UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN

ANTEPROYECTO PARA LA IMPLANTACION DE UN LABORATORIO DE MICROCONTROLADORES PARA LA F.E.S.C. DE LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO.

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA

(ESPECIALIDAD EN ELECTRONICA)

P R E S E N T A :

JOSE SALVADOR VELARDE MENCHACA

ASESOR DE TESIS: ING. JUAN GONZALEZ VEGA

CUAUTITLAN IZCALLI, EDO. DE MEXICO. 1996

TESIS

COMPLETA

UNIVERSIDAD NACIONAL AVENMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

DEPARTAMENTO DE EXAMENES PROFESIONALES

ASUNTO: VOTOS APROBATORIOS

DR. JAIME KELLER TORRES
DIRECTOR DE LA FES-CUAUTITLAN
P R E S E N T E .

AT'N: Ing. Rafael Rodriguez Ceballos
Jefe del Departamento de Exámenes
Profesionales de la F.E.S. - C.

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS TITULADA:

"ANTEPROYECTO PARA LA IMPLANTACIÓN DE UN LABORATORIO DE MICROCONTROLADORES PARA LA F.E.S. CUAUTITLÁN U.N.A.M."

que presenta EL pasante: JOSÉ SALVADOR VELARDE TENCHACA
con número de cuenta: 8962197-4 para obtener el TITULO de:
INGENIERO MECÁNICO ELECTRICISTA .

Considerando que dicha tesis reúne los requisitos necesarios para ser discutida en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E .
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Edo. de Méx., a 11 de ENERO de 199 6

PRESIDENTE	ING. JOSÉ LUIS RIVERA LÓPEZ
VOCAL	ING. JORGE PUENDÍA GÓMEZ
SECRETARIO	ING. JUAN GONZÁLEZ VEGA
PRIMER SUPLENTE	ING. MARGARITA LÓPEZ LÓPEZ
SEGUNDO SUPLENTE	ING. MA. DE LOURDES MALDONADO LÓPEZ

UAE/DEP/VAP/02

J U R A D O:

PRESIDENTE:

ING. JOSE LUIS RIVERA L.

VOCAL:

ING, JORGE BUENDIA

SECRETARIO:

ING. JUAN GONZALES V.

1er SUPLENTE:

ING. MARGARITA LOPEZ L.

2do. SUPLENTE:

ING. Ma. del LOURDES
MALDONADO L.

A ti Madre, que aunque no pudiendo disfrutar de tu presencia, espero que este humilde trabajo sea un agradecimiento por haberme dado el don de la existencia. GRACIAS....

A ti Padre, por los incontables momentos de esfuerzo y lucha que han sido la luz y el modelo a seguir, por todos los momentos en que mi flaqueza estuvo a punto de derrotarme y donde siempre tu mano fuerte y cariñosa nunca me abandonó. Te agradezco infinitamente.

A mis Abuelitos paternos, quienes siempre me han cuidado con cariño inmerecido,con todo amor,y respeto

A mi abuelito materno con un suspiro de agradecimiento por todo lo que me enseñó.

A mis hermanos, para quienes va la exhortación de superar este trabajo.

INDICE:

CAPITULO UNO

CAPITULO I.-

I.1.-INTRODUCCION.

La evolución de la tecnología que produce el microprocesador está marcada por dos grandes ramas.

Una de ellas está representada por los microprocesadores que han evolucionado alargando las longitudes del largo de palabra(i.e. 16,32,64 bits) así como el incremento del poder de procesamiento. La otra rama está marcada por los microprocesadores que han sido combinados con RAM, ROM y varios puertos de entrada y salida en un sólo encapsulado, los cuales son comúnmente llamados MICROCONTROLADORES. Este trabajo versará sobre este tema.

Este trabajo está dirigido a aquéllos quienes desean aprender cómo utilizar el MICROCONTROLADOR en el diseño de un instrumento ó dispositivo.

Para utilizar el MICROCONTROLADOR efectivamente, el diseñador deberá desarrollar al menos tres distintas habilidades. Primeramente, se debe tener el fundamental conocimiento de los componentes disponibles en el MICROCONTROLADOR, es decir; la estructura de los registros del CPU, el conjunto de instrucciones del lenguaje ensamblador que se utilize, sus modos de direccionamiento; y sus recursos en el encapsulado. Así mismo; para que este conocimiento sea realmente efectivo, el diseñador deberá extenderse más allá de los dispositivos simples e idealizados. Por ejemplo; un MICROCONTROLADOR plenamente utilizado puede trabajar varias actividades en tiempo real utilizando interrupciones de control.

Para lograr ésto sin ningún error, el diseñador debe entender cómo el MICROCONTROLADOR manipula las interrupciones y situaciones temporales relacionadas con las actividades de tiempo real.

Como segunda habilidad el diseñador deberá entender los algoritmos requeridos para cada aspecto de un diseño y ser capaz de transladarlos al lenguaje del MICROCONTROLADOR.

