital_Inputs_Loop

        return
                                ; Fin de la rutina "Read_Digital_Inputs_Phase".      Banco 0.



Write_Digital_Outputs_Phase     ; Rutina que realiza la escritura a las salidas digitales en todos los módulos de
                                ; expansión.

        banksel SPI_Command
        movlw   0x21
        movwf   SPI_Command     ; Se carga el comando de lectura de entradas digitales con la dirección del módulo #1.

        bcf     STATUS,IRP      ; Se direccionará indirectamente el banco 0 (área de registros de imagen de E/S).

        movlw   0x27            ; Se inicializa el puntero con la dirección del primer registro de imagen de las
        movwf   FSR             ; salidas digitales.
Digital_Outputs_Loop:
        movf    SSPBUF,W        ; Se lee el buffer SPI para borrar la bandera de recepción.

        movf    SPI_Command,W
        movwf   SSPBUF                  ; Se envía el byte de comando de escritura a las salidas digitales para el módulo i.
        call    SPI_Poll ; (Byte # 1 del protocolo de la comunicación SPI).

        call    Module_First_Byte_Wait
```

```
        movlw   .0              ; Se envía el byte del # de elemento a acceder (irrelevante para este comando).
        movwf   SSPBUF                  ; (Byte # 2 del protocolo de la comunicación SPI).
        call    SPI_Poll

        movf    INDF,W          ; Se escribe el byte de las salidas digitales del módulo direccionado.
        movwf   SSPBUF
        call    SPI_Poll; (Byte # 3 del protocolo de la comunicación SPI).

        incf    SPI_Command     ; Se direcciona el siguiente módulo de expansión.

        incf    FSR
        movf    FSR,W
        sublw   0x2E            ; Se verifica si ya se escribieron todas las imagenes de las salidas digitales.
        btfss   STATUS,Z
        goto    Digital_Outputs_Loop

        return
                                ; Fin de la rutina "Write_Digital_Outputs_Phase".       Banco 0.



Test_Flow_PCx                   ; Rutina que determina si existe flujo para la ejecución de las instrucciones.
                                ; Utiliza el valor almacenado en W como parámetro para el número de posiciones
                                ; que se deben adicionar al PC en caso NO exista flujo (el número depende de
                                ; la instrucción que realizó la llamada).
        banksel Flow_Test
        bsf     Flow_On                 ; Se ASUME que SI existe flujo.

        btfsc   Logic_Stack,7   ; Se verifica si el tope de la pila está en uno.
        return                  ; SI existe flujo, por lo que se retorna para ejecutar la instrucción.

        bcf     Flow_On                 ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low          ; Debe incrementarse el PC en cuatro (4) posiciones (contienen parámetros de
        btfsc   STATUS,C        ; la función).
        incf    PC_Hi

        return                  ; Se retorna para continuar con la ejecución de las demás instrucciones.
                                ; El retorno es hacia la instrucción que NO se ejecutará, pero este retorno
                                ; es necesario para evitar que se provoque un desborde en el Stack del PIC16F877.
```

```
                                ; Fin de la rutina "Test_Flow_PCx". ; Banco 0.


Show_PLC_Mode_LED               ; Rutina que enciende el LED correspondiente con el modo activo del PLC.

        banksel TXREG
        movlw   0x90            ; Comando 'SP WriteX'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x18            ; lsB de la dirección del pin IO7 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF            ; msB de la dirección del pin IO7 del módulo SP.
        movwf   TXREG
        call    Poll_Tx

        movlw   0x00
        btfsc   PLC_Mode_LED
        movlw   0x02

        movwf   TXREG           ; Se enciende el LED correspondiente al modo activo del PLC.
        call    Poll_Tx

        return
                                ; Fin de la rutina "Show_PLC_Mode_LED".         ; Banco 0.


Tx_PLC_Status                   ; Rutina que transmite el byte de estado del PLC hacia el objeto correspondiente
                                ; mapeado en el módulo SP (obj. 0x01)
        banksel TXREG
        movlw   0x80            ; Comando 'SP Write'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x01            ; Dirección del objeto 'PLC_Status' (SM5) en el módulo SP.
```

```
        movwf   TXREG
        call    Poll_Tx

        movf    PLC_Status,W
        movwf   TXREG           ; Se transmite el estado actual del PLC.
        call    Poll_Tx

        retum
                                ; Fin de la rutina "Tx_PLC_Status".          ; Banco 0.


Show_System_Error               ; Rutina que enciende los LEDs correspondientes al último error
                                ; producido en el Sistema.  Se asume que el código de error a escribir
                                ; se encuentra en el registro W.
        banksel TXREG
        movwf   Byte1           ; Se almacena el código de error.

        movlw   0x90            ; Comando 'SP WriteX'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x00            ; lsB de la dirección del puerto 1 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF            ; msB de la dirección del puerto 1 del módulo SP.
        movwf   TXREG
        call    Poll_Tx

        movf    Byte1,W

        movwf   TXREG           ; Se encienden los LEDs correspondientes al último error producido.
        call    Poll_Tx

        movwf   PLC_Status      ; Se escribe el byte especial para los bits de estado del PLC.
        call    Tx_PLC_Status   ; Se transmite el byte de estado del PLC hacia el módulo SP (objeto 0x01).

        btfsc   Byte1,7         ; Se determina si la rutina fue llamada desde el ciclo del modo 'RUN'.
        goto    SYSTEM_Cycle
        goto    SYSTEM_Stopped
```

```
                                ; Fin de la rutina "Show_System_Error".          ; Banco 0.


Min_Max_Scan_Time               ; Rutina que actualiza los bytes de memoria especial SM3 (tiempo de ciclo mínimo) y
                                ; SM4 (tiempo de ciclo máximo).
        movf    Scan_Time,W
        btfsc   SM1,7           ; Si este es el primer ciclo que ejecuta el PLC, entonces los bytes de memoria especial
        movwf   Min_Scan_Time   ; SM3 (Min_Scan_Time) y SM4 (Max_Scan_Time) adquieren el valor de SM2 (Scan_Time).
        btfsc   SM1,7
        movwf   Max_Scan_Time

        subwf   Min_Scan_Time,W         ; Se determina si el tiempo del ciclo actual es el menor que se ha presentado.
        btfsc   STATUS,C
        goto    New_Min_Scan_Time       ; El tiempo del ciclo actual SI es el mínimo que se ha presentado.

        movf    Scan_Time,W
        subwf   Max_Scan_Time,W         ; Se determina si el tiempo del ciclo actual es el mayor que se ha presentado.
        btfss   STATUS,C
        goto    New_Max_Scan_Time       ; El tiempo del ciclo actual SI es el máximo que se ha presentado.

        retum

New_Min_Scan_Time:              ; Se actualiza el valor mínimo del tiempo de ciclo del PLC.
        movf    Scan_Time,W
        movwf   Min_Scan_Time
        retum

New_Max_Scan_Time:              ; Se actualiza el valor máximo del tiempo de ciclo del PLC.
        movf    Scan_Time,W
        movwf   Max_Scan_Time
        retum
                                ; Fin de la rutina "Min_Max_Scan_Time".         ; Banco 0.


SYSTEM_Stopped                  ; Esta rutina es la que el Sistema ejecuta cíclicamente cuando se encuentra en el modo
```

```
                                ; 'STOP'.

        banksel Mode_Flags
        bcf     PLC_Mode_LED ; Se enciende el LED indicador del modo 'STOP'.
        bcf     PLC_Status,7    ; Se actualiza el bit de estado para el modo de operación del PLC (byte especial SM5).
        call    Show_PLC_Mode_LED

        banksel PORTB
        bsf     System_Is_Stopped; El Sistema SI está detenido.
        movf    PORTB,W
        bcf     PLC_Status,6    ; Se borran los bits del modo de operación del PLC en el btye SM5.
        bcf     PLC_Status,5
        btfsc   PORTB,7                 ; Se escribe el modo de operación del PLC en el byte SM5.
        bsf     PLC_Status,6
        btfsc   PORTB,6
        bsf     PLC_Status,5

        call    Tx_PLC_Status   ; Se transmite el byte de estado del PLC hacia el módulo SP (obj. 0x01).

Stopped_Loop:
        call    Monitor_System  ; Se envía la siguiente variable al Monitor en la página Web.

        bsf     INTCON,GIE      ; Se habilitan las interrupciones debido a que el sistema puede haberse colocado en el
                                ; modo 'STOP' desde el servicio de interrupciones y sin la conclusión de la misma, por
                                ; lo que aquí se rehabilitan las interrupciones para que el sistema continúe
                                ; respondiendo a los comandos seriales (que lo pueden colocar en el modo 'RUN').

        clrf    Watchdog_Counter; Se borra el contador de vigilancia de tiempo de ciclo.

        movf    PORTB,W
        andlw   b'11000000'     ; Se verifica si el modo a cambiado a 'RUN'.
        btfss   STATUS,Z
        goto    Stopped_Loop

                                ; SI se cambió al modo 'RUN'.  Se enciende el LED de modo 'RUN', se borra
                                ; el código del último error producido en el Sistema, y se pasa al
                                ; modo 'RUN' (desde "Show System Error").
        movlw   0xFF
        movwf   SM1             ; Se reinicializa SM1 con 0xFF ya que se realizará la transición 'STOP'--'RUN'.
        movlw   0x8F
        goto    Show_System_Error
```

```
                        ; Fin de la rutina "SYSTEM_Stopped".



SYSTEM_Cycle            ; Esta es la rutina del Ciclo del PLC.  El Sistema ejecuta cíclicamente esta rutina
                        ; cuando se encuentra en el modo 'RUN'.

        bcf     PCLATH,3        ; Se selecciona la página 0 de la memoria de programa.
        banksel Mode_Flags

        clrf    Watchdog_Counter; Se borra el contador de vigilancia de tiempo de ciclo.

        clrf    Scan_Time

        comf    Scan_Clock      ; Se genera la señal de reloj para el ciclo del PLC.

        bcf     System_Is_Stopped;El Sistema NO está detenido (está en el ciclo de ejecución del programa del PLC).

        movf    PORTB,W
        andlw   b'11000000'
        btfss   STATUS,Z        ; El ciclo se ejecutará únicamente si el modo es 'RUN', de lo contrario, la ejecución
        goto    SYSTEM_Stopped          ; del programa se traslada al modo 'STOP'.

        banksel PIE1            ; se deshabilita la interrupción de recepción USART para evitar conflictos con las
        bcf     PIE1,RCIE       ; recepciones provocadas por rutinas/instrucciones dentro del ciclo de programa del PLC.

                                ; Se ejecuta la fase de lectura de las entradas digitales (todos los módulos).
        call    Read_Digital_Inputs_Phase

        bsf     PCLATH,3        ; Se selecciona la página 1 de la memoria de programa.

        call    Execute_Program         ; Se ejecuta el programa cargado en la memoria FLASH del PLC.

        bcf     PCLATH,3        ; Se selecciona la página 0 de la memoria de programa.

                                ; Se ejecuta la fase de escritura a las salidas digitales (todos los módulos).
        call    Write_Digital_Outputs_Phase

        call    Monitor_System  ; Se envía la siguiente variable al Monitor en la página Web.
```

```
        call    Min_Max_Scan_Time       ; Se actualizan los bytes SM3 (Min_Scan_Time) y SM4 (Max_Scan_Time).
        clrf    SM1                     ; SM1 será igual a cero (0) para todos los demás ciclos del PLC.
        goto    SYSTEM_Cycle
                                        ; Fin de la rutina "SYSTEM_Cycle".



SYSTEM_INIT                             ; Inicio de la Rutina de Inicialización de Módulos Periféricos, Registros de
                                        ; configuración y Variables.
        clrf    INTCON

                                        ;~~~~~~~~~~~~ Inicialización del Módulo SPI Master ~~~~~~~~~~~~

        banksel PORTC
        bcf     PORTC,3                 ; Se coloca en bajo la señal de reloj (SPI).

        banksel TRISC
        bsf     TRISC,1                 ; El pin RC1 será utilizado como entrada para la señal "Busy" de los módulos de exp.
        bcf     TRISC,3                 ; El pin RC3 será utilizado como salida de reloj para el módulo Master (cpu central).
        bcf     TRISC,5                 ; El pin RC5 será utilizado como salida de datos.

        banksel SSPCON
        bsf     SSPCON,SSPEN; Se habilita el módulo SSP.

        clrf    Ac0
        clrf    Ac1
        movlw   .10
        movwf   Ac2
Modules_Init_Wait:                      ; Espera de 2.2 segundos antes de entrar al ciclo del PLC para permitir que los módulos
        decfsz  Ac0                     ; de expansión sean inicializados, reciban una dirección, y estén listos para realizar
        goto    Modules_Init_Wait;transacciones.
        decfsz  Ac1
        goto    Modules_Init_Wait
        decfsz  Ac2
        goto    Modules_Init_Wait
```

```
                        ;~~~~~~~~~~~~ Fin de la Inicialización del Módulo SPI Master ~~~~~~~~~~~~


                        ;~~~~~~~~~~~~ Configuración para el Módulo USART ~~~~~~~~~~~~

bsf     STATUS,RP0      ; Banco 1.
movlw   .129
movwf   SPBRG           ; 9600 bps (con BRGH=1).
movlw   b'00100110'
movwf   TXSTA           ; Modo asíncrono, 8 bits, alta velocidad.
movlw   b'00000000'
movwf   PIE1    ; El sistema no utilizará las interrupciones de los
        ; periféricos.
bcf     STATUS,RP0      ; Banco 0.
movlw   b'10010000'
movwf   RCSTA           ; Habilitación del puerto serial, 8 bits, continuo.


movlw   0x33            ; Se escribe el comando 'SP ComParams'.
movwf   TXREG
call    Poll_Tx

movlw   0xF5            ; Se escribe el lsB del registro 'SP Baud Rate' (115.2 Kbps).
movwf   TXREG
call    Poll_Tx

movlw   0xFF            ; Se escribe el msB del registro 'SP Baud Rate'.
movwf   TXREG
call    Poll_Tx

movlw   0xEF            ; Se escribe el lsB del código para el delay Tx/Rx (5 uS).
movwf   TXREG
call    Poll_Tx

movlw   0xFF            ; Se escribe el msB del código para el delay Tx/Rx.
movwf   TXREG
call    Poll_Tx

bsf     STATUS,RP0      ; Banco 1.
```

```
    movlw   .10
    movwf   SPBRG           ; 113636 bps (con BRGH=1).

                            ;~~~~~~~~~~~~ Fin de la Configuración Inicial para el Módulo USART ~~~~~~~~~~~~


    banksel ADCON1
    movlw   b'00000111'
    movwf   ADCON1                  ; No se Utilizarán canales analógicos.

    banksel OPTION_REG
    movlw   b'00000100'
    movwf   OPTION_REG      ; Prescaler 1:32 asignado al Módulo del Timer0.


                            ;~~~~~~~~~~~~ Inicialización de las señales de Control ~~~~~~~~~~~~
    banksel PORTA
bsf Vpp             ; Se pone en alto el pin RA0 para EVITAR que la FLASH entre al modo de programación.
    bsf     Clk_Hi          ; Se colocan en alto las dos señales de reloj para los FFs del bus de direcciones.
    bsf     Clk_Low
    bcf     Pak_Clk         ; El reloj del coprocesador matemático debe permanecer en cero (0) cuando no haya E/S.
    banksel TRISA           ; El pin RA0 será utilizado para activar el voltaje de programación de la memoria FLASH.
bcf TRISA,0         ; La señal de RA0 será activa en bajo, es decir, un cero en RA0 provocará la tensión de
                            ; programación en el pin Vpph del IC FLASH.
    bcf     TRISA,1         ; El pin RA1 será utilizado para habilitar/deshabilitar al Pak-IX (funciona como Reset).
    bsf     TRISA,2         ; El pin RA2 será utilizado como señal "Busy" del Pak-IX hacia el PIC16F877 (entrada).
    bcf     TRISA,3         ; Señal de reloj para el MSB del bus de direcciones.
    bcf     TRISA,5         ; Señal de reloj para el LSB del bus de direcciones.
    bsf     TRISB,7         ; Estos pines serán utilizados para la selección del modo de operación del sistema.  Este
    bsf     TRISB,6         ; modo será detectado cada vez que se reciba un byte en el módulo USART (leyendo RB7 y
                            ; RB6).
    bcf     TRISB,5         ; Este pin será utilizado como señal de escritura hacia el RTC.
    bcf     TRISB,4         ; Este pin será utilizado como señal de lectura desde el RTC.
    bcf     TRISC,0         ; El pin RC0 será utilizado como salida para la señal de reloj conectada al cop. matemático.
    bcf     TRISC,2         ; Este pin será utilizado como señal de escritura de direcciones en el RTC.

    movlw   .255            ; Se ponen en alto todos los pines del puerto E, con lo cual se deshabilitan las señales
    banksel PORTE           ; CS', RD' y WR', para que los buses (Datos y direcciones) estén libres.
    movwf   PORTE
    banksel TRISE
    clrf    TRISE           ; Los tres pines del puerto E serán utilizados como salidas (CS', RD' y WR').
```

```
        clrf    Addr_Bus_Hi
        clrf    Addr_Bus_Low
        bsf     PCLATH,3        ; Se selecciona la página 1 (para ejecutar la instrucción "Set_Address").
        call    Set_Address
        bcf     PCLATH,3        ; Se selecciona la página 0.

        bcf     Vpp             ; Transición necesaria para la inicialización y el correcto funcionamiento de la
        nop                     ; señal Vpp (pin RA0).
        bsf     Vpp

        bcf     RTC_Rd          ; Todas las señales del reloj de tiempo real son colocadas en bajo.
        bcf     RTC_Wr
        bcf     RTC_AWr
                                ;~~~~~~~~~~~~ Fin de la Inicialización de las señales de Control ~~~~~~~~~~~~


                                ;~~~~~~~~~~~~ Inicialización del Coprocesador Matemático ~~~~~~~~~~~~
Pak_Init

        bcf     Pak_Clk         ; Se pone en cero (0) la señal de reloj del Pak-IX.
        bcf     Pak_Rst         ; Deshabilitar el Pak (Reset).

        call    Bytes_Delay     ; En este caso, estos Delays son utilizados para dar un poco de tiempo en Reset.
        call    Bytes_Delay     ;
        call    Bytes_Delay     ;

        bsf     Pak_Rst         ; Fin del Reset del Pak.

        clrf    Ac1
        movlw   .25
        movwf   Ac2
EE_Delay:
        call    Bytes_Delay     ; En este caso, estos Delays son utilizados para esperar la finalización
        call    Bytes_Delay     ; de la ejecución del comando ERCL (ejecutado automáticamente al inicializar el Pak).
        call    Bytes_Delay     ;
        call    Bytes_Delay     ;
        decfsz  Ac1
        goto    EE_Delay
        decfsz  Ac2
        goto    EE_Delay
```

```
        call    Busy_Pak

        movlw   0x10            ; Comando OPT.
        call    Pak_Tx

        movlw   0x40            ; Rounding habilitado.
        call    Pak_Tx

                                ;~~~~~~~~~~~~ Fin de la Inicialización del Coprocesador Matemático ~~~~~~~~~~~


                                ;~~~~~~~~~~~~ Inicialización de los registros relacionados con los timers ~~~~~~~~~~~
        bsf     STATUS,IRP      ; Se seleccionan los bancos 2 y 3 para el direccionamiento indirecto.
        movlw   0x10
        movwf   FSR
Timers_RAM1_Clear:              ; Borrado del área de memoria para los timers (todos los registros de los bancos 2 y 3).
        clrf    INDF
        incf    FSR
        movf    FSR,W
        sublw   0x70
        btfss   STATUS,Z
        goto    Timers_RAM1_Clear
        movlw   0x90
        movwf   FSR
Timers_RAM2_Clear:
        clrf    INDF
        incf    FSR
        movf    FSR,W
        sublw   0xF0
        btfss   STATUS,Z
        goto    Timers_RAM2_Clear
                                ; Fin del borrado del área de memoria para los timers.

        banksel TMR0
        movlw   .100            ; Se carga el registro del timer0 con el valor 100 para lograr que el desborde se
        movwf   TMR0            ; provoque cada 998.4 microsegundos (con prescaler 1:32).

        banksel TMR1L
```

```
        movlw   0xDC            ; Se cargan los registros del timer1 con el valor 3,036 para lograr que el desborde
        movwf   TMR1L           ; se provoque cada 100.0000 milisegundos (con prescaler 1:8).
        movlw   0x0B
        movwf   TMR1H

        clrf    T_1m_Count      ; Se inicializa la cuenta de eventos de interrupción de 1 milisegundo.

        banksel T1CON
        bsf     T1CON,T1CKPS1           ; Se selecciona el prescaler con la relación 1:8.
        bsf     T1CON,T1CKPS0
        bsf     T1CON,TMR1ON; Se enciende el timer1.

                                ;~~~~~~~~ Fin de la Inicialización de los registros relacionados con los timers ~~~~~~~


                                ;~~~~~~~~~~~~ Inicialización de los registros relacionados con los Contadores ~~~~~~~~~~~~

        bcf     STATUS,IRP      ; Se seleccionan los bancos 0 y 1 para el direccionamiento indirecto.
        movlw   0xA0
        movwf   FSR
Counters_RAM1_Clear:            ; Borrado del área de memoria para los Contadores (registros 0xA0 - 0xEF, banco 1).
        clrf    INDF
        incf    FSR
        movf    FSR,W
        sublw   0xF0
        btfss   STATUS,Z
        goto    Counters_RAM1_Clear

                                ;~~~~~~~~ Fin de la Inicialización de los registros relacionados con los Contadores ~~~~~~~


                                ;~~ Inicialización de los bits para selección del modo de operación (en el módulo SP). ~~
                                ;  Se inicializan los pines IO6:IO5 para seleccionar el modo 'STOP' (IO6=0, IO5=1).
        banksel TXREG
        movlw   0x90            ; Comando 'SP WriteX'.
        movwf   TXREG
        call    Poll_Tx
```

```
        movlw   0x17            ; lsB de la dirección del pin IO6 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF            ; msB de la dirección del pin IO6 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x00
        movwf   TXREG           ; Se escribe el valor cero (0) en el pin IO6.
        call    Poll_Tx

        movlw   0x90            ; Comando 'SP WriteX'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x16            ; lsB de la dirección del pin IO5 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF            ; msB de la dirección del pin IO5 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x01
        movwf   TXREG           ; Se escribe el valor uno (1) en el pin IO5.
        call    Poll_Tx
                                ;~~~~~~ Fin de la Inicialización de los bits para selección del modo de operación ~~~~~~


        bcf     System_Is_Stopped;El sistema entrará al modo 'STOP', pero aún no está en él.

        clrf    Logic_Stack     ; Se inicializa la pila lógica con cero (0) en cada una de sus posiciones (8 en total)


                                ;~~~~~~~~ Inicialización de los registros para las instrucciones "Edge Up/Down" ~~~~~~~~

        clrf    Edge_Up_Counter         ; Se inicializan con cero (0) los contadores de instrucciones de detección de flancos.
        clrf    Edge_Down_Counter

        movlw   0x7E            ; Se direcciona el área de memoria para los registros de las instrucciones "Edge Up".
        movwf   Addr_Bus_Hi     ; (el penúltimo bloque de 256 bytes en el IC RAM de 32 KB).
        clrf    Addr_Bus_Low

Edge_Up_Area_Init_Loop:
```

```
        bsf     PCLATH,3        ; Se selecciona la página 1.
        call    Set_Address
        bsf     CS              ; Se selecciona el IC RAM.
        movlw   0x01
        call    IO_Write; Se inicializa con unos (1) el área de memoria para las instrucciones "Edge Up".
        bcf     PCLATH,3        ; Se selecciona la página 0.
        incfsz  Addr_Bus_Low
        goto    Edge_Up_Area_Init_Loop



        movlw   0x7F            ; Se direcciona el área de memoria para los registros de las instrucciones "Edge Down".
        movwf   Addr_Bus_Hi     ; (el último bloque de 256 bytes en el IC RAM de 32 KB).
        clrf    Addr_Bus_Low

Edge_Down_Area_Init_Loop:
        bsf     PCLATH,3        ; Se selecciona la página 1.
        call    Set_Address
        bsf     CS              ; Se selecciona el IC RAM.
        movlw   0x00
        call    IO_Write; Se inicializa con ceros (0) el área de memoria para las instrucciones "Edge Down".
        bcf     PCLATH,3        ; Se selecciona la página 0.
        incfsz  Addr_Bus_Low
        goto    Edge_Down_Area_Init_Loop

                                ;~~~ Fin de la Inicialización de los registros para las instrucciones "Edge Up/Down" ~~~


        clrf    Stack_Depth     ; No se han introducido datos en el Stack del PC, por lo que se hace 'Stack_Depth' = 0.
        clrf    Watchdog_Counter; Se borra el contador de vigilancia de tiempo de ciclo.
;       clrf    Rx_Ack          ; Se coloca en cero (0) la bandera de "acknowledge" de recepción USART.
        clrf    Scan_Time       ; Se coloca en cero (0) la variable de tiempo de ciclo.
        clrf    Scan_Clock      ; Se inicializa en bajo la señal de reloj del ciclo del PLC.
        movlw   0x8F
        movwf   PLC_Status      ; Estado inicial del PLC: Modo 'RUN', 0 errores.
        movlw   0xFF
        movwf   SM0
        movwf   SM1


                                ;~~ Inicialización del objeto Rx_Ack en el módulo SP. ~~
```

```
        banksel TXREG
        movlw   0x80            ; Comando 'SP Write'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x00            ; Dirección del objeto Rx_Ack en el módulo SP.
        movwf   TXREG
        call    Poll_Tx

;       movf    Rx_Ack,W
        movwf   TXREG           ; Se escribe el valor al objeto Rx_Ack.
        call    Poll_Tx
                                ;~~~~~~ Fin de la Inicialización del objeto Rx_Ack ~~~~~~

        movlw   0x0C            ; Se inicializa la dirección previa al primer objeto de monitoreo.
        movwf   Object_Address


                                ;~~~~~~ Lectura de los tipos de módulos contectados a la CPU central ~~~~~~

Read_Expansion_Modules_Connected

        banksel SPI_Command
        movlw   0x01
        movwf   SPI_Command     ; Se carga el comando de lectura de Tipo de módulo, con la dirección del módulo #1.

        movlw   0xD0            ; Se carga la dirección del primer objeto de Tipo del Módulo de Expansión.
        movwf   Module_Type_Obj_Addr

Expansion_Types_Loop:
        movf    SSPBUF,W        ; Se lee el buffer SPI para borrar la bandera de recepción.

        movf    SPI_Command,W
        movwf   SSPBUF                  ; Se envía el byte de comando para el módulo i.
        call    SPI_Poll ; (Byte # 1 del protocolo de la comunicación SPI).

        call    Module_First_Byte_Wait

        movlw   .0              ; Se envía el byte del # de elemento a acceder (irrelevante para este comando).
        movwf   SSPBUF                  ; (Byte # 2 del protocolo de la comunicación SPI).
        call    SPI_Poll
```

```
movlw   0x90            ; Comando 'SP WriteX'.
movwf   TXREG
call    Poll_Tx

movf    Module_Type_Obj_Addr,W      ; lsB de la dirección del objeto a escribir en el módulo SP.
movwf   TXREG
call    Poll_Tx

movlw   0x02            ; msB de la dirección del objeto a escribir.
movwf   TXREG
call    Poll_Tx

movwf   SSPBUF                  ; Lectura:  Se recibe el byte de Tipo del módulo direccionado.
call    SPI_Poll; (Byte # 3 del protocolo de la comunicación SPI).
movf    SPI_Byte_In,W
movwf   Module_Type

movf    Module_Type,W ; Tipo del módulo de expansión detectado.
movwf   TXREG         ; Se escribe el tipo en el objeto correspondiente.
call    Poll_Tx

call    Module_Last_Byte_Wait

incf    SPI_Command   ; Se direcciona el siguiente módulo de expansión.
incf    Module_Type_Obj_Addr

movlw   0x10
subwf   SPI_Command,W       ; Se verifica si ya se leyeron todos los módulos de expansión.
btfss   STATUS,Z
goto    Expansion_Types_Loop

                        ;~~~~~~ Fin de la lectura de los tipos de módulos contectados a la CPU central ~~~~~~


                        ;~~~~~~~~~~~~ Inicialización del sistema de interrupciones ~~~~~~~~~~~~
clrf    INTCON
banksel PIE1
bcf     PIE1,RCIE       ; NO se habilita la interrupción de recepción USART, solo se hará en el ciclo de 'STOP'.
bsf     PIE1,TMR1IE     ; Se habilita la interrupción de desborde en el timer1.
bsf     INTCON,T0IE     ; Se habilita la interrupción del overflow del timer0.
```

```
        bsf     INTCON,PEIE     ; Se desenmascaran las interrupciones de los periféricos.
        bsf     INTCON,GIE      ; Habilitación global de las interrupciones.

                                ;~~~~~~~~~~~ Fin de la Inicialización del sistema de interrupciones ~~~~~~~~~~~

        goto    SYSTEM_Cycle

                                ; Fin de la Rutina "SYSTEM_INIT".



        org     0x0800          ; Esta rutina se define al incio de la página 1 en la memoria de programa, con el
                                ; propósito de organizar de forma adecuada todas las rutinas de instrucciones
                                ; del PLC, y separarlas del resto de rutinas del Sistema completo que se encuentran en
                                ; la página 0.

Execute_Program                 ; Esta rutina se encarga de la ejecución del programa almacenado en la memoria FLASH,
                                ; que ha sido cargado desde la página Web.

        banksel Mode_Flags      ; Se selecciona el banco estándar (banco 0).

        bcf     In_System_Monitor       ; El Sistema NO está en la fase de monitoreo.

        clrf    PC_Hi           ; Se inicializa el Program Counter (PC) con la posición 0x0000.
        clrf    PC_Low

        clrf    Edge_Up_Counter         ; Se inicializan con cero (0) los contadores de instrucciones de detección de flancos.
        clrf    Edge_Down_Counter

Networks_Loop:
        clrf    Logic_Stack     ; Se borra la pila lógica para iniciar la ejecución de un circuito nuevo.
Instructions_Loop:

        movlw   0x08            ; Se utiliza direccionamiento relativo para la ejecución de las rutinas de instrucciones
        movwf   PCLATH          ; de las funciones del PLC.  El direccionamiento relativo es realizado en el primer bloque
                                ; de la página 1 (0x0800).

        banksel Mode_Flags      ; Se selecciona el banco estándar (banco 0).
```

```
call    Get_FLASH_Data          ; Se lee la instrucción a ejecutar, almacenada en el IC FLASH.

movwf   Instruction_Full; Se crea una copia del byte de instrucción para almacenar el indicador de parámetro
movwf   Instruction_Code; constante, si éste existe o no (<D7>).
movlw   0x7F            ; Se extrae el código de la instrucción, y se almacena en 'Instruction_Code'.
andwf   Instruction_Code

movlw   0x7D            ; Si la instrucción es 0x7D, entonces se alcanzó el final del programa.
subwf   Instruction_Code,W
btfsc   STATUS,Z

retum

movf    Instruction_Code,W
addwf   PCL
nop
goto    LD              ; Instrucción Load Bit con el código 0x01.
goto    A               ; 0x02
goto    O               ; 0x03
goto    LDN             ; 0x04
goto    AN              ; 0x05
goto    ON              ; 0x06
goto    NOT             ; 0x07
goto    EU              ; 0x08
goto    ED              ; 0x09
goto    ALD             ; 0x0A
goto    OLD             ; 0x0B
goto    LPS             ; 0x0C
goto    LRD             ; 0x0D
goto    LPP             ; 0x0E
goto    OUTPUT                  ; 0x0F
goto    SET_Bits        ; 0x10
goto    RESET_Bits      ; 0x11
goto    NOTH            ; 0x12
goto    LD_B_Equal      ; 0x13
goto    LD_W_Equal      ; 0x14
goto    LD_D_Equal      ; 0x15
goto    LD_R_Equal      ; 0x16
goto    LD_B_Greater    ; 0x17
goto    LD_W_Greater    ; 0x18
```

```
goto    LD_D_Greater    ; 0x19
goto    LD_R_Greater    ; 0x1A
goto    LD_B_Less       ; 0x1B
goto    LD_W_Less       ; 0x1C
goto    LD_D_Less       ; 0x1D
goto    LD_R_Less       ; 0x1E
goto    BTI             ; 0x1F
goto    BTD             ; 0x20
goto    BTR             ; 0x21
goto    ITD             ; 0x22
goto    ITR             ; 0x23
goto    DTI             ; 0x24
goto    DTR             ; 0x25
goto    RTB             ; 0x26
goto    RTI             ; 0x27
goto    RTD             ; 0x28
goto    CTU             ; 0x29
goto    TONR            ; 0x2A
goto    TRESET          ; 0x2B
goto    INVB            ; 0x2C
goto    INVW            ; 0x2D
goto    INVD            ; 0x2E
goto    ANDB            ; 0x2F
goto    ANDW            ; 0x30
goto    ANDD            ; 0x31
goto    ORB             ; 0x32
goto    ORW             ; 0x33
goto    ORD             ; 0x34
goto    XORB            ; 0x35
goto    XORW            ; 0x36
goto    XORD            ; 0x37
goto    MOVB            ; 0x38
goto    MOVW            ; 0x39
goto    MOVDx           ; 0x3A
goto    MOVR            ; 0x3B
goto    SHLB            ; 0x3C
goto    SHRB            ; 0x3D
goto    SHLW            ; 0x3E
goto    SHRW            ; 0x3F
goto    SHLD            ; 0x40
goto    SHRD            ; 0x41
```

```
goto    RLB             ; 0x42
goto    RRB             ; 0x43
goto    RLW             ; 0x44
goto    RRW             ; 0x45
goto    RLD             ; 0x46
goto    RRD             ; 0x47
goto    INCB            ; 0x48
goto    DECB            ; 0x49
goto    INCWx           ; 0x4A
goto    DECWx           ; 0x4B
goto    INCDx           ; 0x4C
goto    DECDx           ; 0x4D
goto    ADD             ; 0x4E
goto    SUB             ; 0x4F
goto    MULT            ; 0x50
goto    DIV             ; 0x51
goto    CHS             ; 0x52
goto    ABS             ; 0x53
goto    SQRT            ; 0x54
goto    LOG             ; 0x55
goto    LOG10           ; 0x56
goto    EXP             ; 0x57
goto    EXP10           ; 0x58
goto    POW             ; 0x59
goto    ROOT            ; 0x5A
goto    RECIP           ; 0x5B
goto    SIN             ; 0x5C
goto    ASIN            ; 0x5D
goto    COS             ; 0x5E
goto    ACOS            ; 0x5F
goto    TAN             ; 0x60
goto    ATAN            ; 0x61
goto    JMPx            ; 0x62
goto    CALLSBRx        ; 0x63
goto    RETx            ; 0x64
goto    CRETx           ; 0x65
goto    ENDPx           ; 0x66
goto    STOPx           ; 0x67
goto    WDTRx           ; 0x68
goto    RTCRDx          ; 0x69
goto    RTCWRx                  ; 0x6A
```

```
                        ; Fin de la rutina "Execute_Program".          Banco 0.


IO_Read                                 ; Esta rutina lee un byte del bus de datos, desde el IC actualmente seleccionado
                        ; con el pin CS'.  El byte leído es almacenado en el registro W.  Se utiliza la
                        ; dirección actual en el bus de direcciones.

        btfss   CS              ; Se determina si se pretende leer del IC RAM.
        goto    External_Read


Check_Analog_Read_Range:        ; Se determina si se desea acceder al área de entradas analógicas (mapeadas en el rango
        movlw   0x02            ; 0x0200 - 0x02FF de la RAM externa).
        subwf   Addr_Bus_Hi,W
        btfss   STATUS,Z
        goto    Read_Range_Check; NO se desea acceder a una entrada analógica.  Se realizará la sig. revisión de área.
        goto    Analog_Item_Read; SI se desea acceder a una entrada analógica.


Read_Range_Check:               ; Puesto que SI se pretende leer del IC RAM, entonces se revisará el rango al cual se
        movlw   0x02            ; desea acceder, para determinar si se accederá al IC RAM externo o a la RAM interna, la
        subwf   Addr_Bus_Hi,W   ; cual posee mapeadas las áreas de memoria I,Q,C y T (primeros 512 bytes en el IC RAM).
        btfsc   STATUS,C        ; Si el resultado no es negativo, entonces se debe acceder al IC RAM (RAM externa).
        goto    External_Read
Internal_Read:                  ; Puesto que el resultado SI es negativo, entonces se debe acceder a la RAM interna.
        bcf     STATUS,IRP      ; Se asume que la localidad RAM interna a acceder se encuentra en el banco 0.
        btfsc   Addr_Bus_Hi,0   ; Si el bit Addr_Bus_Hi <0> es uno, entonces la localidad de memoria RAM interna a
        bsf     STATUS,IRP      ; acceder se encuentra en el banco 2/3, por lo que se hace IRP=1.
        movf    Addr_Bus_Low,W
        movwf   FSR             ; Se carga en 'FSR' el byte bajo de la dirección de la localidad RAM interna a acceder.
        movf    INDF,W          ; Se lee el valor de la localidad RAM interna, y se almacena en el registro W.

        return


External_Read:                  ; Este segmento se ejecuta solo cuando la lectura sea desde la RAM/FLASH externa.
```

```
        movlw   .255
        banksel TRISD
        movwf   TRISD           ; Se coloca al puerto D en modo de entrada de datos.
        banksel PORTE
        bcf     IORD            ; Se lee el byte de entrada y se almacena en el registro W.
        movf    PORTD,W
        bsf     IORD

        return
                                ; Fin de la rutina "IO_Read".                   Banco 0.


IO_Write                ; Esta rutina coloca el contenido del registro W en el bus de datos y lo escribe en el
                        ; IC actualmente seleccionado por medio del pin CS', utilizando la dirección actual en
                        ; el bus de direcciones.

        movwf   Temp            ; Se almacena temporalmente el dato a escribir.

        btfss   CS              ; Se determina si se pretende escribir al IC RAM.
        goto    External_Write


Check_Analog_Write_Range:       ; Se determina si se desea acceder al área de salidas analógicas (mapeadas en el rango
        movlw   0x03            ; 0x0300 - 0x03FF de la RAM externa).
        subwf   Addr_Bus_Hi,W
        btfss   STATUS,Z
        goto    Write_Range_Check; NO se desea acceder a una salida analógica.  Se realizará la sig. revisión de área.
        goto    Analog_Item_Write; SI se desea acceder a una salida analógica.


Write_Range_Check:              ; Puesto que SI se pretende escribir al IC RAM, entonces se revisará el rango al cual se
        movlw   0x02            ; desea acceder, para determinar si se accederá al IC RAM externo o a la RAM interna, la
        subwf   Addr_Bus_Hi,W   ; cual posee mapeadas las áreas de memoria I,Q,C y T (primeros 512 bytes en el IC RAM).
        btfsc   STATUS,C        ; Si el resultado no es negativo, entonces se debe acceder al IC RAM (RAM externa).
        goto    External_Write
Internal_Write:                 ; Puesto que el resultado SI es negativo, entonces se debe acceder a la RAM interna.
        bcf     STATUS,IRP      ; Se asume que la localidad RAM interna a acceder se encuentra en el banco 0.
        btfsc   Addr_Bus_Hi,0   ; Si el bit Addr_Bus_Hi <0> es uno, entonces la localidad de memoria RAM interna a
        bsf     STATUS,IRP      ; acceder se encuentra en el banco 2/3, por lo que se hace IRP=1.
```

```
        movf    Addr_Bus_Low,W
        movwf   FSR             ; Se carga en 'FSR' el byte bajo de la dirección de la localidad RAM interna a acceder.
        movf    Temp,W          ; Se escribe el valor en la localidad RAM interna.
        movwf   INDF

        return


External_Write:                 ; Este segmento se ejecuta solo cuando la escritura sea hacia la RAM/FLASH externa.
        banksel TRISD
        clrf    TRISD           ; Se coloca al puerto D en modo de salida de datos.
        banksel PORTD
        movf    Temp,W
        movwf   PORTD           ; Se escribe el byte de salida en el bus de datos.
        bcf     IOWR            ; Se escribe el byte en el IC seleccionado.
        bsf     IOWR

        return
                                ; Fin de la rutina "IO_Write".                  Banco 0.


Set_Address                     ; Esta rutina escribe los registros de direccionamiento 'Addr_Bus_Low' y 'Addr_Bus_Hi'
                                ; en el bus de direcciones.

        banksel Addr_Bus_Hi
        movf    Addr_Bus_Hi,W   ; Se coloca el MSB de la dirección en el latch del puerto D.
        movwf   PORTD
        banksel TRISD
        clrf    TRISD           ; Se coloca el MSB de la dirección en el bus de datos.
        banksel PORTA
        bcf     Clk_Hi          ; Se coloca el MSB de la dirección en el bus de direcciones.
        nop
        nop
        bsf     Clk_Hi

        movf    Addr_Bus_Low,W          ; Se coloca el LSB de la dirección en el bus de datos.
        movwf   PORTD
        bcf     Clk_Low         ; Se coloca el LSB de la dirección en el bus de direcciones.
```

```
        nop
        nop
        bsf     Clk_Low

        return
                                ; Fin de la rutina "Set_Address".       Banco 0.


Put_RAM_Data                    ; Esta rutina escribe un byte en la memoria RAM externa.  Se asume que el dato a
                                ; escribir se encuentra en W.  Cada escritura incrementa automáticamente la dirección.
        movwf   Data_To_RAM
        incf    Addr_Bus_Low
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi
        call    Set_Address
        movf    Data_To_RAM,W
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write; Se escribe el dato almacenado en 'Data_To_RAM'.

        return
                                ; Fin de la rutina "Put_RAM_Data".      Banco 0.


Get_RAM_Data                    ; Esta rutina lee un byte de la memoria RAM externa.  El byte es almacenado en W.
                                ; Cada lectura incrementa automáticamente la dirección.
        incf    Addr_Bus_Low
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi
        call    Set_Address
        bsf     CS
        call    IO_Read

        return
                                ; Fin de la rutina "Get_RAM_Data".      Banco 0.
```

```
Get_FLASH_Data                  ; Esta rutina lee un byte de la memoria FLASH, en la localidad direccionada por el PC.
                                ; Cada lectura incrementa automáticamente el PC.
        banksel PC_Low

        incf    PC_Low          ; Se incrementa la dirección del PC.
        btfsc   STATUS,Z        ; Se verifica si se debe incrementar el byte alto del PC.
        incf    PC_Hi

        banksel PORTD
        movf    PC_Hi,W                 ; Se coloca el MSB del PC latch del puerto D.
        movwf   PORTD
        banksel TRISD
        clrf    TRISD           ; Se coloca el MSB del PC en el bus de datos.
        banksel PORTA
        bcf     Clk_Hi          ; Se coloca el MSB del PC en el bus de direcciones.
        nop
        nop
        bsf     Clk_Hi

        movf    PC_Low,W        ; Se coloca el LSB del PC en el bus de datos.
        movwf   PORTD
        bcf     Clk_Low         ; Se coloca el LSB del PC en el bus de direcciones.
        nop
        nop
        bsf     Clk_Low

        bcf     CS              ; Se selecciona el IC FLASH.

        call    IO_Read                 ; Se lee el byte en la localidad direccionada por el PC, y se almacena en W.

        return

                                ; Fin de la rutina "Get_FLASH_Data".            Banco 0.


Get_Arg_Val                     ; Rutina que lee el valor del argumento de una instrucción.  La dirección del argumento
                                ; se encuentra almacenada en los dos bytes siguientes al código de la instrucción.
                                ; El valor es almacenado en el registro ByteX ubicado en el banco 0 de la RAM interna.
```

```
; *** Esta rutina provoca la utilización del stack del PIC16F877 hasta el nivel 3 (de 8) ***.

        call    Get_FLASH_Data
        movwf   Addr_Bus_Low    ; Se almacena el lsB de la dirección de la localidad en RAM que posee el byte de interés.
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi     ; Se almacena el msB de la dirección de la localidad en RAM que posee el byte de interés.
        call    Set_Address     ; Se coloca la dirección en el bus de direcciones del Sistema.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Read                 ; Se lee el byte que posee el bit que se cargará en la pila lógica.
        movwf   Bytex

        return
                                ; Fin de la rutina "Get_Arg_Val".       Banco 0.



Analog_Item_Read                ; Rutina que lee el canal analógico direccionado.

        banksel Addr_Bus_Low
        movf    Addr_Bus_Low,W
        movwf   Temp
        rrf     Temp            ; Se extraerá el # del módulo a acceder (se encuentra en Addr_Bus_Low <7:4>).
        rrf     Temp
        rrf     Temp
        rrf     Temp
        movlw   0x0F
        andwf   Temp,W          ; Ahora W posee el # de módulo a acceder.
        addlw   0x30            ; Ahora W posee el código del comando y el # de módulo a acceder.

        bcf     PCLATH,3        ; Se selecciona la página 0.

        movwf   SSPBUF                  ; Se envía el comando 'Read Analog Input', con el # de módulo correspondiente.
        call    SPI_Poll

        call    Module_First_Byte_Wait

        movf    Addr_Bus_Low,W
        andlw   0x1F            ; Ahora W posee el # de entrada analógica a acceder.
        movwf   SSPBUF                  ; Se envía el # de entrada analógica a acceder.
```

```
        call    Test_Busy_Module; Se espera a que el módulo notifique que el resultado está listo.

        movwf   SSPBUF                  ; Lectura:  Se lee el valor del canal analógico.
        call    SPI_Poll

        call    Module_Last_Byte_Wait

        movf    SPI_Byte_In,W

        bsf     PCLATH,3        ; Se selecciona la página 1.

        return
                                ; Fin de la rutina "Analog_Item_Read".          ; Banco 0.


Analog_Item_Write               ; Rutina que escribe al canal analógico direccionado.

        banksel Addr_Bus_Low
        movf    Addr_Bus_Low,W
        movwf   Temp2
        rrf     Temp2           ; Se extraerá el # del módulo a acceder (se encuentra en Addr_Bus_Low<7:4>).
        rrf     Temp2
        rrf     Temp2
        rrf     Temp2
        movlw   0x0F
        andwf   Temp2,W                 ; Ahora W posee el # de módulo a acceder.
        addlw   0x40            ; Ahora W posee el código del comando y el # de módulo a acceder.

        bcf     PCLATH,3        ; Se selecciona la página 0.

        movwf   SSPBUF                  ; Se envía el comando 'Write Analog Output', con el # de módulo correspondiente.
        call    SPI_Poll

        call    Module_First_Byte_Wait

        movf    Addr_Bus_Low,W
        andlw   0x0F            ; Ahora W posee el # de salida analógica a acceder.
        movwf   SSPBUF                  ; Se envía el # de salida analógica a acceder.
        call    SPI_Poll
```

```
        movf    Temp,W
        rnovwf  SSPBUF                  ; Se escribe a la salida analógica.
        call    Test_Busy_Module; Se espera la notificación de escritura exitosa.

        bsf     PCLATH,3        ; Se selecciona la página 1.

        return
                                ; Fin de la rutina "Analog_Item_Write".         ; Banco 0.


BOOLE_B                                 ; Rutina utilizada para la realización de operaciones lógicas con bytes.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    BOOLE_B_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_B_Const        ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x05                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cinco (5) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_B_Const:
        call    Get_FLASH_Data          ; Se lee la constante (primer parámetro).
        movwf   Byte1

        goto    BOOLE_B_Arg2


BOOLE_B_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_B_No_Const ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
```

```
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_B_No_Const:
        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1

BOOLE_B_Arg2:
        call    Get_Arg_Val             ; Se lee el segundo parámetro.
        movf    ByteX,W
        movwf   Byte2

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        movf    BOOLE_Code,W
        sublw   0x01                    ; Se determina si se desea realizar una operación OR.
        btfsc   STATUS,Z
        goto    Do_ORB

        movf    BOOLE_Code,W
        sublw   0x02                    ; Se determina si se desea realizar una operación XOR.
        btfsc   STATUS,Z
        goto    Do_XORB

Do_ANDB:
        movf    Byte1,W                 ; Se realiza la operación lógica AND entre los dos parámetros.
        andwf   Byte2
        goto    BOOLE_B_End

Do_ORB:
        movf    Byte1,W                 ; Se realiza la operación lógica OR entre los dos parámetros.
        iorwf   Byte2
        goto    BOOLE_B_End
```

```
Do_XORB:
        movf    Byte1,W                 ; Se realiza la operación lógica XOR entre los dos parámetros.
        xorwf   Byte2
        goto    BOOLE_B_End

BOOLE_B_End:
        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte2,W
        call    IO_Write        ; Se escribe el resultado.

        goto    Instructions_Loop
                                        ; Fin de la rutina "BOOLE_B".           Banco 0.