Por ejemplo, el diseño del sistema que impide que los frenos de un automóvil se traben; implica un conocimiento amplio tanto del sistema de frenado como de la implementación de su control mediante una secuencia de instrucciones de un MICROCONTROLADOR.

La tercera habilidad consiste en que el diseñador debe entender como los requerimientos extensivos de un instrumento o dispositivo pueden ser repartidos en diferentes partes de un todo.

Este trabajo utiliza el MICROCONTROLADOR de la familia MC51 de Intel.

1.2.-EL PAPEL DEL MICROCONTROLADOR EN LA ACTUALIDAD.

La última década ha observado una evolución vertiginosa en lo que respecta a las capacidades del microprocesador. Los creadores del microprocesador han producido procesadores de 16, 32, 64 bits (los cuáles pueden operar comandos ó instrucciones de 16, 32,64 bits y pueden direccionar megabytes de memoria)para responder a la necesidad cada vez más creciente de unidades de procesamiento central(del inglés cpu-central processing unit).

Estos procesadores han encontrado su nido en el desarrollo de cada vez más poderosos microcomputadores. Se han convertido en la estructura primordial de las estaciones de trabajo; las cuales se han transformado en una herramienta revolucionaria en

Ingeniería, Arquitectura, etc.

Debido a su poder de procesamiento y rapidez, estos microprocesadores de 16 y 32 bits también han encontrado desarrollo dentro del diseño de dispositivos que requieren capacidades sofisticadas de control. Sin embargo, exploraremos otra rama en la evolución de los microprocesadores. Esta rama ha sido desarrollada con igual vigor por los productores de semiconductores. En vez de enfocarse en largos de palabra extensos, el énfasis ha sido dirigido hacia un control extraordinariamente rápida de tiempo real. Centrándose primordialmente en la integración de las facilidades requeridas para soportar un control rápido de eventos en un sólo encapsulado.

En el pasado, las aplicaciones de control de tiempo real han utilizado microprocesadores de 16 y 32 bits junto con encapsulados manejadores de interrupciones, encapsulados temporizadores programables y memoria RAM y ROM para lograr lo que ahora se puede lograr con un sólo MICROCONTROLADOR.

Aún para las situaciones en las que las aplicaciones de control de tiempo real, para las cuales, los componentes disponibles en los MICROCONTROLADORES no son adecuados; tales encapsulados generalmente ofrecen, por mucho; la aproximación al diseño óptimo.

Sus recursos dentro del encapsulado proveen una aproximación integrada a una variedad de tareas rutinarias dentro del control de eventos en tiempo real. Mediante la utilización del encapsulado en su modo de operación expandido, no solamente aprovechamos las particularidades dispuestas en el encapsulado, si no que además; las mejoramos según se acomoden en la aplicación. Todo el poder disponible en cualquier encapsulado periférico está dispuesto para nuestros diseños basados en un MICROCONTROLADOR.

1.3.-APLICACIONES.

Una de las aplicaciones más útiles se han hallado en los localizadores personales debido a la reducción de los receptores de radio. Estas unidades receptoras de mensajes personificadas tienen una longitud de 10 a 15 cm., lo cual le permite ser fácilmente transportadas. Incluye un MICROCONTROLADOR de encapsulado CMOS(semiconductor óxido-metal complementario), el cual extiende la operación de la batería lo más posible. Además el MICROCONTROLADOR se encarga de interpretar los caracteres recibidos con su radio receptor, alerta al usuario con sonidos audibles, y presenta los últimos cinco mensajes recibidos en su desplegador de cristal líquido de bajo consumo de potencia. Cada mensaje puede tener hasta 24 caracteres alfa-numéricos.

Otra aplicación interesante se encuentra en los dispositivos probadores de circuitos integrados, cuyo tamaño es parecido al una calculadora de mano común y corriente; el cual contiene un zócalo el cual acepta encapsulados DIP(dual in line package) de hasta 40 pines. El usuario sólo necesita alinear el pin número 1 a la esquina superior izquierda del zócalo. Los botones selectores ayudan a seleccionar el tipo de encapsulado que se está probando: así como la familia a la que pertenece(e.g. la familia TTL 74LSXXX). Cuando el miembro deseado de la familia es localizado, al presionar la tecla de "test", se aplicará potencia eléctrica y el dispositivo se encargará de realizar las pruebas pertinentes durante un tiempo de una décima de segundo. Si el encapsulado llegara a fallar en la prueba, los pines defectuosos serían indicados, junto con la naturaleza de la falla.

Este probador incluye un modo de funcionamiento automático que se encarga de probar dispositivos desconocidos, pues debido a que puede correr varias pruebas en

milisegundos, este dispositivo puede identificar instantáneamente el componente a prueba así como su apropiado funcionamiento.

1.4.-DISPOSITIVOS PERIFERICOS.