BOOLE_W                                 ; Rutina utilizada para la realización de operaciones lógicas con Words.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    BOOLE_W_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_W_Const        ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_W_Const:
        call    Get_FLASH_Data          ; Se lee la constante (primer parámetro).
        movwf   Byte1
        call    Get_FLASH_Data
        movwf   Byte2

        goto    BOOLE_W_Arg2


BOOLE_W_No_Const:
```

```
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_W_No_Const ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_W_No_Const:
        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2

BOOLE_W_Arg2:
        call    Get_Arg_Val             ; Se lee el segundo parámetro.
        movf    ByteX,W
        movwf   Byte3
        call    Get_RAM_Data
        movwf   Byte4

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        movf    BOOLE_Code,W
        sublw   0x01                    ; Se determina si se desea realizar una operación OR.
        btfsc   STATUS,Z
        goto    Do_ORW

        movf    BOOLE_Code,W
        sublw   0x02                    ; Se determina si se desea realizar una operación XOR.
        btfsc   STATUS,Z
        goto    Do_XORW

Do_ANDW:
```

```
        movf    Byte1,W             ; Se realiza la operación lógica AND entre los dos parámetros.
        andwf   Byte3
        movf    Byte2,W
        andwf   Byte4
        goto    BOOLE_W_End

Do_ORW:
        movf    Byte1,W             ; Se realiza la operación lógica OR entre los dos parámetros.
        iorwf   Byte3
        movf    Byte2,W
        iorwf   Byte4
        goto    BOOLE_W_End

Do_XORW:
        movf    Byte1,W             ; Se realiza la operación lógica XOR entre los dos parámetros.
        xorwf   Byte3
        movf    Byte2,W
        xorwf   Byte4
        goto    BOOLE_W_End

BOOLE_W_End:
        bsf     CS                  ; Se selecciona el IC RAM.
        movf    Byte3,W
        call    IO_Write        ; Se escribe el resultado.
        movf    Byte4,W
        call    Put_RAM_Data

        goto    Instructions_Loop

                                    ; Fin de la rutina "BOOLE_W".          Banco 0.


BOOLE_D                             ; Rutina utilizada para la realización de operaciones lógicas con Doubles.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    BOOLE_D_No_Const

        btfsc   Logic_Stack,7       ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_D_Const    ; SI existe flujo, por lo que se ejecutará la instrucción.
```

```
        movlw   0x08                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en ocho (8) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retoma al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_D_Const:
        call    Get_FLASH_Data          ; Se lee la constante (primer parámetro).
        movwf   Byte1
        call    Get_FLASH_Data
        movwf   Byte2
        call    Get_FLASH_Data
        movwf   Byte3
        call    Get_FLASH_Data
        movwf   Byte4

        goto    BOOLE_D_Arg2


BOOLE_D_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_BOOLE_D_No_Const ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_BOOLE_D_No_Const:
        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3
        call    Get_RAM_Data
```

```
        movwf   Byte4

BOOLE_D_Arg2:
        call    Get_Arg_Val             ; Se lee el segundo parámetro.
        movf    ByteX,W
        movwf   Byte5
        call    Get_RAM_Data
        movwf   Byte6
        call    Get_RAM_Data
        movwf   Byte7
        call    Get_RAM_Data
        movwf   Byte8

        call    Get_FLASH_Data          ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        movf    BOOLE_Code,W
        sublw   0x01                    ; Se determina si se desea realizar una operación OR.
        btfsc   STATUS,Z
        goto    Do_ORD

        movf    BOOLE_Code,W
        sublw   0x02                    ; Se determina si se desea realizar una operación XOR.
        btfsc   STATUS,Z
        goto    Do_XORD

Do_ANDD:
        movf    Byte1,W                 ; Se realiza la operación lógica AND entre los dos parámetros.
        andwf   Byte5
        movf    Byte2,W
        andwf   Byte6
        movf    Byte3,W
        andwf   Byte7
        movf    Byte4,W
        andwf   Byte8
        goto    BOOLE_D_End

Do_ORD:
```

```
        movf    Byte1,W             ; Se realiza la operación lógica OR entre los dos parámetros.
        iorwf   Byte5
        movf    Byte2,W
        iorwf   Byte6
        movf    Byte3,W
        iorwf   Byte7
        movf    Byte4,W
        iorwf   Byte8
        goto    BOOLE_D_End

Do_XORD:
        movf    Byte1,W             ; Se realiza la operación lógica XOR entre los dos parámetros.
        xorwf   Byte5
        movf    Byte2,W
        xorwf   Byte6
        movf    Byte3,W
        xorwf   Byte7
        movf    Byte4,W
        xorwf   Byte8
        goto    BOOLE_D_End

BOOLE_D_End:
        bsf     CS                  ; Se selecciona el IC RAM.
        movf    Byte5,W
        call    IO_Write        ; Se escribe el resultado.
        movf    Byte6,W
        call    Put_RAM_Data
        movf    Byte7,W
        call    Put_RAM_Data
        movf    Byte8,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                    ; Fin de la rutina "BOOLE_D".          Banco 0.


Shift_Rotate_B                      ; Rutina utilizada para Desplazar y Rotar bytes.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
```

```
        goto    SR_B_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_B_Const           ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x05                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cinco (5) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_SR_B_Const:
        call    Get_FLASH_Data                  ; Se lee la constante 'n'.
        movwf   Byte1

        goto    SR_B_In_Out


SR_B_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_B_No_Const        ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retoma al ciclo de ejecución de las demás instrucciones del programa.

Do_SR_B_No_Const:
        call    Get_Arg_Val             ; Se lee el parámetro variable 'n'.
        movf    ByteX,W
        movwf   Byte1

SR_B_In_Out:
        call    Get_Arg_Val             ; Se lee el parámetro de entrada.
        movf    ByteX,W
        movwf   Byte2

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
```

```
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

SR_B_Loop:
        btfss   Shift_Rotate_Code,0     ; Se determina si se debe realizar la operación 'Shift' o la operación 'Rotate'.
        bcf     STATUS,C                ; Se debe realizar la operación 'Shift'.  Se hace 'Carry' = 0.

        btfss   Direction_Code,0 ; Se determina si el sentido es 'Left' o 'Right'.
        rlf     Byte2                   ; El sentido es 'Left'.
        btfsc   Direction_Code,0
        rrf     Byte2                   ; El sentido es 'Right'.

        decfsz  Byte1
        goto    SR_B_Loop


        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte2,W
        call    IO_Write        ; Se escribe el resultado.

        goto    Instructions_Loop
                                        ; Fin de la rutina "Shift_Rotate_B".  ; Banco 0.



Shift_Rotate_W                          ; Rutina utilizada para Desplazar y Rotar Words.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    SR_W_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_W_Const           ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x05                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cinco (5) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi
```

```
        goto    Instructions_Loop; Se retoma al ciclo de ejecución de las demás instrucciones del programa.

Do_SR_W_Const:
        call    Get_FLASH_Data          ; Se lee la constante 'n'.
        movwf   Byte1

        goto    SR_W_In_Out


SR_W_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_W_No_Const        ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_SR_W_No_Const:
        call    Get_Arg_Val             ; Se lee el parámetro variable 'n'.
        movf    ByteX,W
        movwf   Byte1

SR_W_In_Out:
        call    Get_Arg_Val             ; Se lee el parámetro de entrada.
        movf    ByteX,W
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3

        call    Get_FLASH_Data          ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

SR_W_Loop:
        btfss   Shift_Rotate_Code,0     ; Se determina si se debe realizar la operación 'Shift' o la operación 'Rotate'.
        bcf     STATUS,C                ; Se debe realizar la operación 'Shift'.  Se hace 'Carry' = 0.
```

```
        btfss   Direction_Code,0 ; Se determina si el sentido es 'Left' o 'Right'.
        rlf     Byte2                   ; El sentido es 'Left'.
        btfss   Direction_Code,0
        rlf     Byte3

        btfsc   Direction_Code,0
        rrf     Byte2                   ; El sentido es 'Right'.
        btfsc   Direction_Code,0
        rrf     Byte3

        decfsz  Byte1
        goto    SR_W_Loop


        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte2,W
        call    IO_Write        ; Se escribe el resultado.
        movf    Byte3,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                        ; Fin de la rutina "Shift_Rotate_W". ; Banco 0.


Shift_Rotate_D                          ; Rutina utilizada para Desplazar y Rotar Doubles.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    SR_D_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_D_Const           ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x05                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cinco (5) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.
```

```
Do_SR_D_Const:
        call    Get_FLASH_Data          ; Se lee la constante 'n'.
        movwf   Byte1

        goto    SR_D_In_Out


SR_D_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_SR_D_No_Const        ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retoma al ciclo de ejecución de las demás instrucciones del programa.

Do_SR_D_No_Const:
        call    Get_Arg_Val             ; Se lee el parámetro variable 'n'.
        movf    ByteX,W
        movwf   Byte1

SR_D_In_Out:
        call    Get_Arg_Val             ; Se lee el parámetro de entrada.
        movf    ByteX,W
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3
        call    Get_RAM_Data
        movwf   Byte4
        call    Get_RAM_Data
        movwf   Byte5

        call    Get_FLASH_Data          ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address
```

```
SR_D_Loop:
        btfss   Shift_Rotate_Code,0     ; Se determina si se debe realizar la operación 'Shift' o la operación 'Rotate'.
        bcf     STATUS,C                ; Se debe realizar la operación 'Shift'.  Se hace 'Carry' = 0.

        btfss   Direction_Code,0 ; Se determina si el sentido es 'Left' o 'Right'.
        goto    Dir_Left         ; El sentido es 'Left'.

Dir_Right:
        rrf     Byte2                   ; El sentido es 'Right'.
        rrf     Byte3
        rrf     Byte4
        rrf     Byte5

        decfsz  Byte1
        goto    SR_D_Loop

        goto    SR_End

Dir_Left:
        rlf     Byte2                   ; El sentido es 'Left'.
        rlf     Byte3
        rlf     Byte4
        rlf     Byte5

        decfsz  Byte1
        goto    SR_D_Loop

SR_End:
        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte2,W
        call    IO_Write        ; Se escribe el resultado.
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data
        movf    Byte5,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                        ; Fin de la rutina "Shift_Rotate_D".  ; Banco 0.
```

```
RTX                                     ; Rutina "Real to Anything".

        call    Get_Arg_Val             ; Se lee el valor del real a convertir.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3
        call    Get_RAM_Data
        movwf   Byte4

        call    Get_FLASH_Data          ; Se lee el lsB de la dirección donde se colocará el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data          ; Se lee el msB de la dirección donde se colocará el resultado.
        movwf   Addr_Bus_Hi

        call    Set_Address             ; Se direcciona la posición del lsB del resultado.
        bsf     CS                      ; Se selecciona el IC RAM.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        movlw   0x01                    ; Comando LOADX.
        call    Pak_Tx

        movf    Byte4,W                 ; msB del # real que se desea convertir.
        call    Pak_Tx

        movf    Byte3,W
        call    Pak_Tx

        movf    Byte2,W
        call    Pak_Tx

        movf    Byte1,W                 ; lsB del # real que se desea convertir.
        call    Pak_Tx

        movlw   0x09                    ; Comando FPCVT (convierte X de IEEE745 a Pak).
```

```
call    Pak_Tx

movlw   0x05            ; Comando DIGIT.
call    Pak_Tx

movlw   0x00            ; Parámetro 'SIGN'.
call    Pak_Tx

call    Pak_Rx          ; Se recibe el caracter que indicará el signo del número real original.
movf    Receive_Pak,W
movwf   Real_Sign

movlw   0x0B            ; Comando INT (convierte X de Pak a entero de 24 bits).
call    Pak_Tx

call    Pak_Rx          ; Lectura del byte STATUS de retorno del comando INT.
movf    Receive_Pak,W
btfss   STATUS,Z        ; Se determina si sucedió algún error en la operación matemática.
goto    Math_Error      ; SI ocurrió un error matemático.

movlw   0x03            ; Comando READX.
call    Pak_Tx

call    Pak_Rx          ; Se recibe el número Long (se deben recibir los 4 bytes del Pak).
movf    Receive_Pak,W
movwf   Byte4           ; msB del resultado (carece de significado en esta instrucción).
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte3           ; msB del número Double.
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte2           ; msB del número entero.
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte1           ; Byte único para la conversión de # real a byte.

bsf     PCLATH,3        ; Se selecciona la página 1.


movlw   0x00            ; Se revisa si se desea convertir a un número Long.
subwf   RTX_Code,W
```

```
        btfsc   STATUS,Z
        goto    Real_To_Double

        movlw   0x01                    ; Se revisa si se desea convertir a un número entero.
        subwf   RTX_Code,W
        btfsc   STATUS,Z
        goto    Real_To_Int

        movlw   0x02                    ; Se revisa si se desea convertir a un número byte.
        subwf   RTX_Code,W
        btfsc   STATUS,Z
        goto    Real_To_Byte

Real_To_Double:
        bsf     Byte3,7                 ; Se asume que el # real era positivo.
        movlw   0x2D
        subwf   Real_Sign               ; Se determina si se recibió el caracter ASCII 0x2D (signo negativo).
        btfsc   STATUS,Z
        bcf     Byte3,7

        movf    Byte1,W                 ; lsB del # Double.
        call    IO_Write        ; Se escribe el # Long en el IC RAM.
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data            ; msB del # Long.

        return


Real_To_Int:
        bsf     Byte2,7                 ; Se asume que el # real era positivo.
        movlw   0x2D
        subwf   Real_Sign               ; Se determina si se recibió el caracter ASCII 0x2D (signo negativo).
        btfsc   STATUS,Z
        bcf     Byte2,7

        movf    Byte1,W                 ; lsB del # entero.
        call    IO_Write        ; Se escribe el # entero en el IC RAM.
```

```
        movf    Byte2,W
        call    Put_RAM_Data        ; msB del # entero.

        return

Real_To_Byte:
        movf    Byte1,W
        call    IO_Write        ; Se escribe el byte en el IC RAM.

        return

                                ; Fin de la rutina "RTX".       ; Banco 0.


Process_Reals_To_Compare        ; Rutina que carga 2 números reales, y los resta, para luego determinar si son
                                ; iguales, o si A es mayor o menor que B.

        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   0x01            ; Comando LOADX (Se asume que los bytes YA están en formato IEEE745).
        call    Pak_Tx

        movf    Byte4,W
        call    Pak_Tx

        movf    Byte3,W
        call    Pak_Tx

        movf    Byte2,W
        call    Pak_Tx

        movf    Byte1,W
        call    Pak_Tx

        movlw   0x09            ; Comando FPCVT (convierte X de IEEE745 a Pak).
        call    Pak_Tx

        movlw   0x02            ; Comando LOADY.
```

```
call    Pak_Tx

movf    Byte8,W
call    Pak_Tx

movf    Byte7,W
call    Pak_Tx

movf    Byte6,W
call    Pak_Tx

movf    Byte5,W
call    Pak_Tx

movlw   0x04            ; Comando SWAP (para luego convertir Y de IEEE745 a Pak).
call    Pak_Tx

movlw   0x09            ; Comando FPCVT (convierte X de IEEE745 a Pak).
call    Pak_Tx

movlw   0x0E            ; Comando SUB (con STATUS).
call    Pak_Tx

call    Pak_Rx          ; Lectura del byte STATUS de retorno del comando SUB.
movf    Receive_Pak,W
btfss   STATUS,Z        ; Se determina si sucedió algún error en la operación matemática.
goto    Math_Error      ; SI ocurrió un error matemático.

movlw   0x05            ; Comando DIGIT.
call    Pak_Tx

movlw   0x00            ; Parámetro 'SIGN'.
call    Pak_Tx

call    Pak_Rx          ; Se recibe el caracter que indicará el resultado de la comparación entre los dos
                        ; números reales.

bsf     PCLATH,3        ; Se selecciona la página 1.

return
                        ; Fin de la rutina "Process_Reals_To_Compare".      ; Banco 0.
```

```
INC_DEC_B                       ; Rutina utilizada por las instrucciones de Incremento/Decremento de bytes.

        btfsc   Logic_Stack,7   ; Se verifica si el tope de la pila está en uno.
        goto    Do_Inc_Dec_B    ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x02            ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low          ; Debe incrementarse el PC en dos (2) posiciones (contienen parámetros de
        btfsc   STATUS,C        ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_Inc_Dec_B:
        call    Get_Arg_Val     ; Se lee el dato que se desea incrementar/decrementar.
        movf    ByteX,W
        movwf   Byte1

        btfsc   Inc_Dec_Code,0  ; Se determina si se desea incrementar o decrementar.
        goto    Dec_Data_B      ; Se desea decrementar.

Inc_Data_B:
        incf    Byte1,W
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write

        goto    Instructions_Loop

Dec_Data_B:
        decf    Byte1,W
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write        ; Se escribe el valor actualizado (no se cambió de dirección en el bus).

        goto    Instructions_Loop

                                ; Fin de la rutina "INC_DEC_B".         ; Banco 0.
```

```
BTX                                     ; Rutina utilizada por las instrucciones que convierten un byte a cualquier
                                        ; otro tipo de dato.

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                          ; Se retorna a la instrucción que llamó a esta subrutina, de donde la
                                        ; instrucción NO será ejecutada y se trasladará al ciclo de ejecución de las
                                        ; demás instrucciones.

        call    Get_Arg_Val             ; Se lee el valor del byte a convertir en entero/Word/Real.

        call    Get_FLASH_Data                  ; Se lee el lsB de la dirección donde se colocará el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data          ; Se lee el msB de la dirección donde se colocará el resultado.
        movwf   Addr_Bus_Hi

        call    Set_Address             ; Se direcciona la posición del lsB del resultado.
        bsf     CS                      ; Se selecciona el IC RAM.

        return
                                        ; Fin de la rutina "BTX".       ; Banco 0.


DTX                                     ; Rutina utilizada por las instrucciones que convierten un Double en cualquier
                                        ; otro tipo de dato.

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                          ; Se retorna a la instrucción que llamó a esta subrutina, de donde la
                                        ; instrucción NO será ejecutada y se trasladará al ciclo de ejecución de las
                                        ; demás instrucciones.

        call    Get_Arg_Val             ; Se lee el valor del Double a convertir en entero/Real.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2
```

```
        call    Get_RAM_Data
        movwf   Byte3

        call    Get_FLASH_Data          ; Se lee el lsB de la dirección del entero/real donde se colocará el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data          ; Se lee el msB de la dirección del entero/real donde se colocará el resultado.
        movwf   Addr_Bus_Hi

        call    Set_Address             ; Se direcciona la posición del lsB del resultado.
        bsf     CS                      ; Se selecciona el IC RAM.

        goto    Instructions_Loop
                                        ; Fin de la rutina "DTX".       ; Banco 0.


Get_2_Bytes_To_Compare                  ; Rutina que lee todos los bytes de los dos argumentos (Bytes) que se desean
                                        ; comparar.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    Get_2_Bytes_No_Const

        call    Get_FLASH_Data          ; SI existe un parámetro constante.  Se leerá el byte de datos (constante)
        movwf   Byte1                   ; en la variable que posteriormente será utilizada para el procesamiento.
        goto    Get_2_Bytes_Arg2

Get_2_Bytes_No_Const:
        call    Get_Arg_Val             ; NO existe un parámetro constante.  Se lee el byte de datos.
        movf    ByteX,W
        movwf   Byte1

Get_2_Bytes_Arg2:
        call    Get_Arg_Val             ; Se lee el segundo parámetro de la instrucción.
        movf    ByteX,W
        movwf   Byte2

        return
                                        ; Fin de la rutina "Get_2_Bytes_To_Compare".    ; Banco 0.
```

```
Get_4_Bytes_To_Compare                  ; Rutina que lee todos los bytes de los dos argumentos (Words) que se desean
                                        ; comparar.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    Get_4_Bytes_No_Const

        call    Get_FLASH_Data          ; Si existe un parámetro constante.  Se leerá la palabra de datos (constante)
        movwf   Byte1                   ; en las variables que posteriormente serán utilizadas para el procesamiento.
        call    Get_FLASH_Data
        movwf   Byte2
        goto    Get_4_Bytes_Arg2

Get_4_Bytes_No_Const:
        call    Get_Arg_Val             ; NO existe un parámetro constante.  Se lee la palabra de datos (1er. byte).
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data            ; Se lee el segundo byte del primer parámetro (msB).
        movwf   Byte2

Get_4_Bytes_Arg2:
        call    Get_Arg_Val             ; Se lee el primer byte del segundo parámetro (lsB).
        movf    ByteX,W
        movwf   Byte3
        call    Get_RAM_Data            ; Se lee el segundo byte del segundo parámetro (msB).
        movwf   Byte4

        return
                                        ; Fin de la rutina "Get_4_Bytes_To_Compare".        ; Banco 0.


Get_8_Bytes_To_Compare                  ; Rutina que lee todos los bytes de los dos argumentos (double words)
                                        ; que se desean comparar.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    Get_8_Bytes_No_Const

        call    Get_FLASH_Data          ; Si existe un parámetro constante.  Se leerá la palabra de datos (constante)
```

```
        movwf   Byte1                   ; en las variables que posteriormente serán utilizadas para el procesamiento.
        call    Get_FLASH_Data
        movwf   Byte2
        call    Get_FLASH_Data
        movwf   Byte3
        call    Get_FLASH_Data
        movwf   Byte4
        goto    Get_8_Bytes_Arg2

Get_8_Bytes_No_Const:
        call    Get_Arg_Val             ; NO existe un parámetro constante.  Se lee la palabra de datos (1er. byte).
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data            ; Se lee el segundo byte del primer parámetro.
        movwf   Byte2
        call    Get_RAM_Data            ; Se lee el tercer byte del primer parámetro.
        movwf   Byte3
        call    Get_RAM_Data            ; Se lee el cuarto byte del primer parámetro (msB).
        movwf   Byte4

Get_8_Bytes_Arg2:
        call    Get_Arg_Val             ; Se lee el primer byte del segundo parámetro (lsB).
        movf    ByteX,W
        movwf   Byte5
        call    Get_RAM_Data            ; Se lee el segundo byte del segundo parámetro.
        movwf   Byte6
        call    Get_RAM_Data            ; Se lee el tercer byte del segundo parámetro.
        movwf   Byte7
        call    Get_RAM_Data            ; Se lee el cuarto byte del segundo parámetro (msB).
        movwf   Byte8

        return
                                        ; Fin de la rutina "Get_8_Bytes_To_Compare".        ; Banco 0.


Comparison_True

        bsf     STATUS,C                ; La comparación es verdadera.  Se cargará un uno (1) en la pila lógica.
        call    Push
```

```
        sublw   0x07
        btfsc   STATUS,Z        ; Si el bit a operar es D7, entonces ya no se harán desplazamientos.
        goto    Do_A
rlf_A_loop:
        rlf     ByteX           ; Se desplazan los bits hasta colocar el bit deseado en la posición D7.
        incf    ByteY
        movf    ByteY,W
        sublw   0x07
        btfss   STATUS,Z
        goto    rlf_A_loop
Do_A:
        movf    Logic_Stack,W
        andwf   ByteX           ; Se realiza la operación AND.
        call    Pop             ; Se extrae el tope de la pila lógica.
        rlf     ByteX           ; Se coloca el bit del resultado (D7) en el bit 'Carry' para cargarlo en la pila lógica.
        call    Push            ; Se carga el resultado en la pila lógica.

        goto    Instructions_Loop                               ; Banco 0.


O                               ; Instrucción "OR".

        call    Get_Arg_Val     ; Se lee el byte que posee el bit que se operará con el tope la pila lógica (OR).

O_ON_Code:
        call    Get_FLASH_Data
        movwf   ByteY           ; Se almacena el byte que indica el número de bit que se operará (OR).
        sublw   0x07
        btfsc   STATUS,Z        ; Si el bit a operar es D7, entonces ya no se harán desplazamientos.
        goto    Do_O
rlf_O_loop:
        rlf     ByteX           ; Se desplazan los bits hasta colocar el bit deseado en la posición D7.
        incf    ByteY
        movf    ByteY,W
        sublw   0x07
        btfss   STATUS,Z
        goto    rlf_O_loop
Do_O:
        movf    Logic_Stack,W
        iorwf   ByteX           ; Se realiza la operación OR.
        call    Pop             ; Se extrae el tope de la pila lógica.
```

```
        rlf     ByteX           ; Se coloca el bit del resultado (D7) en el bit 'Carry' para cargarlo en la pila lógica.
        call    Push            ; Se carga el resultado en la pila lógica.

        goto    Instructions_Loop                                   ; Banco 0.


LDN                             ; Instrucción "Load Not".

        call    Get_Arg_Val     ; Se lee el byte que posee el bit que se cargará en la pila lógica.

        comf    ByteX           ; Se complementa el byte que posee el bit a cargar en la pila lógica.

        goto    LD_LDN_Code     ; A partir de este punto, la instrucción es igual a "Load".     ; Banco 0.


AN                              ; Instrucción "AND NOT".

        call    Get_Arg_Val     ; Se lee el byte que posee el bit que se operará con el tope la pila lógica (AND NOT).

        comf    ByteX           ; Se complementa el byte que posee el bit a operar con el tope de la pila lógica.

        goto    A_AN_Code       ; A partir de este punto, la instrucción es igual a "AND".      ; Banco 0.


ON                              ; Instrucción "OR NOT".

        call    Get_Arg_Val     ; Se lee el byte que posee el bit que se operará con el tope la pila lógica (OR NOT).

        comf    ByteX           ; Se complementa el byte que posee el bit a operar con el tope de la pila lógica.

        goto    O_ON_Code       ; A partir de este punto, la instrucción es igual a "OR".; Banco 0.


NOT                             ; Instrucción "NOT".

        comf    Logic_Stack,W   ; Se almacena en ByteX el complemento de la pila lógica.
        movwf   ByteX
```

```
        call    Pop             ; Se extrae el tope de la pila lógica.
        rlf     ByteX           ; Se coloca el complemento del tope de la pila lógica en el bit 'Carry'.
        call    Push            ; Se carga el complemento del tope de la pila lógica.

        goto    Instructions_Loop                                   ; Banco 0.


EU                              ; Rutina "Edge UP".

        movf    Edge_Up_Counter,W
        movwf   Addr_Bus_Low    ; Se direcciona la localidad de memoria correspondiente al número de ocurrencia de esta
        movlw   0x7E            ; instrucción dentro del área de las instrucciones "Edge Up".
        movwf   Addr_Bus_Hi
        call    Set_Address

        incf    Edge_Up_Counter         ; Se incrementa el contador de instrucciones "Edge Up" para la próxima ocurrencia de ésta.

        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Read                 ; Se lee el byte que contiene el valor del tope de la pila en el ciclo anterior.
        btfss   STATUS,Z        ; Si el valor del tope de la pila en el ciclo anterior era uno (1), entonces no puede
        goto    No_Edge_Up      ; existir el flanco positivo, por lo que únicamente se actualiza la copia del tope, y
                                ; se pone en cero (no hubo flanco) el tope de la pila.

        btfss   Logic_Stack,7   ; Puesto que el valor del tope de la pila en el ciclo anterior era cero (0), entonces
        goto    No_Edge_Up      ; se debe verificar si ahora se presentó el flanco positivo.

        movlw   0x01            ; SI se presentó el flanco positivo.  Se coloca el uno (1) en la copia del tope de la
        call    IO_Write; pila, y se coloca un uno (1) en el tope de la pila (si hubo flanco positivo).

        bsf     Logic_Stack,7   ; Se coloca el uno (1) en el tope de la pila.

        goto    Instructions_Loop


No_Edge_Up:
        movlw   0x00            ; Se asume que el tope de la pila es cero (0).
        btfsc   Logic_Stack,7
        movlw   0x01
        call    IO_Write; Se actualiza la copia del tope de la pila (en el IC RAM actualmente seleccionado).
```

```
        bcf     Logic_Stack,7   ; Se coloca el cero (0) en el tope de la pila (no hubo flanco positivo).

        goto    Instructions_Loop                                       ; Banco 0.


ED                              ; Instrucción "Edge Down".

        movf    Edge_Down_Counter,W
        movwf   Addr_Bus_Low    ; Se direcciona la localidad de memoria correspondiente al número de ocurrencia de esta
        movlw   0x7F            ; instrucción dentro del área de las instrucciones "Edge Down".
        movwf   Addr_Bus_Hi
        call    Set_Address

        incf    Edge_Down_Counter;Se incrementa el contador de instrucciones "Edge Down" para la próxima ocurrencia de ésta.

        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Read                 ; Se lee el byte que contiene el valor del tope de la pila en el ciclo anterior.
        btfsc   STATUS,Z        ; Si el valor del tope de la pila en el ciclo anterior era cero (0), entonces no puede
        goto    No_Edge_Down    ; existir el flanco negativo, por lo que únicamente se actualiza la copia del tope, y
                                ; se pone en cero (no hubo flanco) el tope de la pila.

        btfsc   Logic_Stack,7   ; Puesto que el valor del tope de la pila en el ciclo anterior era uno (1), entonces
        goto    No_Edge_Up      ; se debe verificar si ahora se presentó el flanco negativo.

        movlw   0x00            ; SI se presentó el flanco negativo.  Se coloca el cero (0) en la copia del tope de la
        call    IO_Write; pila, y se coloca un uno (1) en el tope de la pila (si hubo flanco negativo).

        bsf     Logic_Stack,7   ; Se coloca el uno (1) en el tope de la pila.

        goto    Instructions_Loop


No_Edge_Down:
        movlw   0x00            ; Se asume que el tope de la pila es cero (0).
        btfsc   Logic_Stack,7
        movlw   0x01
        call    IO_Write; Se actualiza la copia del tope de la pila (en el IC RAM actualmente seleccionado).

        bcf     Logic_Stack,7   ; Se coloca el cero (0) en el tope de la pila (no hubo flanco negativo).
```

```
        goto    Instructions_Loop                       ; Barco 0.


ALD                             ; Instrucción "AND Load".

        call    Pop             ; Se extrae el tope de la pila (primer valor).
        rrf     ByteX           ; Se coloca el primer valor de la pila en D7 del registro ByteX.
        call    Pop             ; Se extrae el segundo valor de la pila.
        rrf     ByteY           ; Se coloca el segundo valor de la pila en D7 del registro ByteY.
        movf    ByteX,W
        andwf   ByteY           ; Se realiza la operación AND con los dos primeros bits de la pila.
        rlf     ByteY           ; ByteY <D7> tiene el resultado, por lo que se coloca en el bit 'Carry'.
        call    Push            ; Se carga el resultado en el tope de la pila (la profundidad de ésta disminuyó en uno).

        goto    Instructions_Loop                       ; Barco 0.


OLD                             ; Instrucción "OR Load".

        call    Pop             ; Se extrae el tope de la pila (primer valor).
        rrf     ByteX           ; Se coloca el primer valor de la pila en D7 del registro ByteX.
        call    Pop             ; Se extrae el segundo valor de la pila.
        rrf     ByteY           ; Se coloca el segundo valor de la pila en D7 del registro ByteY.
        movf    ByteX,W
        iorwf   ByteY           ; Se realiza la operación OR con los dos primeros bits de la pila.
        rlf     ByteY           ; ByteY <D7> tiene el resultado, por lo que se coloca en el bit 'Carry'.
        call    Push            ; Se carga el resultado en el tope de la pila (la profundidad de ésta disminuyó en uno).

        goto    Instructions_Loop                       ; Barco 0.


LPS                             ; Instrucción "Logic Push".

        bcf     STATUS,C
        btfsc   Logic_Stack,7
        bsf     STATUS,C
        call    Push            ; Se duplica el valor del tope de la pila (se pierde el último valor de ésta).
```

```
        goto    Instructions_Loop                       ; Banco 0.


LRD                             ; Instrucción "Logic Read".

        bcf     Logic_Stack,7   ; Se asume que el segundo valor de la pila es un cero (0).
        btfsc   Logic_Stack,6   ; Se copia el segundo valor de la pila en el tope de ésta (no se hace "Push" ni "Pop").
        bsf     Logic_Stack,7   ; Si el segundo valor de la pila no es un cero (0), entonces se copia el uno (1).

        goto    Instructions_Loop                       ; Banco 0.


LPP                             ; Instrucción "Logic Pop".

        goto    Pop                                     ; Banco 0.


OUTPUT                                  ; Instrucción "Output" ( = ).

        call    Get_Arg_Val     ; Se lee el byte que posee el bit al que se cargará el tope de la pila lógica.
        call    Get_FLASH_Data
        movwf   ByteY           ; Se almacena el byte que indica el número de bit al que se cargará el tope de la pila.
        incf    ByteY

        btfss   Logic_Stack,7   ; Se determina si el tope de la pila es un uno (1) o un cero (0).
        goto    Out_Zero

Out_One:                        ; Se debe colocar un uno (1) en el bit de salida.
        bsf     STATUS,C        ; Se colocará un uno (1) en la posición del bit de salida, en "ByteZ", y luego se
        clrf    ByteZ           ; realizará una operación OR con el byte que posee el bit de salida.
rlf_Out_One_Bit_Loop:
        rlf     ByteZ           ; Se desplaza el uno (1) hasta colocarlo en la posición del bit de salida.
        decfsz  ByteY
        goto    rlf_Out_One_Bit_Loop
        movf    ByteZ,W
        iorwf   ByteX           ; Se utiliza la operación OR para colocar el uno (1) en la posición del bit de salida.

        call    Set_Address     ; Los registros de direccionamiento en el bus aún conservan la dirección del byte que
```

```
                                ; posee el bit de salida, por lo que se utilizarán para escribir a éste.
        bsf     CS              ; Se selecciona el IC RAM.
        movf    ByteX,W
        call    IO_Write; Se escribe el resultado almacenado en W (ByteX).

        goto    Instructions_Loop


Out_Zero:                       ; Se debe colocar un cero (0) en el bit de salida.
        bcf     STATUS,C        ; Se colocará un cero (0) en la posición del bit de salida, en "ByteZ", y luego se
        movlw   0xFF            ; realizará una operación AND con el byte que posee el bit de salida.
        movwf   ByteZ
rlf_Out_Zero_Bit_Loop:
        rlf     ByteZ           ; Se desplaza el cero (0) hasta colocarlo en la posición del bit de salida.
        decfsz  ByteY
        goto    rlf_Out_Zero_Bit_Loop
        movf    ByteZ,W
        andwf   ByteX           ; Se utiliza la operación AND para colocar el cero (0) en la posición del bit de salida.

        call    Set_Address     ; Los registros de direccionamiento en el bus aún conservan la dirección del byte que
                                ; posee el bit de salida, por lo que se utilizarán para escribir a éste.
        bsf     CS              ; Se selecciona el IC RAM.
        movf    ByteX,W
        call    IO_Write; Se escribe el resultado almacenado en W (ByteX).

        goto    Instructions_Loop                                   ; Banco 0.


SET_Bits                        ; Instrucción "SET".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        call    Get_FLASH_Data          ; Se lee la cantidad de bits que se deben poner en uno (1).
        movwf   S_R_Bit_Counter ; Se almacena la cantidad de bits que se deben poner en uno (1).

        call    Get_Arg_Val     ; Se lee el byte que posee el bit de inicio a partir del cual se escribirán unos (1).
        call    Get_FLASH_Data
        movwf   ByteY           ; Se almacena el byte que indica el número del bit de inicio.
```

```
        incf    ByteY
        movwf   ByteY2          ; Copia de ByteY (ya incrementado).

Set_Bit_i:                      ; Se debe colocar un uno (1) en el bit de salida.
        call    Set_Address     ; Los registros de direccionamiento en el bus aún conservan la dirección del byte a acceder.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Read
        movwf   ByteX           ; Se almacena el byte al cual se le colocarán los bits en uno (1).
                                ; posee el bit de salida, por lo que se utilizarán para escribir a éste.
        bsf     STATUS,C        ; Se colocará un uno (1) en la posición del bit de salida, en "ByteZ", y luego se
        clrf    ByteZ           ; realizará una operación OR con el byte que posee el bit de salida.
rlf_Set_Bit_Loop:
        rlf     ByteZ           ; Se desplaza el uno (1) hasta colocarlo en la posición del bit de salida.
        decfsz  ByteY
        goto    rlf_Set_Bit_Loop
        movf    ByteZ,W
        iorwf   ByteX           ; Se utiliza la operación OR para colocar el uno (1) en la posición del bit de salida.

        bsf     CS              ; Se selecciona el IC RAM.
        movf    ByteX,W
        call    IO_Write; Se escribe el resultado almacenado en W (ByteX).

        decf    S_R_Bit_Counter ; Se decrementa el contador de bits a poner en uno (SET) y se verifica si ya se colocaron
        btfsc   STATUS,Z        ; todos a uno.
        goto    Instructions_Loop;Ya se colocaron todos los bits a uno (1), por lo que se finaliza la instrucción.

        incf    ByteY2          ; Se incrementa el número de bit a colocar en uno (1).
        movf    ByteY2,W
        movwf   ByteY
        movlw   0x09            ; Si el número de bit ya alcanzó el nueve (9), entonces se debe incremetar la dirección
        subwf   ByteY,W         ; del byte, y reinicializar con uno (1) el número de bit.
        btfss   STATUS,Z
        goto    No_New_S_Byte
New_S_Byte:
        incf    Addr_Bus_Low    ; Se lee el siguiente byte que posee los bits que se colocarán en uno (1).
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi

        movlw   0x01            ; Se reinicializa con uno (1) el número de bit.
        movwf   ByteY
```

```
        movwf   ByteY2

No_New_S_Byte:
        goto    Set_Bit_i
                                                        ; Banco 0.


RESET_Bits                      ; Instrucción "RESET".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_FLASH_Data          ; Se lee la cantidad de bits que se deben poner en cero (0).
        movwf   S_R_Bit_Counter ; Se almacena la cantidad de bits que se deben poner en cero (0).

        call    Get_Arg_Val     ; Se lee el byte que posee el bit de inicio a partir del cual se escribirán ceros (0).
        call    Get_FLASH_Data
        movwf   ByteY           ; Se almacena el byte que indica el número del bit de inicio.
        incf    ByteY
        movwf   ByteY2          ; Copia de ByteY (ya incrementado).


Reset_Bit_i:                    ; Se debe colocar un cero (0) en el bit de salida.
        call    Set_Address     ; Los registros de direccionamiento en el bus aún conservan la dirección del byte que
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Read
        movwf   ByteX           ; Se almacena el byte al cual se le colocarán los bits en cero (0).
                                ; posee el bit de salida, por lo que se utilizarán para escribir a éste.
        bcf     STATUS,C        ; Se colocará un cero (0) en la posición del bit de salida, en "ByteZ", y luego se
        movlw   0xFF            ; realizará una operación AND con el byte que posee el bit de salida.
        movwf   ByteZ
rlf_Reset_Bit_Loop:
        rlf     ByteZ           ; Se desplaza el cero (0) hasta colocarlo en la posición del bit de salida.
        decfsz  ByteY
        goto    rlf_Reset_Bit_Loop
        movf    ByteZ,W
        andwf   ByteX           ; Se utiliza la operación AND para colocar el cero (0) en la posición del bit de salida.

        bsf     CS              ; Se selecciona el IC RAM.
```

```
        movf    ByteX,W
        call    IO_Write; Se escribe el resultado almacenado en W (ByteX).

        decf    S_R_Bit_Counter ; Se decrementa el contador de bits a poner en cero (RESET) y se verifica si ya se colocaron
        btfsc   STATUS,Z        ; todos a cero.
        goto    Instructions_Loop;Ya se colocaron todos los bits a cero (0), por lo que se finaliza la instrucción.

        incf    ByteY2          ; Se incrementa el número de bit a colocar en cero (0).
        movf    ByteY2,W
        movwf   ByteY
        movlw   0x09            ; Si el número de bit ya alcanzó el nueve (9), entonces se debe incremetar la dirección
        subwf   ByteY,W         ; del byte, y reinicializar con uno (1) el número de bit.
        btfss   STATUS,Z
        goto    No_New_R_Byte
New_R_Byte:
        incf    Addr_Bus_Low    ; Se lee el siguiente byte que posee los bits que se colocarán en cero (0).
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi

        movlw   0x01            ; Se reinicializa con uno (1) el número de bit.
        movwf   ByteY
        movwf   ByteY2

No_New_R_Byte:
        goto    Reset_Bit_i
                                                                ; Banco 0.

NOTH                            ; Instrucción "Nothing".

        goto    Instructions_Loop
                                                                ; Banco 0.

LD_B_Equal                              ; Instrucción "Load Byte Equal".

        call    Get_2_Bytes_To_Compare  ; Se leen los dos bytes que se desean comparar.

        movf    Byte1,W
        subwf   Byte2                   ; Se determina si los dos bytes son iguales.
        btfss   STATUS,Z
```

```
        goto    Comparison_False
        goto    Comparison_True
                                                        ; Banco 0.

LD_W_Equal                      ; Instrucción "Load Word Equal".

        call    Get_4_Bytes_To_Compare      ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (Words).

        movf    Byte1,W
        subwf   Byte3
        btfss   STATUS,Z
        goto    Comparison_False
        movf    Byte2,W
        subwf   Byte4
        btfss   STATUS,Z
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.

LD_D_Equal                      ; Instrucción "Load Double Equal".

        call    Get_8_Bytes_To_Compare      ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (DWords).

        movf    Byte1,W
        subwf   Byte5
        btfss   STATUS,Z
        goto    Comparison_False
        movf    Byte2,W
        subwf   Byte6
        btfss   STATUS,Z
        goto    Comparison_False
        movf    Byte3,W
        subwf   Byte7
        btfss   STATUS,Z
        goto    Comparison_False
        movf    Byte4,W
```

```
        subwf   Byte8
        btfss   STATUS,Z
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.

LD_R_Equal                      ; Instrucción "Load Real Equal".

        call    Get_8_Bytes_To_Compare

        call    Process_Reals_To_Compare

        movlw   0x20
        subwf   Receive_Pak     ; Se determina si se recibió el caracter ASCII 0x20 (espacio en blanco).
        btfss   STATUS,Z
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.

LD_B_Greater                    ; Instrucción "Load Byte Greater than".

        call    Get_2_Bytes_To_Compare  ; Se leen los dos bytes que se desean comparar.

        movf    Byte1,W
        subwf   Byte2           ; Se determina si 'Byte1' es mayor que 'Byte2'.
        btfsc   STATUS,C
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.

LD_W_Greater                    ; Instrucción "Load Word Greater than".
```

```
        call    Get_4_Bytes_To_Compare      ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (Words).

        movf    Byte4,W
        subwf   Byte2                       ; Se determina si 'Word1' es mayor que 'Word2'.
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        movf    Byte3,W
        subwf   Byte1
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        goto    Comparison_False
                                                                    ; Banco 0.


LD_D_Greater                                ; Instrucción "Load Double Word Greater than".

        call    Get_8_Bytes_To_Compare      ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (DWords).

        movf    Byte8,W
        subwf   Byte4                       ; Se determina si 'Double Word1' es mayor que 'Double Word2'.
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        movf    Byte7,W
        subwf   Byte3
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        movf    Byte6,W
        subwf   Byte2
```

```
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        movf    Byte5,W
        subwf   Byte1
        btfss   STATUS,C
        goto    Comparison_False
        btfss   STATUS,Z
        goto    Comparison_True

        goto    Comparison_False
                                                        ; Banco 0.


LD_R_Greater                    ; Instrucción "Load Real Greater than".

        call    Get_8_Bytes_To_Compare

        call    Process_Reals_To_Compare

        movlw   0x2D
        subwf   Receive_Pak     ; Se determina si se recibió el caracter ASCII 0x2D (signo negativo).
        btfss   STATUS,Z        ; se espera el signo negativo puesto que se intercambiaron X y Y (SWAP).
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.


LD_B_Less                       ; Instrucción "Load Byte Less than".

        call    Get_2_Bytes_To_Compare      ; Se leen los dos bytes que se desean comparar.

        movf    Byte2,W
        subwf   Byte1           ; Se determina si 'Byte1' es menor que 'Byte2'.
        btfsc   STATUS,C
        goto    Comparison_False
```

```
        goto    Comparison_True
                                                        ; Banco 0.

LD_W_Less                       ; Instrucción "Load Word Less than".

        call    Get_4_Bytes_To_Compare  ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (Words).

        movf    Byte4,W
        subwf   Byte2           ; Se determina si 'Word1' es menor que 'Word2'.
        btfss   STATUS,C
        goto    Comparison_True
        btfss   STATUS,Z
        goto    Comparison_False

        movf    Byte3,W
        subwf   Byte1
        btfss   STATUS,C
        goto    Comparison_True
        goto    Comparison_False
                                                        ; Banco 0.

LD_D_Less                       ; Instrucción "Load Double Word Less than".

        call    Get_8_Bytes_To_Compare  ; Se leen TODOS los bytes de los dos argumentos que se desean comparar (Words).

        movf    Byte8,W
        subwf   Byte4           ; Se determina si 'Double Word1' es menor que 'Double Word2'.
        btfss   STATUS,C
        goto    Comparison_True
        btfss   STATUS,Z
        goto    Comparison_False

        movf    Byte7,W
        subwf   Byte3
        btfss   STATUS,C
        goto    Comparison_True
```

```
        btfss   STATUS,Z
        goto    Comparison_False

        movf    Byte6,W
        subwf   Byte2
        btfss   STATUS,C
        goto    Comparison_True
        btfss   STATUS,Z
        goto    Comparison_False

        movf    Byte5,W
        subwf   Byte1
        btfss   STATUS,C
        goto    Comparison_True
        goto    Comparison_False
                                                        ; Banco 0.

LD_R_Less                       ; Instrucción "Load Real Less than".

        call    Get_8_Bytes_To_Compare

        call    Process_Reals_To_Compare

        movlw   0x2B
        subwf   Receive_Pak     ; Se determina si se recibió el caracter ASCII 0x2B (signo positivo).
        btfss   STATUS,Z        ; se espera el signo positivo puesto que se intercambiaron X y Y (SWAP).
        goto    Comparison_False

        goto    Comparison_True
                                                        ; Banco 0.

BTI                             ; Instrucción "Byte to Integer".

        call    BTX             ; Se adquiere el byte a convertir y la dirección del dato de salida.

        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.
```

```
        movf    ByteX,W
        call    IO_Write        ; Se escribe el lsB del entero.

        movlw   0x00
        call    Put_RAM_Data            ; Se escribe el msB del entero.

        goto    Instructions_Loop
                                                        ; Banco 0.


BTD                             ; Instrucción "Byte to Double".

        call    BTX             ; Se adquiere el byte a convertir y la dirección del dato de salida.

        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        movf    ByteX,W
        call    IO_Write        ; Se escribe el lsB del Double.
        movlw   0x00
        call    Put_RAM_Data
        movlw   0x00
        call    Put_RAM_Data
        movlw   0x00
        call    Put_RAM_Data            ; Se escribe el msB del Double.

        goto    Instructions_Loop
                                                        ; Banco 0.


BTR                             ; Instrucción "Byte to Real".

        call    BTX             ; Se adquiere el byte a convertir y la dirección del dato de salida.