Otra clase de aplicaciones para los MICROCONTROLADORES es representada por la variedad de dispositivos que no trabajan por su cuenta; pero adquieren mayor utilidad cuando se advocan a aumentar las capacidades de un microcomputador. De hecho el teclado de un microcomputador emplea un MICROCONTROLADOR para registrar una tecla oprimida, y luego transmitir de manera serial el código de la tecla oprimida. Esta es un excelente ayuda para descargar de trabajo al CPU. El MICROCONTROLADOR del teclado busca continuamente teclas que estén siendo oprimidas, traduce las teclas en códigos, simplificando el diseño mecánico de los conmutadores de las teclas así como la respectiva interfase del MICROCONTROLADOR.

Si varias teclas son oprimidas al mismo tiempo, el MICROCONTROLADOR manda los códigos al CPU en el orden en que fueron oprimidas, pues el MICROCONTROLADOR puede distinguir pequeños intervalos de tiempo. Además cuando una tecla se queda oprimida, el MICROCONTROLADOR genera una serie de códigos repetitivos lo que hace que el caracter aparezca en el monitor como si fuera la tecla repetidamente oprimida.

En el tiempo de encendido, el MICROCONTROLADOR del teclado lleva a cabo un procedimiento de diagnóstico e informa al sistema si detecta el problema. Mientras este procedimiento no detecte teclas en mal estado, seguirá buscando diferentes tipos de problemas tales como conexiones nulas o deficientes dentro del teclado.

Finalmente, debido a la comunicación serial con la computadora se logra una simplificación del cable conector del teclado

Otra aplicación se encuentra en los MODEMS, que son dispositivos para interconectar diferentes computadoras mediante líneas telefónicas. Su diseño emplea dos MICROCONTROLADORES; uno lo utiliza como interpretador de comandos mientras el otro maneja la transmisión y recepción de datos.

Las impresoras a tinta de color utilizan el MICROCONTROLADOR de Intel 8096 para realizar un conjunto amplio de tareas. La impresora utiliza las entradas del MICROCONTROLADOR, que son de alta velocidad; en conjunción con cuatro bucles servodetectores de posición de papel. Utiliza las salidas del MICROCONTROLADOR para el control de fase de 60 Hz para controlar la exposición de color. Utiliza el puerto serial para expander el número de las líneas de salida. El 8096 incluye un convertidor analógico-digital multiplexado que es utilizado para medir la temperatura mediante termistores.

El 8096 no incluye todos los recursos necesarios para realizar todo el trabajo. Consecuentemente, todos los encapsulados temporizadores son utilizados para generar señales con modulación de ancho de pulso. Este es un método simple para generar una señal de salida con una componente de corriente directa proporcional al número digital. También se cuenta con un UART(transmisor receptor asíncrono universal) que se comunica con un procesador separado que monitorea el panel frontal y genera los comandos de control al 8096. El 8096 es también apoyado por RAM y ROM externos.

Los MICROCONTROLADORES son también usualmente utilizados para implementar subfunciones dentro del diseño de un instrumento. Por ejemplo, considere un osciloscopio de precisión digitalizado. Este instrumento puede capturar 16000 muestras de una señal de

onda, con una resolución de 10 bits para cada muestra y con una razón máxima de muestreo de 20 millones de muestras por segundo.

Este instrumento no solo captura y despliega datos, también puede analizarlos. Para mediciones de pulso, el osciloscopio puede automáticamente computar y analizar tiempos de subida y caída, anchos de pulso y amplitud. Con señales de onda las puede convertir al dominio de frecuencia, así como sacar el análisis respectivo, y además de desplegar espectros de magnitud y fase en el mismo tubo de rayos catódicos en donde está desplegada la señal de onda con dominio en el tiempo.

Este instrumento puede crear una copia permanente del desplegado original y cualquier derivado en un floppy disk. También puede grabar el entero estado de el instrumento para que las secuencias complejas de la configuración del instrumento puedan ser reutilizadas.

Un MICROCONTROLADOR es empleado en el diseño del osciloscopio de precisión digitalizado para manejar las funciones del panel frontal. Puede explorar el teclado para detectar teclas presionadas. Controla los LEDS indicadores intereonstruidos dentro de las teclas para indicar cuáles han sido presionadas. El tubo de rayos catódicos incluye capacidades de entradas mediante el contacto físico con la pantalla del tubo de rayos catódicos, tarea la cual está obviamente controlada por otro MICROCONTROLADOR.

1.5.-APLICACIONES EN LA INDUSTRIA AUTOMOTRIZ.

Las necesidades de la industria automotriz por los MICROCONTROLADORES es tremenda. Una aplicación importante ha sido la creación de módulos de control de la máquina de ignición.

Esta unidad mantiene un control de lazo cerrado para optimizar el ahorro de combustible y

controlar exhaustivas emisiones de contaminante. Emplea sensores para el oxigeno, temperatura del anticongelante y presión barométrica. Se controla además la mezcla de aire y combustible, el tiempo de duración de chispa y la conversión del torque.

Los MICROCONTROLADORES están empezando a servir en otras aplicaciones en automóviles. Un sistema que impide el trabado de los frenos monitorea cada neumático. Si el piloto frena repentinamente con fuerza, el sistema adquiere la máxima fricción del sistema de balatas con un efecto en el cual los frenos presionan y depresionan el disco de la balata muchas veces por segundo. El carro se detiene con una rapidez y con un grado de control que excede grandiosamente el posible control que el piloto pudiera conseguir sin el sistema.

Otra aplicación automotríz es el control dinámico de manejo. Se puede ajustar el sistema de suspensión de un carro chico durante el frenado en curva, de tal forma que se crea el lujo de manejo generalmente asociado con automóviles grandes.

Páneles de instrumentos electrónicos, desplegadores de tubo de rayos catódicos, sistemas de navegación, controladores de transmisiones, sistemas de cableado multiplexado, teléfono celular, sistemas de seguridad, dirección hidráulica controlada electrónicamente, sistemas de evasión de choques; representan algunas otras aplicaciones para automóviles. El automóvil del futuro será necesariamente una maravilla tecnológica.

1.6.-BREVE ESQUEMA HISTORICO.

Aunque las computadoras han estado entre nosotros solo unas cuantas décadas, su impacto han sido profundo, rivalizando seriamente con el teléfono, automóvil, o la televisión. Su presencia es sentida por todos nosotros, si somos programadores de computadoras o

recibimos nuestros recibos de cuentas impresos mediante grandes sistemas de computo. Nuestra noción de las computadoras usualmente las ubica en el papel de procesadores de datos, realizando grandes operaciones con números con gran perfección.

Como un componente en muchos productos industriales y de consumo, como en cajas registradoras y básculas, en las casas en los hornos de microondas, lavadoras de ropa, relojes de alarma, termoestatos, video casseteras, equipos estereofónicos ,copiadoras. En todos estos equipos las computadoras realizan funciones de control mediante el interfasamieto con el mundo real.

Es difícil imaginar el mundo presente de herramientas electrónicas y jugetes sin la ayuda del microprocesador. Sin embargo esta maravilla en un solo chip ha alcanzando escasamente su cumpleños numero veinte. En 1971 Intel introdujo el 8080, el primer microprocesador exitoso. Poco tiempo después Motorola, RCA, Mos Technology y Zilog introdujeron dispositivos similares: 6800, 1801, 6502, y el Z80, respectivamente.

Estos circuitos integrados por si solos eran muy inútiles (y así continuan en la actualidad); pero como parte de un sistema mínimo se volvian el componente central en útiles productos para aprender sobre estos circuitos y el diseño de microprocesadores. Los sistemas mínimos más memorables se encuentran en el D2 de Motorola , el Kim-1 de Mos Technology y el SDK-85 por Intel, y siempre se encontraron en laboratorios de universidades y de diseño.

Un dispositivo similar al microprocesador es el MICROCONTROLADOR. En 1976 INTEL introdujo el 8748, el primer dispositivo en la familia MCS-48 de MICROCONTROLADORES.

Dentro de un simple circuito integrado conteniendo cerca de 17000 transistores, el 8748 tenia un CPU, 1k de memoria EPROM, 64 bytes de RAM, 27 líneas de entrada y de salida y un timer de 8 bits. Este circuito integrado y otros dispositivos de la familia MCS-48 que siguieron, se convirtieron enseguida en un standart industrial en aplicaciones orientadas al control.

El reemplazo de dispositivos electromecánicos se volvió común en dispositivos tales como lavadoras y controladores de trafico.

La capacidad, tamaño y complejidad de MICROCONTROLADORES avanzó en magnitud en 1980 con el anuncio de parte de INTEL de la familia 8051 de MICROCONTRO-LADORES. En comparación con el 8048, estos dispositivos contenían arriba de 60,000 transistores, 4K bytes de ROM, 128 bytes de RAM, 32 lineas de entrada/ salida, un puerto serie y dos timers de 16 bits,- una cantidad apreciable de circuitos para un solo circuito integrado. Nuevos y mejorados dispositivos han sido agregados a la familia MCS-51 y actualmente existen variaciones de virtualmente el doble de estas especificaciones. La Corporación Siemens ofrece como segunda fuente para la familia MCS-51, el SAB80515, una versión mejorada de 8051 en una presentación de 68 pines con seis puertos de 8 bits para entrada y salida, 13 fuentes de interrupción y un convertidor de 8 bits A/D con 8 canales de entrada.

La familia 8051 está bien establecida como una de las más versátiles y poderosas dentro de los MICROCONTROLADORES de 8 bits, su posición como MICROCONTROLADOR lider será más fuerte en los años venideros.

CAPITULO

DOS

DESCRIPCION DEL MICROCONTROLADOR DE LA FAMILIA MCS-51 DE INTEL CORPORATION.

2.1.- Conceptos básicos del microcomputador.-

La mayoría de las computadoras utilizan internamente el sistema de numeración binario (base 2). Todas las variables, constantes y caracteres alfanuméricos son representados por grupos de dígitos binarios llamados bits, cada uno de los cuales tienen sólo valores de 0 ó 1. Las microcomputadoras se clasifican por el número de bits que pueden manejar en un proceso en un momento dado.