        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        bcf     PCLATH,3                ; Se selecciona la página 0.
```

```
movlw   0x01            ; Comando LOADX.
call    Pak_Tx

movlw   0x00            ; msB del entero de 24 bits equivalente al byte que se desea convertir a real.
call    Pak_Tx

movlw   0x00
call    Pak_Tx

movlw   0x00
call    Pak_Tx

movf    ByteX,W         ; lsB del entero de 24 bits equivalente al byte que se desea convertir a real.
call    Pak_Tx

movlw   0x07            ; Comando FLOAT (convierte X de entero de 24 bits a Pak).
call    Pak_Tx

call    Pak_Rx          ; Lectura del byte STATUS de retorno del comando FLOAT.
movf    Receive_Pak,W
btfss   STATUS,Z        ; Se determina si sucedió algún error en la operación matemática.
goto    Math_Error      ; Si ocurrió un error matemático.

movlw   0x06            ; Comando IEEECVT (Convierte de Pak a IEEE745).
call    Pak_Tx

movlw   0x03            ; Comando READX.
call    Pak_Tx

call    Pak_Rx          ; Se recibe el número real IEEE745.
movf    Receive_Pak,W
movwf   Byte4           ; msB del real IEEE745.
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte3
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte2
call    Pak_Rx
movf    Receive_Pak,W
```

```
        movwf   Byte1                   ; lsB del real IEEE745.

        bsf     PCLATH,3                ; Se selecciona la página 1.

        movf    Byte1,W                 ; lsB del # real.
        call    IO_Write        ; Se escribe el # real en el IC RAM.
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data            ; msB del # real.

        goto    Instructions_Loop
                                                                ; Banco 0.


ITD                                     ; Instrucción "Integer to Double".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.


        call    Get_Arg_Val             ; Se lee el valor del entero a convertir en Double.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2

        call    Get_FLASH_Data                  ; Se lee el lsB de la dirección del Double donde se colocará el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data                  ; Se lee el msB de la dirección del Double donde se colocará el resultado.
        movwf   Addr_Bus_Hi

        call    Set_Address             ; Se direcciona la posición del lsB del resultado.
        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte1,W
        call    IO_Write        ; Se escribe el lsB del Double.
```

```
        movf    Byte2,W
        andlw   0x7F            ; Se enmascara el msb, que contiene el signo del # entero.
        call    Put_RAM_Data

        movlw   0x00
        btfsc   Byte2,7         ; Se copia el signo del # entero en el msb del # Double.
        movlw   0x80
        call    Put_RAM_Data

        movlw   0x00
        call    Put_RAM_Data    ; Se escribe el msB del Double.

        goto    Instructions_Loop
                                                        ; Banco 0.

ITR                             ; Instrucción "Integer to Real".

        call    Test_Flow_PC4   ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val     ; Se lee el valor del entero a convertir en Real.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2

        call    Get_FLASH_Data          ; Se lee el lsB de la dirección del real donde se colocará el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data          ; Se lee el msB de la dirección del real donde se colocará el resultado.
        movwf   Addr_Bus_Hi

        call    Set_Address     ; Se direcciona la posición del lsB del resultado.
        bsf     CS              ; Se selecciona el IC RAM.

        bcf     PCLATH,3        ; Se selecciona la página 0.

        movlw   0x01            ; Comando LOADX.
```

```
call    Pak_Tx

movlw   0x00            ; msB del entero de 24 bits equivalente al entero que se desea convertir a real.
call    Pak_Tx

movlw   0x00
call    Pak_Tx

movf    Byte2,W
andlw   0x7F            ; Se enmascara el msb, que contiene el signo del # entero.
call    Pak_Tx

movf    Byte1,W         ; lsB del entero de 24 bits equivalente al entero que se desea convertir a real.
call    Pak_Tx

movlw   0x07            ; Comando FLOAT (convierte X de entero de 24 bits a Pak).
call    Pak_Tx

call    Pak_Rx          ; Lectura del byte STATUS de retorno del comando FLOAT.
movf    Receive_Pak,W
btfss   STATUS,Z        ; Se determina si sucedió algún error en la operación matemática.
goto    Math_Error      ; Sí ocurrió un error matemático.

movlw   0x0A
btfss   Byte2,7         ; Se determina si el # real debe tener signo negativo.
call    Pak_Tx          ; Se coloca el signo negativo en el # real.

movlw   0x06            ; Comando IEEECVT (Convierte de Pak a IEEE745).
call    Pak_Tx

movlw   0x03            ; Comando READX.
call    Pak_Tx

call    Pak_Rx          ; Se recibe el número real IEEE745.
movf    Receive_Pak,W
movwf   Byte4           ; msB del real IEEE745.
call    Pak_Rx
movf    Receive_Pak,W
movwf   Byte3
call    Pak_Rx
movf    Receive_Pak,W
```

```
        movwf   Byte2
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte1                   ; lsB del real IEEE745.

        bsf     PCLATH,3                ; Se selecciona la página 1.

        movf    Byte1,W                 ; lsB del # real.
        call    IO_Write        ; Se escribe el # real en el IC RAM.
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data            ; msB del # real.

        goto    Instructions_Loop
                                                        ; Banco 0.


DTI                                     ; Instrucción "Double to Integer".

        call    DTX                     ; Se adquieren los datos del Double a convertir y la dirección de salida.

        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        movf    Byte1,W
        call    IO_Write        ; Se escribe el lsB del entero.

        bcf     Byte2,7                 ; Se asume que el signo del # Double es negativo (Se copia en el msb del # entero).
        btfsc   Byte3,7                 ; Se determina si el Double es negativo o positivo.
        bsf     Byte2,7                 ; El signo del # Double es positivo, por lo que se copia en el msb del # entero.
        movf    Byte2,W
        call    Put_RAM_Data            ; Se escribe el msB del entero.

        goto    Instructions_Loop
                                                        ; Banco 0.
```

```
DTR                                     ; Instrucción "Double to Real".

        call    DTX                     ; Se adquieren los datos del Double a convertir y la dirección de salida.

        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        movlw   0x01                    ; Comando LOADX.
        call    Pak_Tx

        movlw   0x00                    ; msB del entero de 24 bits equivalente al Double que se desea convertir a real.
        call    Pak_Tx

        movf    Byte3,W
        andlw   0x7F                    ; Se enmascara el msb, que contiene el signo del # Double.
        call    Pak_Tx

        movf    Byte2,W
        call    Pak_Tx

        movf    Byte1,W                 ; lsB del entero de 24 bits equivalente al Double que se desea convertir a real.
        call    Pak_Tx

        movlw   0x07                    ; Comando FLOAT (convierte X de entero de 24 bits a Pak).
        call    Pak_Tx

        call    Pak_Rx                  ; Lectura del byte STATUS de retorno del comando FLOAT.
        movf    Receive_Pak,W
        btfss   STATUS,Z                ; Se determina si sucedió algún error en la operación matemática.
        goto    Math_Error              ; Si ocurrió un error matemático.

        movlw   0x0A
        btfss   Byte3,7                 ; Se determina si el # real debe tener signo negativo.
        call    Pak_Tx                  ; Se coloca el signo negativo en el # real.

        movlw   0x06                    ; Comando IEEECVT (Convierte de Pak a IEEE745).
        call    Pak_Tx
```

```
        movlw   0x03                    ; Comando READX.
        call    Pak_Tx

        call    Pak_Rx                  ; Se recibe el número real IEEE745.
        movf    Receive_Pak,W
        movwf   Byte4                   ; msB del real IEEE745.
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte3
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte2
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte1                   ; lsB del real IEEE745.

        bsf     PCLATH,3                ; Se selecciona la página 1.

        movf    Byte1,W                 ; lsB del # real.
        call    IO_Write        ; Se escribe el # real en el IC RAM.
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data            ; msB del # real.

        goto    Instructions_Loop
                                                        ; Banco 0.

RTD                                     ; Instrucción "Real to Double".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.


        movlw   0x00                    ; Se asigna el código para la conversión 'Real to Double'.
        movwf   RTX_Code
```

```
        call    RTX

        goto    Instructions_Loop
                                                        ; Banco 0.

RTI                                     ; Instrucción "Real to Integer".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        movlw   0x01                    ; Se asigna el código para la conversión 'Real to Integer'.
        movwf   RTX_Code

        call    RTX

        goto    Instructions_Loop
                                                        ; Banco 0.

RTB                                     ; Instrucción "Real to Byte".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        movlw   0x02                    ; Se asigna el código para la conversión 'Real to Byte'.
        movwf   RTX_Code

        call    RTX

        goto    Instructions_Loop
                                                        ; Banco 0.
```

```
CTU                                     ; Instrucción "Counter Up".

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    CTU_No_Const

        call    Get_FLASH_Data                  ; Se lee la constante (Word) para el valor predeterminado del Contador.
        movwf   Byte2           ; lsB del VP.
        call    Get_FLASH_Data
        movwf   Byte3           ; msB del VP.

        goto    Check_If_Reset

CTU_No_Const:

        call    Get_Arg_Val     ; Se lee la variable que contiene el valor predeterminado.
        movf    ByteX,W
        movwf   Byte2           ; lsB del VP.
        call    Get_RAM_Data
        movwf   Byte3           ; msB del VP.

Check_If_Reset:
        call    Get_FLASH_Data          ; Se lee el # de Contador.
        movwf   Byte1

        bcf     STATUS,IRP      ; Se utilizará direccionamiento indirecto en el banco 1.
        movlw   0xA0            ; El puntero será inicializado con la dirección de inicio del área de memoria
        addwf   Byte1,W         ; de los Contadores, más un desplazamiento según el # de Contador a acceder.
        addwf   Byte1,W         ; (cada Contador utiliza dos bytes, por ello se suma dos veces).
        movwf   FSR             ; El puntero está posicionado en el lsB del Contador indicado por la instrucción.

        btfss   Logic_Stack,7   ; Se determina si se debe borrar el Contador.
        goto    Check_If_Count
        clrf    INDF            ; Sí se debe borrar el Contador (y el correspondiente bit C).
        incf    FSR             ; Se borra el Contador (Word).
        clrf    INDF

        goto    Instructions_Loop

Check_If_Count:
        movf    Logic_Stack,W   ; Se crea una copia de la pila lógica para restablecer su contenido luego de
```

```
        movwf   Logic_Stack_Copy        ; ejecutar la rutina "Edge Up" para determinar si hubo flanco en la entrada Up
                                        ; del Contador.

        call    Pop                     ; Se determina si se debe incrementar el Contador.  Se extrae el tope de la pila
                                        ; para efectuar una instrucción "Edge Up" en el bit 'Logic_Stack <6>'.

                                        ; Rutina "Edge Up" con el agregado del incremento del Contador.
        movf    Edge_Up_Counter,W
        movwf   Addr_Bus_Low            ; Se direcciona la localidad de memoria correspondiente al número de ocurrencia de
        movlw   0x7E                    ; esta instrucción dentro del área de las instrucciones "Edge Up".
        movwf   Addr_Bus_Hi
        call    Set_Address

        incf    Edge_Up_Counter             ; Se incrementa el contador de instrucciones "Edge Up" para la próxima ocurrencia
                                        ; de ésta.
        bsf     CS                      ; Se selecciona el IC RAM.
        call    IO_Read                     ; Se lee el byte que contiene el valor del tope de la pila en el ciclo anterior.
        btfss   STATUS,Z                ; Si el valor del tope de la pila en el ciclo anterior era uno (1), entonces no
        goto    No_Counter_Edge_Up      ; puede existir el flanco positivo, por lo que únicamente se actualiza la copia
                                        ;  del tope, y se pone en cero (no hubo flanco) el tope de la pila.

        btfss   Logic_Stack,7           ; Puesto que el valor del tope de la pila en el ciclo anterior era cero (0),
        goto    No_Counter_Edge_Up      ; entonces se debe verificar si ahora se presentó el flanco positivo.

        movlw   0x01                    ; SI se presentó el flanco positivo.  Se coloca el uno (1) en la copia del tope de
        call    IO_Write        ; la pila y se continúa con el incremento del valor del Contador.

        movlw   0xFF                    ; Se determina si el Contador ya llegó al valor máximo.
        subwf   INDF,W
        btfsc   STATUS,Z
        goto    Check_msB_Counter
        incf    FSR                     ; Aún no ha llegado al valor máximo, por lo tanto puede realizar el incremento.
        goto    Do_Count                ; Se incrementó el puntero solo debido a que en 'Do_Count' se asume que éste se
                                        ; encuentra direccionando al msB del Contador.
Check_msB_Counter:
        incf    FSR
        movf    INDF,W
        andlw   0x7F
        sublw   0x7F
        btfss   STATUS,Z
        goto    Do_Count
```

```
        goto    CTU_End                 ; Ya llegó a su valor máximo, por lo que NO se realizará el incremento.

Do_Count:
        movlw   0x00                    ; Se incrementa el valor del Contador.
        decf    FSR                     ; El puntero se posiciona en el lsB del Contador.
        incf    INDF
        btfsc   STATUS,Z
        movlw   0x01
        incf    FSR                     ; El puntero se posiciona en el msB del Contador.
        addwf   INDF

        movf    Byte3,W                 ; Se determina si el Contador ya es mayor o igual que el valor predeterminado (VP).
        andlw   0x7F                    ; Se enmascara el bit del signo del V.P.
        subwf   INDF,W
        btfss   STATUS,C
        goto    Reset_Counter_Bit       ; Se pone en cero (0) el bit C.
        btfss   STATUS,Z
        goto    Set_Counter_Bit         ; Se pone en uno (1) el bit C.

        decf    FSR                     ; El puntero se posiciona en el lsB del Contador.
        movf    Byte2,W
        subwf   INDF,W
        incf    FSR                     ; ('incf' no afecta al bit 'Carry' del registro 'STATUS').
        btfss   STATUS,C
        goto    Reset_Counter_Bit       ; Se pone en cero (0) el bit C.
        goto    Set_Counter_Bit         ; Se pone en uno (1) el bit C.


Set_Counter_Bit:
        bsf     INDF,7
        goto    CTU_End

Reset_Counter_Bit:
        bcf     INDF,7
        goto    CTU_End


No_Counter_Edge_Up:
        movlw   0x00                    ; Se asume que el tope de la pila es cero (0).
        btfsc   Logic_Stack,7
        movlw   0x01
```

```
        call    IO_Write        ; Se actualiza la copia del tope de la pila (en el IC RAM actualmente seleccionado).

CTU_End:
        movf    Logic_Stack_Copy,W      ; Se restablece el contenido original de la pila lógica antes de ejecutar la
        movwf   Logic_Stack             ; instrucción "Counter Up".

        goto    Instructions_Loop
                                        ; Fin de la rutina "Edge Up" con el agregado del incremento del Contador.

                                                                        ; Banco 0.


TONR                                    ; Instrucción "Timer On Retentive".

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    TONR_No_Const


        call    Get_FLASH_Data                  ; Se lee la constante (Word) para el valor predeterminado del Temporizador.
        movwf   Byte2                   ; lsB del VP.
        call    Get_FLASH_Data
        movwf   Byte3                   ; msB del VP.

        goto    Get_Timer_Number

TONR_No_Const:

        call    Get_Arg_Val             ; Se lee la variable que contiene el valor predeterminado.
        movf    ByteX,W
        movwf   Byte2                   ; lsB del VP.
        call    Get_RAM_Data
        movwf   Byte3                   ; msB del VP.

Get_Timer_Number:
        call    Get_FLASH_Data                  ; Se lee el # de Temporizador.
        movwf   Byte1

        movlw   .48                     ; Se determinará si el Temporizador a acceder se encuentra en el banco 2 ó 3.
        subwf   Byte1,W
        btfsc   STATUS,C
```

```
        goto    Timer_In_Bank3      ; NO se direccionará el banco 2.  El Temporizador a acceder está en el banco 3.
        movlw   0x10                ; El Temporizador al que se desea acceder está en el banco 2.
        movwf   FSR                 ; El puntero es inicializado con la dirección de inicio del área de memoria T.
        goto    Set_Pointer

Timer_In_Bank3:
        movlw   0x90                ; NO se direccionará el banco 2.  El Temporizador a acceder está en el banco 3.
        movwf   FSR
        movlw   .48
        subwf   Byte1               ; 'Byte1' es alterado y ahora posee el valor del desplazamiento que se realizará
                                    ; en el puntero.

Set_Pointer:
        bsf     STATUS,IRP          ; Se utilizará direccionamiento indirecto en los bancos 2 y 3.

        movf    FSR,W
        addwf   Byte1,W             ; Se realiza un desplazamiento en el puntero según el # de Temporizador a acceder.
        addwf   Byte1,W             ; (cada Temporizador utiliza dos bytes, por ello se suma dos veces).
        movwf   FSR                 ; El puntero está posicionado en el lsB del Temporizador indicado por la
                                    ; instrucción.

Check_If_Timer_Enabled:
        incf    FSR                 ; El puntero se posiciona en el msB del Temporizador.
        btfss   Logic_Stack,7       ; Se determina si se debe habilitar/deshabilitar el Temporizador.
        bcf     INDF,6              ; Se DESHABILITA el Temporizador.
        btfsc   Logic_Stack,7
        bsf     INDF,6              ; Se HABILITA el Temporizador.
                                    ; Se realizó esta doble comparación para proteger al Sistema en contra de un
                                    ; retraso en el Temporizador accedido, puesto que si se habilita/deshabilita
                                    ; el Temporizador de forma asumida, y luego se habilita/deshabilita como resultado
                                    ; de la revisión del tope de la pila lógica, se puede provocar un retraso en el
                                    ; Temporizador si cuando éste se encuentra habilitado/deshabilitado sucede la
                                    ; Interrupción del Sistema de actualización de los temporizadores.
        movlw 0x7F
        andwf   Byte3               ; Se enmascara el bit del signo del V.P.

        movf    INDF,W              ; Se determina si el Temporizador ya es mayor o igual que el valor predeterminado.
        andlw   0x3F                ; Se enmascaran los dos bits más significativos del Temporizador (bit T y bit TEN).
        subwf   Byte3,W
```

```
        btfss   STATUS,C
        goto    Set_Timer_Bit           ; Se pone en uno (1) el bit T.
        btfss   STATUS,Z
        goto    Reset_Timer_Bit         ; Se pone en cero (0) el bit T.

        decf    FSR                     ; El puntero se posiciona en el lsB del Temporizador.
        movf    Byte2,W
        subwf   INDF,W
        incf    FSR                     ; msB del Temporizador ('incf' no afecta al bit 'Carry' del registro 'STATUS').
        btfss   STATUS,C
        goto    Reset_Timer_Bit         ; Se pone en cero (0) el bit T.
        goto    Set_Timer_Bit           ; Se pone en uno (1) el bit T.

Set_Timer_Bit:
        bsf     INDF,7
        goto    Instructions_Loop

Reset_Timer_Bit:
        bcf     INDF,7
        goto    Instructions_Loop
                                                        ; Banco 0.


TRESET                                  ; Instrucción "Timer Reset".

        call    Get_FLASH_Data                  ; Se lee el # de Temporizador.
        movwf   Byte1

        movlw   .48                     ; Se determinará si el Temporizador a acceder se encuentra en el banco 2 ó 3.
        subwf   Byte1,W
        btfsc   STATUS,C
        goto    Reset_Timer_In_Bank3    ; NO se direccionará el banco 2.  El Temporizador a acceder está en el banco 3.
        movlw   0x10                    ; El Temporizador al que se desea acceder está en el banco 2.
        movwf   FSR                     ; El puntero es inicializado con la dirección de inicio del área de memoria T.
        goto    Set_R_Pointer

Reset_Timer_In_Bank3:
        movlw   0x90                    ; NO se direccionará el banco 2.  El Temporizador a acceder está en el banco 3.
        movwf   FSR
        movlw   .48
```

```
        subwf   Byte1           ; 'Byte1' es alterado y ahora posee el valor del desplazamiento que se realizará
                                ; en el puntero.
Set_R_Pointer:
        bsf     STATUS,IRP      ; Se utilizará direccionamiento indirecto en los bancos 2 y 3.

        movf    FSR,W
        addwf   Byte1,W         ; Se realiza un desplazamiento en el puntero según el # de Temporizador a acceder.
        addwf   Byte1,W         ; (cada Temporizador utiliza dos bytes, por ello se suma dos veces).
        movwf   FSR             ; El puntero está posicionado en el lsB del Temporizador indicado por la
                                ; instrucción.

        btfss   Logic_Stack,7   ; Se determina si debe borrarse el Temporizador.
        goto    Instructions_Loop

        clrf    INDF            ; Se borra el Temporizador.
        incf    FSR
        clrf    INDF

        goto    Instructions_Loop
                                                        ; Banco 0.


INVB                            ; Instrucción "Invert Byte".

        call    Test_Flow_PC4   ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val     ; Se lee el byte de datos que se desea invertir.

        call    Get_FLASH_Data          ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        comf    ByteX,W         ; Se realiza el complemento.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write        ; Se escribe el resultado.
```

```
        goto    Instructions_Loop
                                                        ; Banco 0.

INVW                                    ; Instrucción "Invert Word".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val             ; Se lee el dato Word que se desea invertir.
        movf    ByteX,W
        movwf   Byte1                   ; lsB del dato a invertir.
        call    Get_RAM_Data
        movwf   Byte2                   ; msB del dato a invertir.

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        comf    Byte1,W                 ; Se realiza el complemento del lsB.
        bsf     CS                      ; Se selecciona el IC RAM.
        call    IO_Write        ; Se escribe el complemento del lsB.

        comf    Byte2,W
        call    Put_RAM_Data            ; Se ecribe el complemento del msB.

        goto    Instructions_Loop
                                                        ; Banco 0.

INVD                                    ; Instrucción "Invert Double".

        call    Test_Flow_PC4           ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.
```

```
        call    Get_Arg_Val             ; Se lee el dato Long que se desea invertir.
        movf    ByteX,W
        movwf   Byte1                   ; lsB del dato a invertir.
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3
        call    Get_RAM_Data
        movwf   Byte4                   ; msB del dato a invertir.

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        comf    Byte1,W                 ; Se realiza el complemento del lsB.
        bsf     CS                      ; Se selecciona el IC RAM.
        call    IO_Write        ; Se escribe el complemento del lsB.

        comf    Byte2,W
        call    Put_RAM_Data

        comf    Byte3,W
        call    Put_RAM_Data

        comf    Byte4,W
        call    Put_RAM_Data            ; Se ecribe el complemento del msB.

        goto    Instructions_Loop
                                                        ; Banco 0.


ANDB                                    ; Instrucción "AND Byte".

        movlw   0x00                    ; Se asigna el código para la operación lógica AND.
        movwf   BOOLE_Code
        goto    BOOLE_B
                                                        ; Banco 0.
```

```
ANDW                    ; Instrucción "AND Word".

        movlw   0x00    ; Se asigna el código para la operación lógica AND.
        movwf   BOOLE_Code
        goto    BOOLE_W
                                                ; Banco 0.

ANDD                    ; Instrucción "AND double".

        movlw   0x00    ; Se asigna el código para la operación lógica AND.
        movwf   BOOLE_Code
        goto    BOOLE_D
                                                ; Banco 0.

ORB                     ; Instrucción "OR Byte".

        movlw   0x01    ; Se asigna el código para la operación lógica OR.
        movwf   BOOLE_Code
        goto    BOOLE_B
                                                ; Banco 0.

ORW                     ; Instrucción "OR Word".

        movlw   0x01    ; Se asigna el código para la operación lógica OR.
        movwf   BOOLE_Code
        goto    BOOLE_W
                                                ; Banco 0.

ORD                     ; Instrucción "OR Double".
```

```
        movlw   0x01            ; Se asigna el código para la operación lógica OR.
        movwf   BOOLE_Code
        goto    BOOLE_D
                                                        ; Banco 0.

XORB                            ; Instrucción "XOR Byte".

        movlw   0x02            ; Se asigna el código para la operación lógica XOR.
        movwf   BOOLE_Code
        goto    BOOLE_B
                                                        ; Banco 0.

XORW                            ; Instrucción "XOR Word".

        movlw   0x02            ; Se asigna el código para la operación lógica XOR.
        movwf   BOOLE_Code
        goto    BOOLE_W
                                                        ; Banco 0.

XORD                            ; Instrucción "XOR Double".

        movlw   0x02            ; Se asigna el código para la operación lógica XOR.
        movwf   BOOLE_Code
        goto    BOOLE_D
                                                        ; Banco 0.

MOVB                            ; Instrucción "Move Byte".