Los microcomputadoras de la familia MCS-51 (llamadas también MICROCONTROLADORES de la familia MCS-51) tienen una unidad de proceso central (C.P.U.) de ocho bits.

La mayoría de las operaciones procesan variables con un ancho de 8 bits.

Toda la memoria RAM y ROM interna, si está presente: es de un largo de ocho bits, así como de virtualmente otros componentes dentro del microcontrolador. Una variable de ocho bits (byte) puede asumir uno de los 256 valores distintos, los cuales usualmente representan valores enteros entre 0 y 255.

Otros tipos de números e instrucciones son representados por uno ó más bytes utilizando ciertas convenciones.

Por ejemplo, para representar valores positivos y negativos, el bit más significativo (D7) indica el signo de los otros siete bits; 0 si es positivo, 1 si es negativo permitiendo variables enteras entre -128 y +127. Para los enteros de extremada longitud, varios bytes son manipulados conjuntamente como enteros de precisión múltiple con signo ó sin él, 16, 24 ú más bits de ancho.

Números de dos dígitos decimales pueden ser representados por un grupo de ocho bits, utilizando cuatro bits para cada dígito decimal. Esta representación es llamada Decimal

Codificado en Binario(DCB) y es usualmente utilizada internamente en programas que interactúan intensamente con seres humanos.

Los caracteres alfa-numéricos son usualmente representados utilizando el código ASCII (del inglés -American Standard Code for Information Interchange-Código de Standard Americano para el Intercambio de la Información). Cada caracter es asociado con un número binario de siete bits único. De esta manera, un byte puede representar un caracter, y una palabra, ó una secuencia de letras pueden representarse por una serie o cadena de bytes. Como el código ASCII utiliza solamente 128 caracteres, el bit más significativo de un byte no es necesario para identificar entre caracteres. Usualmente D7 esta configurado en 0 para todos los caracteres. En algunos códigos especiales D7 es utilizado para indicar la paridad de los siete bits restantes; configurando o desconfigurando como sea necesario para asegurar que el número total de bits con valor 1 en el código de ocho bits es par (paridad par) ó impar (paridad impar). El microcontrolador de la familia MCS-51 incluye hardware para computar la paridad cuando sea necesario.

Un programa de computador consiste en una secuencia ordenada de instrucciones específicas y simples a ser ejecutadas una a la vez por el C.P.U. El método o secuencia de pasos utilizados colectivamente para resolver una aplicación de usuario es llamada 'ALGORITMO'.

El programa es colocado dentro de la computadora como una secuencia de números binarios, donde cada número corresponde a cada una de las instrucciones básicas llamadas en inglés "opcodes"; los cuales son entendibles y ejecutables por el C.P.U. En la familia MCS-51 cada localidad de memoria de programa es un byte. Una instrucción completa consiste en una secuencia de uno ó más bytes, en donde el primero define la operación a ser ejecutada y los bytes adicionales (si son necesarios) contienen información adicional, tal

como valores de datos ó direcciones de variables. Ninguna instrucción tiene más de tres bytes.

La manera en que los códigos binarios llamados "opcodes" y los bytes modificadores son asignados a las operaciones del C.P.U. es denominado lenguaje máquina. El escribir un programa directamente en lenguaje máquina puede convertirse en una tarea excesiva y tediosa. Los seres humanos piensan en palabras y conceptos más que en números codificados, de tal suerte que a cada operación y recurso del C.P.U. le está dado ó asignado un nombre y una abreviación standard denominada nemónico. Así los programas son más fácilmente entendibles y discutibles utilizando estos nemónicos standard ó usualmente denominado en su conjunto lenguaje ensamblador. Existen programas que pueden traducir el programa desde el lenguaje ensamblador al código de lenguaje máquina. Para la familia del microcontrolador que nos ocupa; tal lenguaje se denomina Ensamblador MCS-51 ó ASM51. Existen algunas importantes diferencias entre el lenguaje máquina del computador y el lenguaje ensamblador utilizado como herramienta para representarlo. El lenguaje máquina ó conjunto de instrucciones es el conjunto de operaciones que el C.P.U. puede realizar mientras el programa se está ejecutando y está estrictamente determinado por el diseño del hardware de la computadora.

El lenguaje ensamblador es un conjunto standard más ó menos arbitrario de símbolos incluyendo el conjunto de instrucciones nemotécnicas, pero con características adicionales que simplifican substancialmente el proceso de diseño. Por ejemplo. ASM51 tiene controles para crear y formatear un listado de programa, y un número de directivas para la localización de variables y la inserción de bytes arbitrarios de datos dentro del código objeto para la creación de tablas de constantes.