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    MOVE_B_No_Const

        btfsc   Logic_Stack,7   ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_B_Const         ; SI existe flujo, por lo que se ejecutará la instrucción.
```

```
        movlw   0x03                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en tres (3) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_MOVE_B_Const:
        call    Get_FLASH_Data          ; Se lee la constante (primer parámetro).
        movwf   Byte1

        goto    MOVE_B_OUT


MOVE_B_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_B_No_Const      ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x04                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cuatro (4) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_MOVE_B_No_Const:
        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1

MOVE_B_OUT:
        call    Get_FLASH_Data          ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte1,W
        call    IO_Write        ; Se mueve el dato.
```

```
        goto    Instructions_Loop
                                                        ; Banco 0.

MOVW                                    ; Instrucción "Move Word".

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    MOVE_W_No_Const

        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_W_Const                 ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x04                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cuatro (4) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_MOVE_W_Const:
        call    Get_FLASH_Data                  ; Se lee la constante (primer parámetro).
        movwf   Byte1
        call    Get_FLASH_Data
        movwf   Byte2

        goto    MOVE_W_OUT


MOVE_W_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_W_No_Const ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x04                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cuatro (4) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.
```

```
Do_MOVE_W_No_Const:
        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2

MOVE_W_OUT:
        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte1,W
        call    IO_Write        ; Se mueve el dato.
        movf    Byte2,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                                ; Banco 0.


MOVDx

        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    MOVD


MOVR                                    ; Instrucción "Move Real".

        goto    MOVDx                   ; Para la rutina de movimiento de bytes, los datos de tipo Real y Double
                                        ; son indiferentes.


SHLB                                    ; Instrucción "Shift Left Byte".
```

```
        movlw   0x00            ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x00            ; Se asigna el código 'Left'.
        movwf   Direction_Code

        goto    Shift_Rotate_B
                                                        ; Banco 0.

SHRB                            ; Instrucción "Shift Right Byte".

        movlw   0x00            ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x01            ; Se asigna el código 'Right'.
        movwf   Direction_Code

        goto    Shift_Rotate_B
                                                        ; Banco 0.

RLB                             ; Instrucción "Rotate Left Byte".

        movlw   0x01            ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x00            ; Se asigna el código 'Left'.
        movwf   Direction_Code

        goto    Shift_Rotate_B
                                                        ; Banco 0.

RRB                             ; Instrucción "Rotate Right Byte".

        movlw   0x01            ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x01            ; Se asigna el código 'Right'.
        movwf   Direction_Code
```

```
        goto    Shift_Rotate_B
                                                        ; Banco 0.

SHLW                                    ; Instrucción "Shift Left Word".

        movlw   0x00                    ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x00                    ; Se asigna el código 'Left'.
        movwf   Direction_Code

        goto    Shift_Rotate_W
                                                        ; Banco 0.

SHRW                                    ; Instrucción "Shift Right Word".

        movlw   0x00                    ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x01                    ; Se asigna el código 'Right'.
        movwf   Direction_Code

        goto    Shift_Rotate_W
                                                        ; Banco 0.

RLW                                     ; Instrucción "Rotate Left Word".

        movlw   0x01                    ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x00                    ; Se asigna el código 'Left'.
        movwf   Direction_Code

        goto    Shift_Rotate_W
                                                        ; Banco 0.
```

```
RRW                                     ; Instrucción "Rotate Right Word".

        movlw   0x01                    ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x01                    ; Se asigna el código 'Right'.
        movwf   Direction_Code

        goto    Shift_Rotate_W

                                                ; Banco 0.

SHLD                                    ; Instrucción "Shift Left Double".

        movlw   0x00                    ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x00                    ; Se asigna el código 'Left'.
        movwf   Direction_Code

        goto    Shift_Rotate_D

                                                ; Banco 0.

SHRD                                    ; Instrucción "Shift Right Double".

        movlw   0x00                    ; Se asigna el código para la operación 'Shift'.
        movwf   Shift_Rotate_Code
        movlw   0x01                    ; Se asigna el código 'Right'.
        movwf   Direction_Code

        goto    Shift_Rotate_D

                                                ; Banco 0.

RLD                                     ; Instrucción "Rotate Left Double".

        movlw   0x01                    ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x00                    ; Se asigna el código 'Left'.
```

```
        movwf   Direction_Code

        goto    Shift_Rotate_D
                                                        ; Banco 0.

RRD                                     ; Instrucción "Rotate Right Double".

        movlw   0x01                    ; Se asigna el código para la operación 'Rotate'.
        movwf   Shift_Rotate_Code
        movlw   0x01                    ; Se asigna el código 'Right'.
        movwf   Direction_Code

        goto    Shift_Rotate_D
                                                        ; Banco 0.

INCB                                    ; Instrucción "Increment Byte".

        movlw   0x00                    ; Se asigna el código para incremento.
        movwf   Inc_Dec_Code
        goto    INC_DEC_B
                                                        ; Banco 0.

DECB                                    ; Instrucción "Decrement Byte".

        movlw   0x01                    ; Se asigna el código para decremento.
        movwf   Inc_Dec_Code
        goto    INC_DEC_B
                                                        ; Banco 0.

INCWx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    INCW
```

```
DECWx
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    DECW

INCDx
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    INCD

DECDx
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    DECD


ADD                             ; Instrucción "Add".

        movlw   0x0F            ; Comando ADD.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    Do_Math                 ; Se realiza la operación matemática.
                                                        ; Banco 0.

SUB                             ; Instrucción "Subtraction".

        movlw   0x0E            ; Comando SUB.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    Do_Math                 ; Se realiza la operación matemática.
                                                        ; Banco 0.

MULT                            ; Instrucción "Multiply".

        movlw   0x0C            ; Comando MUL.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    Do_Math                 ; Se realiza la operación matemática.
                                                        ; Banco 0.

DIV                             ; Instrucción "Divide".
        movlw   0x0D            ; Comando DIV.
        bcf     PCLATH,3        ; Se selecciona la página 0.
        goto    Do_Math                 ; Se realiza la operación matemática.
```

```
                                                    ; Banco 0.

CHS                                 ; Instrucción "Change Sign".

        movlw   0x0A                ; Comando CHS.
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    Do_Math6            ; Se realiza la operación matemática.
                                                    ; Banco 0.

ABS                                 ; Instrucción "Absolute Value".

        movlw   0x11                ; Comando ABS.
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    Do_Math6            ; Se realiza la operación matemática.
                                                    ; Banco 0.

SQRT                                ; Instrucción "Square Root".

        movlw   0x19                ; Comando SQRT.
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    Do_Math6            ; Se realiza la operación matemática.
                                                    ; Banco 0.

LOG                                 ; Instrucción "Natural Logarithm".

        movlw   0x1A                ; Comando LOG.
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    Do_Math6            ; Se realiza la operación matemática.
                                                    ; Banco 0.

LOG10                               ; Instrucción "Logarithm base 10".

        movlw   0x1B                ; Comando LOG10.
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    Do_Math6            ; Se realiza la operación matemática.
                                                    ; Banco 0.
```

```
EXP                                     ; Instrucción "Exponential base e".

        movlw   0x1C                    ; Comando EXP.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                                ; Banco 0.

EXP10                                   ; Instrucción "Exponential base 10".

        movlw   0x1D                    ; Comando EXP10.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                                ; Banco 0.

POW                                     ; Instrucción "Power".

        movlw   0x1E                    ; Comando POW.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math                         ; Se realiza la operación matemática.
                                                                ; Banco 0.

ROOT                                    ; Instrucción "Root".

        movlw   0x1F                    ; Comando ROOT.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math                         ; Se realiza la operación matemática.
                                                                ; Banco 0.

RECIP                                   ; Instrucción "Reciprocal".

        movlw   0x20                    ; Comando RECIP.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                                ; Banco 0.
```

```
SIN                                     ; Instrucción "Sine".

        movlw   0x21                    ; Comando SIN.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                        ; Banco 0.

ASIN                                    ; Instrucción "ArcSine".

        movlw   0x24                    ; Comando ASIN.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                        ; Banco 0.

COS                                     ; Instrucción "Cosine".

        movlw   0x22                    ; Comando COS.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                        ; Banco 0.

ACOS                                    ; Instrucción "ArcCosine".

        movlw   0x25                    ; Comando ACOS.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                        ; Banco 0.

TAN                                     ; Instrucción "Tangent".

        movlw   0x23                    ; Comando TAN.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                        ; Banco 0.
```

```
ATAN                                    ; Instrucción "ArcTangent".

        movlw   0x26                    ; Comando ATAN.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    Do_Math6                ; Se realiza la operación matemática.
                                                                ; Banco 0.

JMPx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    JMP

CALLSBRx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    CALLSBR

RETx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    RET

CRETx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    CRET

ENDPx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    ENDP
                                                                ; Banco 0.
STOPx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    STOP

WDTRx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    WDTR

RTCRDx
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    RTCRD
```

```
RTCWRx
        bcf     PCLATH,3            ; Se selecciona la página 0.
        goto    RTCWR


        org     0x0430              ; Las instrucciones definidas de aquí en adelante se encuentran almacenadas en
                                    ; la página 0 de la memoria de programa del PIC16F877.  Se colocan allí para
                                    ; "rellenar" el bloque de 1000 palabras de programa que se había dejado libre
                                    ; al realizar el salto con la directiva 'org 0x0800'.  El propósito de esto es
                                    ; lograr mayor legibilidad en el código fuente, pues todo el código que sigue
                                    ; completa el set de instrucciones incorporadas en el PLC (que se había
                                    ; iniciado en el bloque superior, con la directiva 'org 0x0800').


Get_10_Bytes_To_Do_Math             ; Rutina que lee TODOS los bytes requeridos por las instrucciones de
                                    ; funciones matemáticas de dos operandos y una dirección de salida.
                                    ; Además, esta rutina carga automáticamente en el coprocesador
                                    ; matemático los dos argumentos de la función.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    Get_10_Bytes_No_Const

        movlw   0x08                ; Si no existe flujo, el PC se debe incrementar en ocho (8) posiciones.
        call    Test_Flow_PCx       ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                      ; Se retoma a la función que llamó a esta rutina, la instrucción no será
                                    ; ejecutada luego que se compare el estado de 'Flow_On' ("apagado").

        bsf     PCLATH,3            ; Se selecciona la página 1.

        call    Get_FLASH_Data          ; SI existe un parámetro constante.  Se leerá el # real (constante)
        movwf   Byte1               ; en las variables que posteriormente serán utilizadas para el procesamiento.
        call    Get_FLASH_Data
        movwf   Byte2
        call    Get_FLASH_Data
        movwf   Byte3
        call    Get_FLASH_Data
        movwf   Byte4
```

```
        bcf     PCLATH,3            ; Se selecciona la página 0.

        goto    Get_10_Bytes_Arg2

Get_10_Bytes_No_Const:

        movlw   0x06                ; Si no exista flujo, el PC se debe incrementar en seis (6) posiciones.
        call    Test_Flow_PCx       ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                      ; Se retorna a la función que llamó a esta rutina, la instrucción no será
                                    ; ejecutada luego que se compare el estado de 'Flow_On' ("apagado").

        bsf     PCLATH,3            ; Se selecciona la página 1.

        call    Get_Arg_Val         ; NO existe un parámetro constante.  Se lee la palabra de datos (1er. byte).
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data        ; Se lee el segundo byte del primer parámetro.
        movwf   Byte2
        call    Get_RAM_Data        ; Se lee el tercer byte del primer parámetro.
        movwf   Byte3
        call    Get_RAM_Data        ; Se lee el cuarto byte del primer parámetro (msB).
        movwf   Byte4

Get_10_Bytes_Arg2:

        bsf     PCLATH,3            ; Se selecciona la página 1.

        call    Get_Arg_Val         ; Se lee el primer byte del segundo parámetro (lsB).
        movf    ByteX,W
        movwf   Byte5
        call    Get_RAM_Data        ; Se lee el segundo byte del segundo parámetro.
        movwf   Byte6
        call    Get_RAM_Data        ; Se lee el tercer byte del segundo parámetro.
        movwf   Byte7
        call    Get_RAM_Data        ; Se lee el cuarto byte del segundo parámetro (msB).
        movwf   Byte8

        call    Get_FLASH_Data              ; Se lee la dirección de salida.
        movwf   Byte9
        call    Get_FLASH_Data
```

```
movwf   Byte10

bcf     PCLATH,3        ; Se selecciona la página 0.

                        ; Se procede a cargar los argumentos en los registros X y Y del coprocesador
                        ; matemático.

movlw   0x01            ; Comando LOADX.
call    Pak_Tx

movf    Byte8,W         ; msB del argumento #2 (se asume que el dato ya está en IEEE745).
call    Pak_Tx

movf    Byte7,W
call    Pak_Tx

movf    Byte6,W
call    Pak_Tx

movf    Byte5,W         ; lsB del argumento #2.
call    Pak_Tx

movlw   0x09            ; Comando FPCVT (convierte X de IEEE745 a Pak).
call    Pak_Tx


movlw   0x04            ; Comando SWAP.
call    Pak_Tx


movlw   0x01            ; Comando LOADX.
call    Pak_Tx

movf    Byte4,W         ; msB del argumento #1 (se asume que el dato ya está en IEEE745).
call    Pak_Tx

movf    Byte3,W
call    Pak_Tx

movf    Byte2,W
call    Pak_Tx
```

```
        movf    Byte1,W         ; lsB del argumento #1.
        call    Pak_Tx

        movlw   0x09            ; Comando FPCVT (convierte X de IEEE745 a Pak).
        call    Pak_Tx

        return                  ; La página 0 está seleccionada.

                                ; Fin de la rutina "Get_10_Bytes_To_Do_Math".      ; Banco 0.


Get_6_Bytes_To_Do_Math          ; Rutina que lee TODOS los bytes requeridos por las instrucciones de
                                ; funciones matemáticas de un operando y una dirección de salida.
                                ; Además, esta rutina carga automáticamente en el coprocesador
                                ; matemático el argumento de la función.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    Get_6_Bytes_No_Const

        movlw   0x06            ; Si no existe flujo, el PC se debe incrementar en seis (6) posiciones.
        call    Test_Flow_PCx   ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                  ; Se retorna a la función que llamó a esta rutina, la instrucción no será
                                ; ejecutada luego que se compare el estado de 'Flow_On' ("apagado").

        bsf     PCLATH,3        ; Se selecciona la página 1.

        call    Get_FLASH_Data          ; Si existe un parámetro constante.  Se leerá el # real (constante)
        movwf   Byte1           ; en las variables que posteriormente serán utilizadas para el procesamiento.
        call    Get_FLASH_Data
        movwf   Byte2
        call    Get_FLASH_Data
        movwf   Byte3
        call    Get_FLASH_Data
        movwf   Byte4

        bcf     PCLATH,3        ; Se selecciona la página 0.
```

```
        goto    Get_6_Out_Address

Get_6_Bytes_No_Const:

        movlw   0x04            ; Si no existe flujo, el PC se debe incrementar en cuatro (4) posiciones.
        call    Test_Flow_PCx   ; Se determina si existe flujo para ejecutar esta instrucción.
        btfss   Flow_On
        return                  ; Se retorna a la función que llamó a esta rutina, la instrucción no será
                                ; ejecutada luego que se compare el estado de 'Flow_On' ("apagado").

        bsf     PCLATH,3        ; Se selecciona la página 1.

        call    Get_Arg_Val     ; NO existe un parámetro constante.  Se lee la palabra de datos (1er. byte).
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data    ; Se lee el segundo byte del primer parámetro.
        movwf   Byte2
        call    Get_RAM_Data    ; Se lee el tercer byte del primer parámetro.
        movwf   Byte3
        call    Get_RAM_Data    ; Se lee el cuarto byte del primer parámetro (msB).
        movwf   Byte4

Get_6_Out_Address:

        bsf     PCLATH,3        ; Se selecciona la página 1.

        call    Get_FLASH_Data          ; Se lee la dirección de salida.
        movwf   Byte9
        call    Get_FLASH_Data
        movwf   Byte10

        bcf     PCLATH,3        ; Se selecciona la página 0.

                                ; Se procede a cargar los argumentos en los registros X y Y del coprocesador
                                ; matemático.

        movlw   0x01            ; Comando LOADX.
        call    Pak_Tx

        movf    Byte4,W         ; msB del argumento #1 (se asume que el dato ya está en IEEE745).
```

```
        call    Pak_Tx

        movf    Byte3,W
        call    Pak_Tx

        movf    Byte2,W
        call    Pak_Tx

        movf    Byte1,W         ; lsB del argumento #1.
        call    Pak_Tx

        movlw   0x09            ; Comando FPCVT (convierte X de IEEE745 a Pak).
        call    Pak_Tx

        return                  ; La página 0 está seleccionada.

                                ; Fin de la rutina "Get_6_Bytes_To_Do_Math".    ; Banco 0.

Math_Result                     ; Rutina que lee y escribe el resultado de todas las instrucciones de funciones
                                ; matemáticas que utilizan dos argumentos y una dirección de salida.

        bcf     PCLATH,3        ; Se selecciona la página 0.

        call    Pak_Rx          ; Lectura del byte STATUS del comando ejecutado.
        movf    Receive_Pak,W
        btfss   STATUS,Z        ; Se determina si sucedió algún error en la operación matemática.
        goto    Math_Error      ; Sí ocurrió un error matemático.

        movlw   0x06            ; Comando IEEECVT (Convierte de Pak a IEEE745).
        call    Pak_Tx

        movlw   0x03            ; Comando READX.
        call    Pak_Tx

        call    Pak_Rx          ; Se recibe el número real IEEE745.
        movf    Receive_Pak,W
        movwf   Byte4           ; msB del real IEEE745.
```

```
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte3
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte2
        call    Pak_Rx
        movf    Receive_Pak,W
        movwf   Byte1                   ; lsB del real IEEE745.


        bsf     PCLATH,3                ; Se selecciona la página 1.

        movf    Byte9,W                 ; Se coloca la dirección de salida en el bus de direcciones del Sistema.
        movwf   Addr_Bus_Low
        movf    Byte10,W
        movwf   Addr_Bus_Hi
        call    Set_Address
        bsf     CS                      ; Se selecciona el IC RAM.

        movf    Byte1,W                 ; lsB del resultado (IEEE745).
        call    IO_Write        ; Se escribe el # real en el IC RAM.
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data            ; msB del resultado (IEEE745).

        return                          ; Se retorna en la página 1.

                                        ; Fin de la rutina "Math_Result".      ; Banco 0.


Do_Math6                                ; Este segmento de la rutina "Do_Math" es utilizado por las funciones
                                        ; matemáticas que requieren un único operando.

        movwf   Math_Code               ; Se almacena temporalmente el código de la función matemática.

        call    Get_6_Bytes_To_Do_Math          ; Se adquieren TODOS los parámetros de la función (4 de datos + 2 de salida).
```

```
        goto    Continue_With_Do_Math  ; Se continúa con el resto de la rutina normal.

Do_Math                                ; Rutina que realiza la operación matemática codificada en el registro W.

        movwf   Math_Code              ; Se almacena temporalmente el código de la función matemática.

        call    Get_10_Bytes_To_Do_Math ; Se adquieren TODOS los parámetros de la función.

Continue_With_Do_Math:

        bsf     PCLATH,3               ; Se selecciona la página 1.

        btfss   Flow_On                ; Se determina si existe flujo para ejecutar la instrucción.
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        bcf     PCLATH,3               ; Se selecciona la página 0.

        movf    Math_Code,W            ; Se envía el código de la función matemática.
        call    Pak_Tx

        call    Math_Result            ; Se lee y escribe el resultado de la operación matemática.

        return

                                       ; Fin de la rutina "Do_Math".          ; Banco 0.


Stack_Overflow_Error                   ; Rutina que realiza la notificación de error cuando se ha provocado un
                                       ; desborde en el Stack del Program Counter.
        movlw   0x2B
        goto    Show_System_Error
                                       ; Fin de la rutina "Stack_Overflow_Error".   Banco 0.


Stack_Underflow_Error                  ; Rutina que realiza la notificación de error cuando se ha provocado un
                                       ; desborde negativo en el Stack del Program Counter.
        movlw   0x2A
        goto    Show_System_Error
```

```
                                ; Fin de la rutina "Stack_Underflow_Error".   Banco 0.


MOVD                            ; Instrucción "Move Double".

        bcf     PCLATH,3        ; Se selecciona la página 0.

        btfss   Instruction_Full,7 ; Se determina si la instrucción posee un parámetro constante.
        goto    MOVE_D_No_Const

        btfsc   Logic_Stack,7   ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_D_Const         ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x06            ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low          ; Debe incrementarse el PC en seis (6) posiciones (contienen parámetros de
        btfsc   STATUS,C        ; la función).
        incf    PC_Hi

        bsf     PCLATH,3        ; Se selecciona la página 1.

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_MOVE_D_Const:

        bsf     PCLATH,3        ; Se selecciona la página 1.

        call    Get_FLASH_Data          ; Se lee la constante (primer parámetro).
        movwf   Byte1
        call    Get_FLASH_Data
        movwf   Byte2
        call    Get_FLASH_Data
        movwf   Byte3
        call    Get_FLASH_Data
        movwf   Byte4

        bcf     PCLATH,3        ; Se selecciona la página 0.

        goto    MOVE_D_OUT
```

```
MOVE_D_No_Const:
        btfsc   Logic_Stack,7           ; Se verifica si el tope de la pila está en uno.
        goto    Do_MOVE_D_No_Const      ; SI existe flujo, por lo que se ejecutará la instrucción.

        movlw   0x04                    ; NO existe flujo.  NO debe ejecutarse la instrucción.
        addwf   PC_Low                  ; Debe incrementarse el PC en cuatro (4) posiciones (contienen parámetros de
        btfsc   STATUS,C                ; la función).
        incf    PC_Hi

        bsf     PCLATH,3                ; Se selecciona la página 1.

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de las demás instrucciones del programa.

Do_MOVE_D_No_Const:

        bsf     PCLATH,3                ; Se selecciona la página 1.

        call    Get_Arg_Val             ; Se lee el primer parámetro.
        movf    ByteX,W
        movwf   Byte1
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3
        call    Get_RAM_Data
        movwf   Byte4

MOVE_D_OUT:

        bsf     PCLATH,3                ; Se selecciona la página 1.

        call    Get_FLASH_Data                  ; Se lee la dirección donde se desea colocar el resultado.
        movwf   Addr_Bus_Low
        call    Get_FLASH_Data
        movwf   Addr_Bus_Hi
        call    Set_Address

        bsf     CS                      ; Se selecciona el IC RAM.
        movf    Byte1,W
        call    IO_Write        ; Se mueve el dato.
        movf    Byte2,W
```

```
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movf    Byte4,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                        ; Banco 0.

INCW                                    ; Instrucción "Increment Word".

        movlw   0x02                    ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
        call    Test_Flow_PCx           ; Se determina si existe flujo para ejecutar esta instrucción.
        bsf     PCLATH,3                ; Se selecciona la página 1.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val             ; Se lee el valor de la palabra a incrementar.
        movf    ByteX,W
        movwf   Byte1                   ; lsB de la palabra a incrementar.
        call    Get_RAM_Data
        movwf   Byte2                   ; msB de la palabra a incrementar.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        btfss   Byte2,7                 ; Se determina si se debe sumar o restar uno (1), dependiendo del signo de la
        goto    Inc_Negative_W          ; palabra a incrementar.

Inc_Positive_W:
        incf    Byte1
        btfsc   STATUS,Z
        incf    Byte2

        goto    INCW_End


Inc_Negative_W:
        movlw   0x01
        subwf   Byte1
        btfss   STATUS,C
```

```
        decf    Byte2

        movf    Byte1,W         ; Se determina si la palabra es igual a 0x0000, de ser así, debe hacerse
        btfsc   STATUS,Z        ; positiva (era negativa.)
        movf    Byte2,W
        btfsc   STATUS,Z
        bsf     Byte2,7         ; Se coloca el signo positivo.

INCW_End:
        bsf     PCLATH,3        ; Se selecciona la página 1.
        movlw   0x01            ; Se decrementa el PC para que apunte al lsB de la palabra original.
        subwf   Addr_Bus_Low
        btfss   STATUS,C
        decf    Addr_Bus_Hi
        call    Set_Address

        movf    Byte1,W         ; Se escribe el valor actualizado.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write
        movf    Byte2,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                        ; Banco 0.