Adicionalmente. ASM51 puede realizar sofisticadas operaciones matemáticas, computar

direcciones ó evaluar expresiones aritméticas para ayudar al programador en estas tareas. Sin embargo, estos cálculos deben utilizar unicamente información conocida en el momento de realizar el ensamblado del programa.

Por ejemplo, el 8031 realiza cálculos aritméticos en tiempo de corrida de programa, con ocho bits a la vez. ASM51 puede realizar similares operaciones con 16 bits a la vez. El 8031 puede realizar un paso a la vez por instrucción, mientras que el ASM51 puede realizar complejos cálculos en cada linea de código fuente. Sin embargo, las operaciones realizadas por el ensamblador deben utilizar valores de parámetros determinados en el tiempo de ensamblado, pues ninguna variable tiene valores desconocidos hasta que la ejecución del programa empieza. Por ejemplo, cuando el lenguaje ensamblador tiene la siguiente linea en su código fuente:

```
ADD   A,#(LOOP_COUNT + 1)* 3
```

El ASM51 encontrará el valor de la constante previamente definida "LOOP_COUNT" en una tabla interna de constantes, incrementará el valor, multiplicará la suma por 3, y (asumiendo que está entre -256 y 255 inclusive)truncará el producto a ocho bits. Cuando esta instrucción es ejecutada, la ALU del 8051 solamente sumará la constante resultante a el acumulador.

Algunas diferencias existen para distinguir números de diferente bases. El 8031 realiza todas las computaciones en sistema binario (aunque se tiene previsto que se realizan las conversiones al sistema decimal). En el curso de la escritura de un programa, seguramente; podría ser conveniente especificar constantes utilizando alguna otra base de numeración tal como la de base 16. En otras ocasiones, puede ser deseable especificar el código ASCII para algunos caracteres o cadena de éllos sin referirse a la tabla. El ASM51 permite

diferentes representaciones para constantes, las cuales son convertidas al sistema binario mientras cada instrucción es ensamblada.

Por ejemplo, los números binarios son representados en lenguaje ensamblador por una serie de unos y ceros (naturalmente), seguidos por la letra "B"(de binario); los números octales como una serie de números de ocho dígitos(0-7) seguidos por la letra O(de octal). Los números hexadecimales son representados por un serie de dígitos hexadecimales(0-9,A-F), seguidos por la letra "H". Un número hexadecimal debe comenzar con un dígito decimal; de otra manera parecería como un símbolo definido por el usuario. Un cero antepuesto puede ser útil para identificar un número. La cadena de caracteres "BACH" puede ser una etiqueta legal para una rutina de música barroca sintética; pero la cadena de caracteres "0BACH" es la constante hexadecimal "BAC". Este es un caso donde la inserción del 0 hace la gran diferencia.

Los números decimales son representados por una secuencia de dígitos decimales, opcionalmente seguidos por la letra "D".

Cuando un código ASCII es necesario en un programa, el deseado caracter deberá ser encerrado por dos apóstrofes(como en '#') y el lenguaje ensamblador lo convertirá en el apropiado y correspondiente código (en este caso 23H). Una cadena de caracteres entre apóstrofes es traducido como una serie de constantes; es decir, 'BACH' se convierte en 42H,41H,43H,48H.

2.2.-ARQUITECTURA Y ORGANIZACION.-

La figura 1 describe el microcontrolador mediante la organización de bloques. Cada microcomputadora combina un C.P.U., con dos tipos de memoria(RAM,ROM ó EPROM), puertos de entrada y de salida, el status de modo de funcionamiento, los registros de datos y la lógica necesaria para una variedad de funciones

periféricas.

Estos elementos se comunican através de canales de ocho bits de ancho que corre por todo el encapsulado. como si fuera una tubería. Este canal ó bus se comunica al exterior mediante puertos de entrada y salida cuando se desea memoria ó se desean expansiones de entrada y de salida.

Vamos a realizar una sinopsis de cada bloque que se muestra en la figura 1.

2.3.- UNIDAD CENTRAL DE PROCESAMIENTO (C.P.U.).-

El C.P.U. es el cerebro de la microcomputadora, leyendo el programa de usuario y ejecutando las instrucciones contenidas en el mismo. Sus elementos primarios son una unidad lógico-aritmética de ocho bits asociada con los registros A, B, PSW y SP, el contador de programa de 16 bits y los registros de apuntador de datos.

2.4.-UNIDAD ARITMETICO LOGICA.

Esta unidad puede realizar (como el nombre implica) funciones aritméticas y lógicas en variables de ocho bits. Esto es: suma, substracción, multiplicación y división básica; así como las operaciones lógicas tales como: Intersección(AND), Conjunción(OR) y conjunción exclusiva(OR exclusiva), así como la rotación, complemento y borrado de bits. La unidad aritmético-lógica realiza decisiones sobre salto condicional en subrutinas, provee rutas de datos y ofrece registros temporales utilizados para la transferencia de datos dentro del sistema. Otras instrucciones son construidas a partir de estas operaciones primitivas, la capacidad de sumar puede incrementar registros o computar automáticamente direcciones destino del programa; la substracción es utilizada en el decremento o comparación de las magnitudes de dos variables.