DECW                            ; Instrucción "Decrement Word".

        movlw   0x02            ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
        call    Test_Flow_PCx   ; Se determina si existe flujo para ejecutar esta instrucción.
        bsf     PCLATH,3        ; Se selecciona la página 1.
        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val     ; Se lee el valor de la palabra a decrementar.
        movf    ByteX,W
        movwf   Byte1           ; lsB de la palabra a decrementar.
        call    Get_RAM_Data
        movwf   Byte2           ; msB de la palabra a decrementar.
```

```
        bcf     PCLATH,3                ; Se selecciona la página 0.

        btfss   Byte2,7                 ; Se determina si se debe sumar o restar uno (1), dependiendo del signo de la
        goto    Dec_Negative_W          ; palabra a decrementar.

Dec_Positive_W:
        movlw   0x01
        subwf   Byte1
        btfss   STATUS,C
        goto    Check_If_W_UnderFlow    ; Se determina si se debe hacer el cambio de signo, de '+' a '-' y colocar el
                                        ; valor en -1.

        goto    DECW_End


Check_If_W_UnderFlow:
        decf    Byte2                   ; Se determina si la palabra es igual a 0x7FFF, de ser asi, debe hacerse
        movlw   0x7F                    ; negativa (era positiva.) y colocarse en el valor -1.
        subwf   Byte2,W
        btfss   STATUS,Z
        goto    DECW_End

        movlw   0x01                    ; Se asigna el valor 0x0001 a la palabra ya decrementada (valor = -1).
        movwf   Byte1
        movlw   0x00
        movwf   Byte2

        goto    DECW_End


Dec_Negative_W:
        incf    Byte1
        btfsc   STATUS,Z
        incf    Byte2


DECW_End:
        bsf     PCLATH,3                ; Se selecciona la página 1.
        movlw   0x01                    ; Se decrementa el PC para que apunte al lsB de la palabra original.
        subwf   Addr_Bus_Low
        btfss   STATUS,C
```

```
        decf    Addr_Bus_Hi
        call    Set_Address

        movf    Byte1,W                 ; Se escribe el valor actualizado.
        bsf     CS                      ; Se selecciona el IC RAM.
        call    IO_Write
        movf    Byte2,W
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                                ; Banco 0.

INCD                                    ; Instrucción "Increment Double".

        movlw   0x02                    ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
        call    Test_Flow_PCx           ; Se determina si existe flujo para ejecutar esta instrucción.
        bsf     PCLATH,3                ; Se selecciona la página 1.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val             ; Se lee el valor del double a incrementar.
        movf    ByteX,W
        movwf   Byte1                   ; lsB del double a incrementar.
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3                   ; msB del double a incrementar.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        btfss   Byte3,7                 ; Se determina si se debe sumar o restar uno (1), dependiendo del signo del
        goto    Inc_Negative_D          ; double a incrementar.

Inc_Positive_D:
        incf    Byte1
        btfsc   STATUS,Z
        incf    Byte2
        btfsc   STATUS,Z
        incf    Byte3
        btfsc   STATUS,Z
```

```
        incf    Byte4

        goto    INCD_End


Inc_Negative_D:
        movlw   0x01
        subwf   Byte1
        btfss   STATUS,C
        decf    Byte2

        movf    Byte1,W         ; Se determina si el double es igual a 0x00000000, de ser así, debe hacerse
        btfsc   STATUS,Z        ; positivo (era negativo.)
        movf    Byte2,W
        btfsc   STATUS,Z
        movf    Byte3,W
        btfsc   STATUS,Z
        bsf     Byte2,7         ; Se coloca el signo positivo.


INCD_End:
        bsf     PCLATH,3        ; Se selecciona la página 1.
        movlw   0x02            ; Se decrementa el PC para que apunte al lsB del double original.
        subwf   Addr_Bus_Low
        btfss   STATUS,C
        decf    Addr_Bus_Hi
        call    Set_Address

        movf    Byte1,W         ; Se escribe el valor actualizado.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movlw   0x00
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                ; Banco 0.
```

```
DECD                                    ; Instrucción "Decrement Double".

        movlw   0x02                    ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
        call    Test_Flow_PCx           ; Se determina si existe flujo para ejecutar esta instrucción.
        bsf     PCLATH,3                ; Se selecciona la página 1.
        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        call    Get_Arg_Val             ; Se lee el valor del double a decrementar.
        movf    ByteX,W
        movwf   Byte1                   ; lsB del double a decrementar.
        call    Get_RAM_Data
        movwf   Byte2
        call    Get_RAM_Data
        movwf   Byte3                   ; msB del double a decrementar.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        btfss   Byte3,7                 ; Se determina si se debe sumar o restar uno (1), dependiendo del signo del
        goto    Dec_Negative_D          ; double a decrementar.

Dec_Positive_D:
        movlw   0x01
        subwf   Byte1
        btfss   STATUS,C
        subwf   Byte2
        btfss   STATUS,C
        goto    Check_If_D_UnderFlow    ; Se determina si se debe hacer el cambio de signo, de '+' a '-' y colocar el
                                        ; valor en -1.

        goto    DECD_End


Check_If_D_UnderFlow:
        decf    Byte3                   ; Se determina si el double es igual a 0x007FFFFF, de ser así, debe hacerse
        movlw   0x7F                    ; negativo (era positivo.) y colocarse en el valor -1.
        subwf   Byte3,W
        btfss   STATUS,Z
        goto    DECD_End
```

```
        movlw   0x01            ; Se asigna el valor 0x00000001 al double ya decrementado (valor = -1).
        movwf   Byte1
        movlw   0x00
        movwf   Byte2
        movwf   Byte3

        goto    DECD_End

Dec_Negative_D:
        incf    Byte1
        btfsc   STATUS,Z
        incf    Byte2
        btfsc   STATUS,Z
        incf    Byte3


DECD_End:
        bsf     PCLATH,3        ; Se selecciona la página 1.
        movlw   0x02            ; Se decrementa el PC para que apunte al lsB del double original.
        subwf   Addr_Bus_Low
        btfss   STATUS,C
        decf    Addr_Bus_Hi
        call    Set_Address

        movf    Byte1,W         ; Se escribe el valor actualizado.
        bsf     CS              ; Se selecciona el IC RAM.
        call    IO_Write
        movf    Byte2,W
        call    Put_RAM_Data
        movf    Byte3,W
        call    Put_RAM_Data
        movlw   0x00
        call    Put_RAM_Data

        goto    Instructions_Loop
                                                        ; Banco 0.


JMP                             ; Instrucción "Jump to Label".

        movlw   0x02            ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
```

```
        call    Test_Flow_PCx           ; Se determina si existe flujo para ejecutar esta instrucción.

        bsf     PCLATH,3                ; Se selecciona la página 1.

        btfss   Flow_On
        goto    Instructions_Loop; Se retoma para continuar con la ejecución de las demás instrucciones.

        call    Get_FLASH_Data                  ; Se lee la dirección hacia donde se trasladará la ejecución del programa.
        movwf   Byte1           ; (Se ASUME que el Compilador proporciona la dirección menos una (1) posición,
        call    Get_FLASH_Data                  ; contemplando que antes de ejecutar la instrucción deseada, el Sistema
        movwf   Byte2           ; incrementa el PC en una posición).

        movf    Byte1,W         ; Se modifica el Program Counter (PC).
        movwf   PC_Low
        movf    Byte2,W
        movwf   PC_Hi

        goto    Instructions_Loop; Se retoma al ciclo de ejecución de instrucciones con el valor del PC
                                ; ya modificado.
                                                                ; Banco 0.

CALLSBR                                 ; Instrucción "Call Subroutine".

        movlw   0x02            ; Si no existe flujo, el PC se debe incrementar en dos (2) posiciones.
        call    Test_Flow_PCx   ; Se determina si existe flujo para ejecutar esta instrucción.

        bsf     PCLATH,3        ; Se selecciona la página 1.

        btfss   Flow_On
        goto    Instructions_Loop; Se retorna para continuar con la ejecución de las demás instrucciones.

        movf    Logic_Stack,W   ; Se crea una copia de la pila lógica.
        movwf   LSBckUp

        movlw   0x80            ; Se ponen ceros (0) en todas las posiciones de la pila, excepto en el tope.
        movwf   Logic_Stack

        call    Get_FLASH_Data                  ; Se lee la dirección hacia donde se trasladará la ejecución del programa.
        movwf   Byte1           ; (Se ASUME que el Compilador proporciona la dirección menos una (1) posición,
        call    Get_FLASH_Data                  ; contemplando que antes de ejecutar la instrucción deseada, el Sistema
```

```
movwf   Byte2                   ; incrementa el PC en una posición).

movlw   0xE0                    ; Se asigna la dirección de inicio del Stack del PC para almacenar la
movwf   Addr_Bus_Low            ; dirección de retorno luego de ejecutar la subrutina que se está llamando.
movlw   0x7D
movwf   Addr_Bus_Hi

movf    Stack_Depth,W           ; Se suma el "offset" correspondiente al # de subrutinas ya anidadas.
addwf   Addr_Bus_Low
btfsc   STATUS,C
incf    Addr_Bus_Hi

call    Set_Address
bsf     CS                      ; Se selecciona el IC RAM.

movf    PC_Low,W                ; Se almacena la dirección de retorno en el Stack del PC.
call    IO_Write        ; Nótese que es la posición del msB de la dirección hacia donde se traslada
movf    PC_Hi,W                         ; la ejecución del programa (una posición antes de la siguiente instrucción).
call    Put_RAM_Data

movlw   0x02                    ; Se suman dos posiciones a la profundidad del Stack del PC.
addwf   Stack_Depth

movlw   .34                     ; Se determina si se ha provocado un desborde del Stack del PC.
subwf   Stack_Depth,W

bcf     PCLATH,3                ; Se selecciona la página 0.

btfsc   STATUS,C
goto    Stack_Overflow_Error    ; Se ha desbordado el Stack del PC.

bsf     PCLATH,3                ; Se selecciona la página 1.

movf    Byte1,W                 ; NO se ha provocado un desborde del Stack del PC, por lo que se cargará
movwf   PC_Low                  ; el nuevo valor para el PC.
movf    Byte2,W
movwf   PC_Hi

goto    Instructions_Loop; Se retorna al ciclo de ejecución de instrucciones con el valor del PC
                        ; ya modificado.
                                                        ; Banco 0.
```

```
RET                                     ; Instrucción "Return from Subroutine".

        movlw   0x02                    ; Se determina si existe por lo menos una dirección que extraer del Stack
        subwf   Stack_Depth,W           ; del PC.  De no ser así, se produce un error de "Stack Underflow".
        btfss   STATUS,C
        goto    Stack_Underflow_Error   ; Se intentó retornar de una subrutina sin antes haber llamado a ésta.

        movlw   0xDE                    ; Se lee la dirección hacia donde debe retornar la ejecución del programa.
        movwf   Addr_Bus_Low
        movlw   0x7D
        movwf   Addr_Bus_Hi
        movf    Stack_Depth,W
        addwf   Addr_Bus_Low            ; Se apunta al lsB de la dirección de retorno.
        btfsc   STATUS,C
        incf    Addr_Bus_Hi

        bsf     PCLATH,3                ; Se selecciona la página 1.

        call    Set_Address
        bsf     CS                      ; Se selecciona el IC RAM.
        call    IO_Read                         ; Se lee el lsB de la dirección de retorno.
        movwf   Byte1
        call    Get_RAM_Data            ; Se lee el msB de la dirección de retorno.
        movwf   Byte2

        movlw   0x02                    ; Se decrementa la profunidad del Stack del PC en dos posiciones.
        subwf   Stack_Depth             ; (debidas a la extracción de la dirección de retorno).

        movf    LSBckUp,W               ; Se restablece la pila lógica.
        movwf   Logic_Stack

        movf    Byte1,W                 ; Se carga el nuevo valor para el PC.
        movwf   PC_Low
        movf    Byte2,W
        movwf   PC_Hi

        goto    Instructions_Loop; Se retorna al ciclo de ejecución de instrucciones con el valor del PC
                                 ; ya modificado.
                                                                ; Banco 0.
```

```
CRET                                    ; Instrucción "Conditional Return from Subroutine".

        bsf     PCLATH,3                ; Se selecciona la página 1.

        btfss   Logic_Stack,7           ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop; NO existe flujo, por lo que se retorna al ciclo de instrucciones.
        bcf     PCLATH,3                ; Se selecciona la página 0.
        goto    RET                     ; SI existe flujo, por lo que se retorna de la subrutina actual.

                                                                        ; Banco 0.

ENDP                                    ; Instrucción "End Program".

        bsf     PCLATH,3                ; Se selecciona la página 1.

        btfss   Logic_Stack,7           ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop; NO existe flujo, por lo que se retorna al ciclo de instrucciones.
        return                          ; Se retorna del ciclo de ejecución de las instrucciones, con lo
                                        ; cual se finaliza la ejecución del programa y se retoma al ciclo
                                        ; del PLC, que procederá con la fase de escritura a las salidas
                                        ; digitales.

                                                                        ; Banco 0.

STOP                                    ; Instrucción "Stop System".

        bsf     PCLATH,3                ; Se selecciona la página 1.

        btfss   Logic_Stack,7           ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop; NO existe flujo, por lo que se retorna al ciclo de instrucciones.

        bcf     PCLATH,3                ; Se selecciona la página 0.

        banksel TXREG                   ; Se selecciona el modo IO6=0, IO5=1 en el módulo SP, de tal forma que se
                                        ; active el modo 'STOP'.

        movlw   0x90                    ; Comando 'SP WriteX'.
```

```
        movwf   TXREG
        call    Poll_Tx
        movlw   0x17                    ; lsB de la dirección del pin IO6 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF                    ; msB de la dirección del pin IO6 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x00
        movwf   TXREG                   ; Se escribe el valor cero (0) en el pin IO6.
        call    Poll_Tx

        movlw   0x90                    ; Comando 'SP WriteX'.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x16                    ; lsB de la dirección del pin IO5 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0xFF                    ; msB de la dirección del pin IO5 del módulo SP.
        movwf   TXREG
        call    Poll_Tx
        movlw   0x01
        movwf   TXREG                   ; Se escribe el valor uno (1) en el pin IO5.
        call    Poll_Tx

        goto    SYSTEM_Stopped                  ; El Sistema se traslada al ciclo del modo 'STOP'.

                                                                        ; Banco 0.

WDTR                                    ; Instrucción "Watchdog Timer Reset".

        bsf     PCLATH,3                ; Se selecciona la página 1.

        btfss   Logic_Stack,7           ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop; NO existe flujo, por lo que se retorna al ciclo de instrucciones.

        clrf    Watchdog_Counter        ; Se borra el contador de vigilancia de tiempo de ciclo.

        goto    Instructions_Loop

                                                                        ; Banco 0.
```

```
RTCRD                                   ; Instrucción que lee el reloj de tiempo real (13 bytes).
        banksel Addr_Bus_Low

        bsf     PCLATH,3                ; Se selecciona la página 1 de la memoria de programa.

        btfss   Logic_Stack,7           ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop

        bcf     PCLATH,3                ; Se selecciona la página 0 de la memoria de programa.

        movlw   0xD3                    ; lsB de inicio del área de memoria del RTC (memoria R).
        movwf   Addr_Bus_Low
        movlw   0x7D                    ; msB de inicio del área de memoria del RTC.
        movwf   Addr_Bus_Hi

        movlw   0x00                    ; Dirección del primer registro del RTC.
        movwf   Temp2
Read_Regs_Loop:
        banksel TRISB
        movlw   b'11000000'             ; Se realizará una escritura.
        movwf   TRISB
        banksel Temp2
        movf    Temp2,W
        movwf   PORTB                   ; Se direcciona el registro i a leer.
        bsf     RTC_AWr
        nop
        nop
        nop
        nop
        nop
        bcf     RTC_AWr

        banksel TRISB
        movlw   b'11001111'             ; Se realizará una lectura.
        movwf   TRISB

        banksel PORTB
        bsf     RTC_Rd
```

```
        bsf     PCLATH,3            ; Se selecciona la página 1 de la memoria de programa.

        call    Set_Address

        movf    PORTB,W
        andlw   0x0F                ; Se enmascaran los cuatro bits de datos del RTC.
        bsf     CS                  ; Se selecciona el IC RAM.
        call    IO_Write        ; Se almacena el registro i leído desde el RTC.

        bcf     PCLATH,3            ; Se selecciona la página 0 de la memoria de programa.

        bcf     RTC_Rd

        incf    Temp2               ; Se direcciona el siguiente registro del RTC.
        incf    Addr_Bus_Low        ; Se direcciona el siguiente registro de la memoria R.
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi

        movf    Temp2,W
        sublw   0x0D                ; Se determina si ya fueron leídos todos los registros del RTC (13).
        btfss   STATUS,Z
        goto    Read_Regs_Loop

        bsf     PCLATH,3            ; Se selecciona la página 1 de la memoria de programa.

        goto    Instructions_Loop
                                                        ; Banco 0.

RTCWR                               ; Instrucción que escribe en el reloj de tiempo real (13 bytes).
        banksel Addr_Bus_Low

        bsf     PCLATH,3            ; Se selecciona la página 1 de la memoria de programa.

        btfss   Logic_Stack,7       ; Se determina si existe flujo para la ejecución de esta instrucción.
        goto    Instructions_Loop

        bcf     PCLATH,3            ; Se selecciona la página 0 de la memoria de programa.

        movlw   0xD3                ; lsB de inicio del área de memoria del RTC (memoria R).
        movwf   Addr_Bus_Low
```

```
        movlw   0x7D                ; msB de inicio del área de memoria del RTC.
        movwf   Addr_Bus_Hi

        movlw   0x00                ; Dirección del primer registro del RTC.
        movwf   Temp2
Write_Regs_Loop:
        banksel TRISB
        movlw   b'11000000'         ; Se realizarán dos escrituras.
        movwf   TRISB
        banksel Temp2
        movf    Temp2,W
        movwf   PORTB               ; Se direcciona el registro i a escribir.
        bsf     RTC_AWr
        nop
        nop
        nop
        nop
        nop
        nop
        bcf     RTC_AWr

        bsf     PCLATH,3            ; Se selecciona la página 1 de la memoria de programa.

        call    Set_Address
        bsf     CS                  ; Se selecciona el IC RAM.
        call    IO_Read                     ; Se lee el registro i que se escribirá desde el área de memoria R.

        bcf     PCLATH,3            ; Se selecciona la página 0 de la memoria de programa.

        andlw   0x0F                ; Se enmascaran los cuatro bits de datos del RTC.
        movwf   PORTB

        bsf     RTC_Wr              ; Se activa la señal de escritura hacia el RTC.

        movlw   0x05
        movwf   Ac0
RTC_Wr_Loop:
        decfsz  Ac0
        goto    RTC_Wr_Loop

        bcf     RTC_Wr
```

```
        incf    Temp2           ; Se direcciona el siguiente registro del RTC.
        incf    Addr_Bus_Low    ; Se direcciona el siguiente registro de la memoria R.
        btfsc   STATUS,Z
        incf    Addr_Bus_Hi

        movf    Temp2,W
        sublw   0x0D            ; Se determina si ya fueron escritos todos los registros del RTC (13).
        btfss   STATUS,Z
        goto    Write_Regs_Loop

        bsf     PCLATH,3        ; Se selecciona la página 1 de la memoria de programa.

        goto    Instructions_Loop
                                                        ; Banco 0.


Monitor_System                  ; Sistema de transmisión de variables RAM, para la atención de las
                                ; solicitudes de monitoreo remoto.
        banksel Object_Address

        movf    PORTB,W                 ; Si el Sistema se encuentra en el modo 'PROG' o en el modo 'ERASE',
        btfsc   PORTB,7                 ; NO debe ejecutarse la fase de monitoreo, pues provoca conflictos
        retum                   ; con los datos seriales de programación/borrado.

        bsf     In_System_Monitor       ; El Sistema SI está en la fase de monitoreo.

        movlw   0x04            ; Se incrementan cuatro (4) posiciones a la dirección del
        addwf   Object_Address  ; objeto de monitoreo a leer (longitud del objeto anterior).

        movlw   0xB0            ; Se determinará si se debe reinicializar el puntero de objetos
        subwf   Object_Address,W ; de monitoreo.
        btfss   STATUS,Z
        goto    Not_Monitor_End

Monitor_End:
        movlw   0x0C            ; Se reinicializa el puntero de objetos.  Con la próxima llamada a
        movwf   Object_Address  ; esta rutina se reiniciará el ciclo de monitoreo.

Not_Monitor_End:
```

```
banksel TXREG
movlw   0xC2                ; Comando 'SP Read'.
movwf   TXREG
call    Poll_Tx

bcf     INTCON,GIE          ; Se deshabilitan las interrupciones del Sistema.

movf    Object_Address,W    ; Dirección del objeto de monitoreo a leer (dirección de variable).
movwf   TXREG
call    Poll_Tx

call    Poll_Rx             ; Se lee el lsB de la variable a monitorear.
movf    RCREG,W
movwf   Addr_Bus_Low

call    Poll_Rx             ; Se lee el msB de la variable a monitorear.
movf    RCREG,W
movwf   Addr_Bus_Hi

call    Poll_Rx             ; Se lee el tipo de dato de la variable a monitorear.
movf    RCREG,W
movwf   Byte1

btfsc   Addr_Bus_Hi,7       ; Se determina si se debe procesar el objeto actual.
retum

movlw   0x8F                ; Comando 'SP WriteX' (menos uno).
addwf   Byte1,W             ; Offset que indica la cantidad de bytes que se escribirán.

movwf   TXREG               ; Se transmite el comando 'SP WriteX'.
call    Poll_Tx

movf    Object_Address,W
movwf   TXREG               ; Se transmite el lsB de la dirección del objeto a escribir.
call    Poll_Tx

movlw   0x01
movwf   TXREG               ; Se transmite el msB de la dirección del objeto a escribir.
call    Poll_Tx

movlw   0x01
```

```
        subwf   Addr_Bus_Low        ; Se decrementa el byte bajo del bus de direcciones para
        btfss   STATUS,C            ; utilizar la rutina "Get_RAM_Data".
        decf    Addr_Bus_Hi

Read_Send_Variable_Loop:
        bsf     PCLATH,3            ; Se selecciona la página 1 de la memoria de programa.
        call    Get_RAM_Data        ; Se lee el byte i de la variable a monitorear.
        bcf     PCLATH,3            ; Se selecciona la página 0 de la memoria de programa.
        movwf   TXREG               ; Se transmite el byte i de la variable a monitorear.
        call    Poll_Tx

        decfsz  Byte1
        goto    Read_Send_Variable_Loop

        return
                                    ; Fin de la rutina "Monitor_System". Banco 0.


        END
```

****

Software de bajo nivel. Firmware del Módulo de Expansión Analógico

****

```
LIST P=16F877
    INCLUDE P16F877.INC

    __CONFIG _WDT_OFF & _HS_OSC & _CP_OFF & _LVP_OFF & _PWRTE_ON & _BODEN_ON

        #DEFINE     This_Module_In  PORTB,0
        #DEFINE     Next_Module_Out     PORTB,1
        #DEFINE     Busy_Module     PORTB,2
        #DEFINE     Clk0            PORTB,3
        #DEFINE     Clk1            PORTB,4
        #DEFINE     Clk2            PORTB,5


    CBLOCK 0X20
```

```
                SPI_Command
                SPI_Byte_In
                Module_Address
                Analog_Output_Channel

        ENDC


            CBLOCK      0X70

                W_temp
                STATUS_Temp
                FSR_Temp
                INDFx
                Pointer
                Ac0
                AC1
                AC2

            ENDC


        ORG 0
        goto        Slave_SPI_Init

SPI_Poll
        banksel SSPSTAT             ; El tiempo máximo para que el esclavo salga de esta rutina, luego de haberse recibido
        btfss   SSPSTAT,BF  ; el byte, es de 2.6 uS.
        goto    SPI_Poll
        banksel SSPBUF
        movf    SSPBUF,W    ; Se lee el búfer de transmisión para borrar la bandera de recepción.
        movwf   SPI_Byte_In ; Se almacena el byte recibido.

        return
                            ; Fin de la rutina "SPI_Poll".              Banco 0.