FIGURA 1

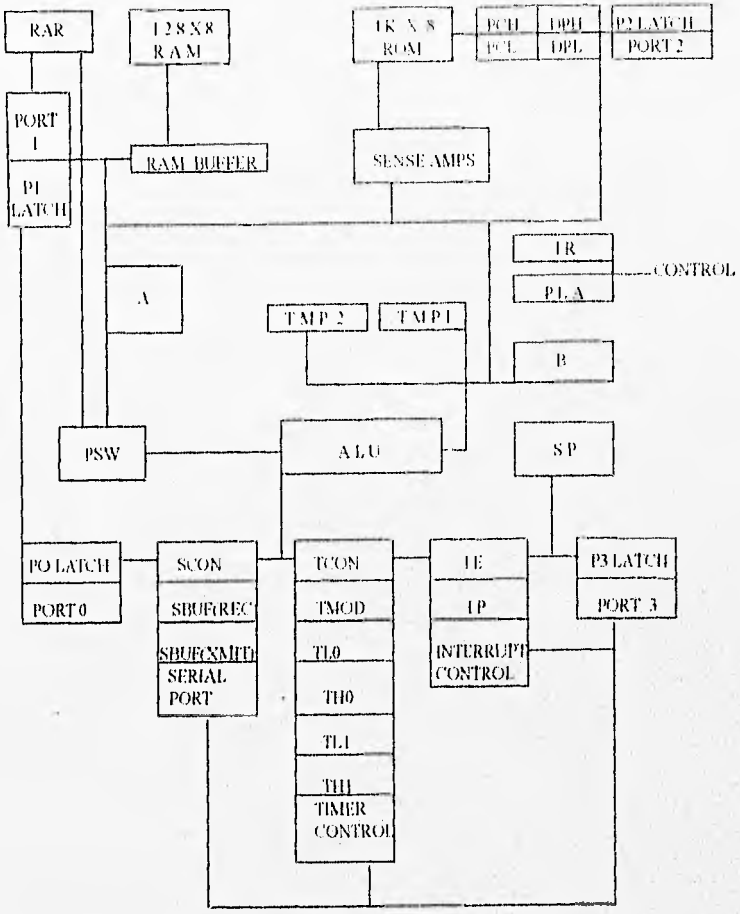


DIAGRAMA DE BLOQUES DE 8031
(ESTRUCTURA INTERNA)

Estas operaciones primitivas son automaticamente colocadas en orden descendente y combinadas con la lógica dedicada a construir instrucciones complejas tales como el incrementar un registro de 16 bits. Para ejecutar una forma de la instrucción de comparación, por ejemplo, el 8051 incrementa el contador del programa tres veces, lee tres bytes de la memoria de programa, computando una dirección de un registro con operaciones lógicas, lee datos de memoria interna dos veces, hace la comparación aritmética de dos variables, computa la dirección destino de 16 bits y decide realizar o no un salto condicional: todo lo anterior en dos microsegundos.

Una importante y única característica de la arquitectura de MCS-51 es que la unidad aritmético-lógica puede manipular datos de un solo bit así como de ocho bits, los bits individuales pueden ser configurados, borrados o complementados, movidos, probados y utilizados en computaciones binarias. Mientras el soporte de datos más primitivo puede parecer inicialmente un paso atrás en una era de incremento de longitud de largo de palabra, éste hace al 8051 especialmente adaptable para aplicaciones del tipo de control. Tales algoritmos implican inherentemente variables de entrada y salida de tipo Boleano, las cuales eran consecuentemente más difíciles de implementar en otros microprocesadores.

Gracias a esta poderosa unidad aritmético-lógica, el conjunto de instrucciones del 8051 soporta suficientemente bien el control de tiempo real y algoritmos con intensas cantidades de datos. Un total de 51 operaciones separadas mueven y manipulan tres tipos de datos; a saber: Boleanos(falso y verdadero),Byte(8 bits), y dirección(16 bits).Así como también existen 11 modos de direccionamiento: siete para datos, cuatro para control de secuencia de programa (sin embargo, solo ocho son utilizadas por más de unas cuantas instrucciones especializadas). La mayoría de las instrucciones permiten varios modos de direccionamiento, trayendo un total de instrucciones de 111.

Sinópsis del conjunto de instrucciones.

Estas 111 instrucciones son clasificadas dentro de cinco grupos:

1.-Operaciones aritméticas

2.-Operaciones lógicas para variables de tipo byte.

3.-Instrucciones para la transferencia de datos

4.-Manipulación de variables tipo booleanas

5.-Saltos de programa y control de máquina.

El conjunto de instrucciones está diseñado para realizar programas eficientes tanto en el tamaño del programa código como en la rapidez de la ejecución. Ninguna instrucción requiere más de tres bytes de memoria de programa, pues la mayoría requiere de uno ó dos bytes. Virtualmente todas las instrucciones se ejecutan en uno ó dos ciclos de instrucción, lo cual se realiza en uno ó dos microsegundos con un cristal de 12 MHZ, con sus excepciones: las cuales las representan la multiplicación y la división; que requieren hasta de cuatro ciclos de instrucción.