Wait_Acquisition        ; Rutina que espera 20 uS para el tiempo de adquisición de la señal analógica (ADC).
        movlw   .33
```

```
        movwf   Ac0
Acq_Loop:
        decfsz  Ac0
        goto    Acq_Loop

        return
                                ; Fin de la rutina "Wait_Acquisition".          No se altera la selección del banco.

Test_Command                    ; Rutina que determina si el primer byte de cada ciclo enviado por la cpu central le
                                ; corresponde al módulo de expansión, de ser así, evalúa el comando recibido, recibe
                                ; el byte siguiente (Byte #2), y lee/escribe el Byte #3 según corresponda al caso.

        bcf     Busy_module     ; Se indica que el módulo aún no está listo para la transacción.

        movf    SPI_Byte_In,W
        andlw   b'11110000'     ; Se extrae el código del comando recibido.
        movwf   SPI_Command     ; Se almacena el código del comando recibido.

        movf    SPI_Byte_In,W
        andlw   b'00001111'     ; Se extrae la dirección del módulo de expansión destino para el Byte #1.
        subwf   Module_Address,W; Se verifica si la transacción debe realizarse con este módulo.
        btfsc   STATUS,Z
        goto    Module_Addressed; Si la dirección es la de este módulo, entonces se reciben los dos bytes siguientes.
        call    SPI_Poll ; Si la dirección no es la de este módulo, entonces se ignoran los dos bytes siguientes.
        call    SPI_Poll

        return                  ; Se retorna al ciclo principal del módulo de expansión.

Not_Valid_Command:
        call    SPI_Poll
        call    SPI_Poll

        return


Module_Addressed:
        clrf    PCLATH          ; Se utilizará direccionamiento relativo en el primer bloque de 256 bytes.
        bcf     STATUS,C
        rrf     SPI_Command
        rrf     SPI_Command
        rrf     SPI_Command
```

```
        rrf     SPI_Command

        movf    SPI_Command,W
        addwf   PCL
        goto    Read_Module_ID
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Read_Analog_Input
        goto    Write_Analog_Output
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command

        ; Nota:                 Si el comando no es válido, simplemente se ignoran los dos bytes siguientes, por lo
                              ; que no se manipulará la señal Busy_Module.  Cuando se cumpla el tiempo Tm, la cpu
                              ; central notificará el error de E/S.
                              ; (pues la cpu central no recibirá la señal de ACK del módulo, en el pin RC1).

Read_Analog_Input:

        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (# de elemento a acceder).
        movlw   .8              ; Se verificará que el # de canal analógico a acceder no sea mayor que 7 (no confundirse
        subwf   SPI_Byte_In,W   ; por el hecho de que la instrucción "movlw" tenga un número 8).
        btfsc   STATUS,C        ; Si el # de canal no es válido, se retorna al ciclo del módulo, sin enviar la señal ACK
        return                  ; a la cpu central, lo que provocará una notificación de error de E/S.

        bcf     ADCON0,CHS0     ; Puesto que el # de canal es válido, se procede con la conversión A/D.
        bcf     ADCON0,CHS1
        bcf     ADCON0,CHS2
        btfsc   SPI_Byte_In,0   ; Se selecciona el canal que se desea acceder.
        bsf     ADCON0,CHS0
        btfsc   SPI_Byte_In,1
        bsf     ADCON0,CHS1
```

```
        btfsc   SPI_Byte_In,2
        bsf     ADCON0,CHS2

        bsf     ADCON0,ADON  ; Se enciende el módulo del ADC.
        call    Wait_Acquisition
        bsf     ADCON0,Go_Done        ; Se inicia la conversión A/D.

Poll_AD:                ; Se espera hasta que la conversión A/D se haya finalizado.
        btfsc   ADCON0,Go_Done
        goto    Poll_AD

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN         ; Se reconfigura el módulo SSP con el pin SDO como salida de datos.
        bcf     TRISC,5
        bsf     SSPSTAT,SSPEN

        banksel ADRESH
        movf    ADRESH,W      ; Se carga el resultado en el buffer de transmisión SPI.
        movwf   SSPBUF

        bsf     Busy_Module     ; Se indica que el resultado está listo para ser leído por la cpu central.

        call    SPI_Poll ; Se espera a que la cpu central lea el byte del resultado de la conversión A/D.

        bcf     ADCON0,ADON  ; Se apaga el módulo del ADC.

        bcf     Busy_Module     ; Ya finalizada la transacción se indica que el módulo no tiene resultados disponibles.

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN         ; Se reconfigura el módulo SSP con el pin SDO como entrada de datos.
        bsf     TRISC,5
        bsf     SSPSTAT,SSPEN

        return                  ; Se retoma al ciclo principal del módulo de expansión.


Write_Analog_Output:

        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (# de elemento a acceder).
```

```
        movlw   .3              ; Se verificará que el # de canal analógico a acceder no sea mayor que 2 (no se debe
        subwf   SPI_Byte_In,W   ; confundir con el número 3 que aparece en la instrucción "movlw").
        btfss   STATUS,C
        goto    Channel_OK

        call    SPI_Poll ; Canal analógico de salida no válido.  Se recibe el byte que se desea escribir, y luego
        retum                   ; se retoma al ciclo principal del módulo de expansión, sin enviar la señal ACK.

Channel_OK:
        movf    SPI_Byte_In,W   ; Se almacena el # de elemento a acceder.
        movwf   Analog_Output_Channel
        call    SPI_Poll ; Se recibe el byte a escribir en la salida analógica.

        movf    SPI_Byte_In,W   ; Se escribe en el puerto D el valor para la salida analógica.
        movwf   PORTD

        clrf    PCLATH
        movf    Analog_Output_Channel,W
        addwf   PCL
        goto    Output_Channel0
        goto    Output_Channel1
        goto    Output_Channel2

Output_Channel0:
        bcf     Clk0            ; Se genera el pulso de reloj para cargar los datos en el DAC del canal 0.
        nop
        nop
        bsf     Clk0
        goto    Output_Channel_End


Output_Channel1:
        bcf     Clk1            ; Se genera el pulso de reloj para cargar los datos en el DAC del canal 1.
        nop
        nop
        bsf     Clk1
        goto    Output_Channel_End


Output_Channel2:
        bcf     Clk2            ; Se genera el pulso de reloj para cargar los datos en el DAC del canal 2.
```

```
        nop
        nop
        bsf     Clk2
        goto    Output_Channel_End

Output_Channel_End:
        bsf     Busy_Module     ; Se indica que el resultado fue escrito satisfactoriamente en la salida analógica.

        return                  ; Se retorna al ciclo principal del módulo de expansión.


Read_Module_ID:

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN       ; Se reconfigura el módulo SSP con el pin SDO como salida de datos.
        bcf     TRISC,5
        bsf     SSPSTAT,SSPEN

        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (irrelevante en esta transacción).

        banksel EEADR
        clrf    EEADR           ; El código de identificación del módulo se encuentra en la dirección
                                ; EEPROM 0x00.
        banksel EECON1
        bcf     EECON1,EEPGD
        bsf     EECON1,RD

        banksel EEDATA
        movf    EEDATA,W
        movlw   0x02
        banksel SSPBUF
        movwf   SSPBUF                  ; Se carga el byte de las entradas digitales en el buffer de transmisión.
        call    SPI_Poll ; Se espera a que la cpu central lea el byte de las entradas digitales.

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN       ; Se reconfigura el módulo SSP con el pin SDO como entrada de datos.
        bsf     TRISC,5         ; Con esto se logra liberar el bus para que se realice la próxima transacción entre
        bsf     SSPSTAT,SSPEN       ; la cpu central y cualquier otro módulo de expansión.
```

```
        banksel SSPBUF                  ; Se selecciona el Banco 0 (por estándar).

        retum                   ; Se retoma al ciclo principal del módulo de expansión.


                                ; Fin de la rutina "Test_Command".      Banco 0.

System_Stopped

        nop
        goto    System_Stopped


Slave_Cycle                     ; Rutina cíclica de los módulos de expansión.

        call    SPI_Poll ; El esclavo se mantiene esperando hasta que reciba el primer byte desde la cpu central.
        call    Test_Command    ; Luego de recibir el primer byte, evalúa si éste "le pertenece" y qué tipo de comando
                                ; le fue enviado.

        goto    Slave_Cycle
                                ; Fin de la rutina "Slave_Cycle".


Get_Address                     ; Rutina que asigna dinámicamente una dirección para el módulo de expansión.

        banksel PORTB
        movf    PORTB,W                 ; Se lee el puerto B para cargar los estados en el latch.
        btfss   This_Module_In  ; Se determina si existe algún módulo instalado "detrás" de este módulo de expansión.
        goto    Wait_Serial_Address

        movlw   0x01
        movwf   Module_Address  ; NO hay un módulo instalado después de este, por lo que se adquiere la dirección 0x01.

        incf    Module_Address,W; La dirección actual + 1 será asignada al módulo siguiente.

        movwf   TXREG           ; Se envía la dirección para el módulo siguiente.

        goto    Slave_Cycle

Wait_Serial_Address:
```

```
Poll_Rx                         ; Ciclo que "polea" la bandera de recepción USART y finaliza cuando ya se haya recibido
        banksel PIR1            ; un byte (el byte de dirección asignada al módulo).
      btfss     PIR1,RCIF
      goto      Poll_Rx

        banksel RCREG
        movf    RCREG,W
        movwf   Module_Address  ; Se asigna a este módulo la dirección enviada por el módulo adyacente.

        incf    Module_Address,W; La dirección actual + 1 será asignada al módulo siguiente.

        movwf   TXREG           ; Se envía la dirección para el módulo siguiente.

        goto    Slave_Cycle

                                ; Fin de la rutina "Get_Address".

Slave_SPI_Init                  ; Rutina que inicializa el módulo SPI.

                                ;~~~~~~~~~~~~ Configuración para el Módulo USART ~~~~~~~~~~~

      bsf STATUS,RP0    ; Banco 1
      movlw     .129
      movwf     SPBRG           ; 9600 bps (con BRGH=1).
      movlw     b'00100110'
      movwf     TXSTA           ; Modo asíncrono, 8 bits, alta velocidad.
      movlw     b'00000000'
      movwf     PIE1    ; El sistema no utilizará las interrupciones de los
                ; periféricos.
      bcf STATUS,RP0    ; Banco 0
      movlw     b'10010000'
      movwf     RCSTA           ; Habilitación del puerto serial, 8 bits, continuo.

                                ;~~~~~~~~~~~~ Fin de la Configuración para el Módulo USART ~~~~~~~~~~~

        banksel ADCON0
        bsf     ADCON0,ADCS1 ; Se selecciona Fosc/32 como reloj para el convertidor A/D.

        banksel ADCON1
        clrf    ADCON1                  ; Se utilizarán 6 canales analógicos, con alineación hacia la izquierda.
```

```
        bsf     ADCON1,3        ; Los pines RA3:RA2 serán utilizados como referencias de voltaje 5V y 2.5V
                                ; respectivamente.
        banksel OPTION_REG
        bcf     OPTION_REG,7    ; Se habilitan los pull-ups del puerto B.

        banksel PORTB
        bcf     Busy_Module     ; Al inicio, el módulo de expansión no está listo para realizar una transacción.
        bcf     Next_Module_Out         ; Se pone en bajo la señal de entrada en el módulo siguiente para asignación de direc.
        bsf     Clk0            ; Al inicio, los relojes de los FFs para los DACs se colocan en alto.
        bsf     Clk1
        bsf     Clk2

        movf    SSPBUF,W        ; Se lee el buffer para borrar la bandera de recepción.

        banksel TRISB
        bsf     TRISB,0         ; El pin RB0 será utilizado como entrada para la señal de direccionamiento dinámico.
        bcf     TRISB,1         ; El pin RB1 será utilizado como salida para la señal de direccionamiento dinámico.
        bcf     TRISB,2         ; El pin RB2 será utilizado como salida para la señal "Busy" del módulo de expansión.
        bcf     TRISB,3         ; Los pines RB3,RB4 y RB5 serán utilizados como salidas para las señales de reloj hacia
        bcf     TRISB,4         ; los FFs de los DACs.
        bcf     TRISB,5         ;

        bsf     TRISC,5         ; El pin RC5 será utilizado como salida de datos (al inicio se configura como entrada).

        clrf    TRISD           ; Todo el puerto D será utilizado como puerto de salida para los DACs.

        banksel SSPCON
        bsf     SSPCON,SSPM0; El módulo SSP se configura como Slave sin SS'.
        bsf     SSPCON,SSPM2;
        bsf     SSPCON,SSPEN; Se habilita el módulo SSP.

        bcf     Next_Module_Out         ; Se indica la presencia de este módulo (indicación para el módulo adyacente).

        clrf    Ac0
        clrf    Ac1
        movlw   .3
        movwf   Ac2
.Modules_Init_Wait:             ; Espera de 0.3 segundos antes de entrar al ciclo del Módulo para permitir que éste sea
        decfsz  Ac0             ; señalizado por el módulo adyacente, para determinar si se trata del último módulo.
        goto    Modules_Init_Wait
        decfsz  Ac1
```

```
        goto    Modules_Init_Wait
        decfsz  Ac2
        goto    Modules_Init_Wait

        goto    Get_Address     ; Se adquiere una dirección para el módulo.


                                ; Fin de la rutina "Slave_SPI_Init".

        END
```

```
;*************************************************************************************************************
                        Software de bajo nivel.  Firmware del Módulo de Expansión Digital
;*************************************************************************************************************
LIST P=16F877
    INCLUDE P16F877.INC

    __CONFIG _WDT_OFF & _HS_OSC & _CP_OFF & _LVP_OFF & _PWRTE_ON & _BODEN_ON

      #DEFINE       This_Module_In  PORTA,0
      #DEFINE       Next_Module_Out         PORTA,1


    CBLOCK 0X20

            SPI_Command
            SPI_Byte_In
            Module_Address

    ENDC


      CBLOCK        0X70

            W_temp
            STATUS_Temp
```

```
                FSR_Temp
                INDFx
                Pointer
                Ac0
                AC1
                AC2

        ENDC


    ORG 0
    goto        Slave_SPI_Init

SPI_Poll
        banksel SSPSTAT                 ; El tiempo máximo para que el esclavo salga de esta rutina, luego de haberse recibido
        btfss   SSPSTAT,BF      ; el byte, es de 2.6 uS.
        goto    SPI_Poll
        banksel SSPBUF
        movf    SSPBUF,W        ; Se lee el búfer de transmisión para borrar la bandera de recepción.
        movwf   SPI_Byte_In     ; Se almacena el byte recibido.

        return
                                ; Fin de la rutina "SPI_Poll".                  Banco 0.

Test_Command                    ; Rutina que determina si el primer byte de cada ciclo enviado por la cpu central le
                                ; corresponde al módulo de expansión, de ser así, evalúa el comando recibido, recibe
                                ; el byte siguiente (Byte #2 de), y lee/escribe el Byte #3 según corresponda al caso.
        movf    SPI_Byte_In,W
        andlw   b'11110000'     ; Se extrae el código del comando recibido.
        movwf   SPI_Command     ; Se almacena el código del comando recibido.

        movf    SPI_Byte_In,W
        andlw   b'00001111'     ; Se extrae la dirección del módulo de expansión destino para el Byte #1.
        subwf   Module_Address,W; Se verifica si la transacción debe realizarse con este módulo.
        btfsc   STATUS,Z
        goto    Module_Addressed; Si la dirección es la de este módulo, entonces se reciben los dos bytes siguientes.
        call    SPI_Poll ; Si la dirección no es la de este módulo, entonces se ignoran los dos bytes siguientes.
        call    SPI_Poll

        return                  ; Se retoma al ciclo principal del módulo de expansión.
```

```
Not_Valid_Command:
        call    SPI_Poll
        call    SPI_Poll

        return


Module_Addressed:
        clrf    PCLATH          ; Se utilizará direccionamiento relativo en el primer bloque de 256 bytes.
        bcf     STATUS,C
        rrf     SPI_Command
        rrf     SPI_Command
        rrf     SPI_Command
        rrf     SPI_Command

        movf    SPI_Command,W
        addwf   PCL
        goto    Read_Module_ID
        goto    Read_Digital_Inputs
        goto    Write_Digital_Outputs
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command
        goto    Not_Valid_Command

        return                  ; Si el comando no es válido, simplemente se retorna al ciclo principal del módulo
                                ; esclavo, cuando se cumpla el tiempo Tm, la cpu central notificará el error de E/S.
                                ; (pues la cpu central no recibirá la señal de ACK del módulo, en el pin RC1).
```

```
Read_Digital_Inputs:            ; Nótese que con este comando no se lleva a cabo "handshaking".

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN   ; Se reconfigura el módulo SSP con el pin SDO como salida de datos.
        bcf     TRISC,5
        bsf     SSPSTAT,SSPEN

        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (irrelevante en esta transacción).
        movf    PORTB,W
        movwf   SSPBUF          ; Se carga el byte de las entradas digitales en el buffer de transmisión.
        call    SPI_Poll ; Se espera a que la cpu central lea el byte de las entradas digitales.

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN   ; Se reconfigura el módulo SSP con el pin SDO como entrada de datos.
        bsf     TRISC,5         ; Con esto se logra liberar el bus para que se realice la próxima transacción entre
        bsf     SSPSTAT,SSPEN   ; la cpu central y cualquier otro módulo de expansión.

        banksel SSPBUF          ; Se selecciona el Banco 0 (por estándar).

        return                  ; Se retorna al ciclo principal del módulo de expansión.


Write_Digital_Outputs:          ; Nótese que con este comando no se lleva a cabo "handshaking".

        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (irrelevante en esta transacción).
        call    SPI_Poll ; Se espera a que la cpu central escriba el byte de las salidas digitales.
        movf    SPI_Byte_In,W
        movwf   PORTD           ; Se escribe el byte a las salidas digitales.

        return                  ; Se retorna al ciclo principal del módulo de expansión.


Read_Module_ID:

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN   ; Se reconfigura el módulo SSP con el pin SDO como salida de datos.
        bcf     TRISC,5
        bsf     SSPSTAT,SSPEN
```

```
        call    SPI_Poll ; Se espera a que la cpu central envíe el byte #2 (irrelevante en esta transacción).

        banksel EEADR
        clrf    EEADR           ; El código de identificación del módulo se encuentra en la dirección
                                ; EEPROM 0x00.
        banksel EECON1
        bcf     EECON1,EEPGD
        bsf     EECON1,RD

        banksel EEDATA
        movf    EEDATA,W
        movlw   0x01
        banksel SSPBUF
        movwf   SSPBUF                  ; Se carga el byte de las entradas digitales en el buffer de transmisión.
        call    SPI_Poll ; Se espera a que la cpu central lea el byte de las entradas digitales.

        banksel SSPSTAT
        bcf     SSPSTAT,SSPEN           ; Se reconfigura el módulo SSP con el pin SDO como entrada de datos.
        bsf     TRISC,5         ; Con esto se logra liberar el bus para que se realice la próxima transacción entre
        bsf     SSPSTAT,SSPEN           ; la cpu central y cualquier otro módulo de expansión.

        banksel SSPBUF                  ; Se selecciona el Banco 0 (por estándar).

        return                  ; Se retorna al ciclo principal del módulo de expansión.


                                ; Fin de la rutina "Test_Command".          Banco 0.

System_Stopped

        nop
        goto    System_Stopped


Slave_Cycle                     ; Rutina cíclica de los módulos de expansión.

        call    SPI_Poll ; El esclavo se mantiene esperando hasta que reciba el primer byte desde la cpu central.
        call    Test_Command    ; Luego de recibir el primer byte, evalúa si éste "le pertenece" y qué tipo de comando
                                ; le fue enviado.
```

```
        goto    Slave_Cycle
                                ; Fin de la rutina "Slave_Cycle".

Get_Address                     ; Rutina que asigna dinámicamente una dirección para el módulo de expansión.

        banksel PORTA
        movf    PORTA,W                 ; se lee el puerto A para cargar los estados en el latch.
        btfss   This_Module_In  ; Se determina si existe algún módulo instalado "detrás" de este módulo de expansión.
        goto    Wait_Serial_Address

        movlw   0x01
        movwf   Module_Address  ; NO hay un módulo instalado después de este, por lo que se adquiere la dirección 0x01.

        incf    Module_Address,W; La dirección actual + 1 será asignada al módulo siguiente.

        movwf   TXREG           ; Se envía la dirección para el módulo siguiente.

        goto    Slave_Cycle

Wait_Serial_Address:

Poll_Rx                         ; Ciclo que "polea" la bandera de recepción USART y finaliza cuando ya se haya recibido
        banksel PIR1            ; un byte (el byte de dirección asignada al módulo).
    btfss       PIR1,RCIF
    goto        Poll_Rx

        banksel RCREG
        movf    RCREG,W
        movwf   Module_Address  ; Se asigna a este módulo la dirección enviada por el módulo adyacente.

        incf    Module_Address,W; La dirección actual + 1 será asignada al módulo siguiente.

        movwf   TXREG           ; Se envía la dirección para el módulo siguiente.

        goto    Slave_Cycle

                                ; Fin de la rutina "Get_Address".

Slave_SPI_Init                  ; Rutina que inicializa el módulo SPI.

                                ;~~~~~~~~~~~~ Configuración para el Módulo USART ~~~~~~~~~~~
```

```
bsf STATUS,RP0     ; Banco 1
movlw      .129
movwf      SPBRG           ; 9600 bps (con BRGH=1).
movlw      b'00100110'
movwf      TXSTA           ; Modo asíncrono, 8 bits, alta velocidad.
movlw      b'00000000'
movwf      PIE1    ; El sistema no utilizará las interrupciones de los
           ; periféricos.
bcf STATUS,RP0     ; Banco 0
movlw      b'10010000'
movwf      RCSTA           ; Habilitación del puerto serial, 8 bits, continuo.

                           ;~~~~~~~~~~~~ Fin de la Configuración para el Módulo USART ~~~~~~~~~~~~

    banksel ADCON1
    movlw   b'00000111'
    movwf   ADCON1                  ; No se Utilizarán canales analógicos.

    banksel OPTION_REG
    bcf     OPTION_REG,7  ; Se habilitan los pull-ups del puerto B.

    banksel PORTA
    bcf     Next_Module_Out         ; Se pone en bajo la señal de entrada en el módulo siguiente para asignación de direc.

    movf    SSPBUF,W        ; Se lee el buffer para borrar la bandera de recepción.

    banksel TRISA
    bsf     TRISA,0         ; El pin RA0 será utilizado como entrada para la señal de direccionamiento dinámico.
    bcf     TRISA,1         ; El pin RA1 será utilizado como salida para la señal de direccionamiento dinámico.
    bcf     TRISA,2         ; El pin RA2 será utilizado como salida para la señal "Busy" del módulo de expansión.
    bsf     TRISC,5         ; El pin RC5 será utilizado como salida de datos (al inicio se configura como entrada).

    movlw   0xFF
    movwf   TRISB           ; Todo el puerto B será utilizado como puerto de entradas digitales (8 entradas).

    clrf    TRISD           ; Todo el puerto D será utilizado como puerto de salidas digitales (8 salidas).

    banksel SSPCON
    bsf     SSPCON,SSPM0; El módulo SSP se configura como Slave sin SS'.
    bsf     SSPCON,SSPM2;
```

```
        bsf     SSPCON,SSPEN; Se habilita el módulo SSP.

        bcf     Next_Module_Out         ; Se indica la presencia de este módulo (indicación para el módulo adyacente).

        clrf    Ac0
        clrf    Ac1
        movlw   .3
        movwf   Ac2
Modules_Init_Wait:              ; Espera de 0.3 segundos antes de entrar al ciclo del Módulo para permitir que éste sea
        decfsz  Ac0             ; señalizado por el módulo adyacente, para determinar si se trata del último módulo.
        goto    Modules_Init_Wait
        decfsz  Ac1
        goto    Modules_Init_Wait
        decfsz  Ac2
        goto    Modules_Init_Wait

        goto    Get_Address     ; Se adquiere una dirección para el módulo.


                                ; Fin de la rutina "Slave_SPI_Init".

        END
```

Imprenta "GORA"
25 Av. 25-71, Zona 5
Telefax: 2335-5733 - 5218-7292