Muchas instrucciones tales como funciones aritméticas y lógicas así como control de programa, proveen una forma larga y una corta para la misma instrucción, permitiendo al programador optimizar el programa código producido para una aplicación específica. Por ejemplo, cualquier byte de RAM puede ser cargado con una instrucción de tres bytes de largo, con dos ciclos de instrucción, pero los comúnmente usados registros de trabajo en la RAM pueden ser inicializados en un ciclo con una forma de dos bytes. Cualquier bit dondequiera en el encapsulado puede ser configurado, borrado, ó complementado por una instrucción lógica simple de tres bytes utilizando dos ciclos. Pero bits críticos de control, pines de entrada y salida, y banderas de software pueden ser controladas por instrucciones de dos bytes y ciclo simple. Mientras los saltos de tres bytes y llamadas de subrutina pueden

ir a donde quiera en la memoria de programa. secciones cercanas al código serán alcanzadas por versiones cortas relativas o versiones absolutas.

Un beneficio colateral significativo de un conjunto de instrucciones más poderoso que los de aquéllos de previos encapsulados de microcomputadoras es que es más fácil generar software orientado hacia aplicaciones. Modos generalizados de direccionamiento para instrucciones de byte y bit reducen el número de líneas en el código fuente escritas y compiladas para una aplicación dada. Esto conduce a un costo proporcionalmente bajo en software, gran confianza y rápidos ciclos de diseño.

2.5.-ACUMULADOR Y PSW.-

El 8051 y su predecesor 8048 están basados primariamente en una arquitectura basada en el acumulador; un registro de ocho bits: llamado el acumulador ('A') mantiene un operando fuente y recibe el resultado de las instrucciones aritméticas (adición, substracción, multiplicación y división). El acumulador puede ser la fuente o el destino de operaciones lógicas y de un número de instrucciones especiales de movimiento de datos, incluyendo expansiones de RAM externa y tablas de datos constantes. Varias funciones se aplican exclusivamente a el acumulador; tales como: rotación, computación de paridad, prueba de cero etc.

Muchas instrucciones afectan implícita ó explícitamente muchas banderas de status, las cuales están agrupadas para formar la palabra de status del programa (PSW-Program Status Word)fig. 2. El bit de status más activo es llamado bandera acarreo (abreviada 'C'). Este bit permite múltiples operaciones aritméticas de precisión incluyendo adición, substracción y rotación. El acarreo sirve como acumulador boleano para operaciones de un solo bit así como manipulaciones de bit. La bandera de sobreflujo(OV) detecta cuando se presenta un sobreflujo aritmético que ocurre en operaciones con enteros con signo, haciendo dos

complementos aritméticos posibles. La bandera de paridad (P) es actualizada después de cada ciclo de instrucción con la paridad par de los contenidos del acumulador.

El C.P.U. no controla dos registros llamados bancos de registro, RS1 y RS0. Mejor aún, están controladas vía software para poder habilitar uno de los cuatro bancos de registros.

Aún cuando la arquitectura está basada en el acumulador, se han hecho previsiones para evitar el acumulador en situaciones de instrucciones comunes. Los datos pueden ser movidos desde cualquier localidad del encapsulado a cualquier registro, dirección ó dirección indirecta, cualquier registro puede ser cargado con cualquier constante etc. sin que todo ésto afecte al acumulador. Las operaciones lógicas pueden ser realizadas entre registros ó variables para alterar campos de bits, sin usar o afectar el acumulador. Las variables pueden ser incrementadas, decrementadas ó probadas sin la ayuda del acumulador. Banderas y bits de control pueden ser manipuladas y probados sin afectar nada más.

2.6.-Otros registros del C.P.U.

Un registro especial de ocho bits denominado ('B') sirve en la ejecución de las instrucciones de multiplicación y división. Este registro es utilizado en conjunción con el acumulador como el que contiene el segundo operando, y regresa el resultado de ocho bits.

El registro llamado Stack pointer (apuntador de pila)(SP)es un registro apuntador de ocho bits que indica la dirección del último byte introducido dentro de la pila. El apuntador de pila es automáticamente incrementado o decrementado en todas las instruciones PUSH o POP y todas las llamadas de subrutinas y regresos. En teoria, El apuntador de pila en el 8051 puede tener una profundidad de hasta de 128 bytes. El apuntador de pila toma el valor default de 7 cuando hay reset de tal forma que el apuntador de pila empezará desde la localidad número 8.

FIGURA 2.

ORGANIZACION DE LA PALABRA DE STATUS DE PROGRAMA

(msb) (lsb)

CY AC FO RS1 RS0 OV - P

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
CY	PSW.7	Bandera de acarreo. Puesta/borrada por medio de hardware o software durante ciertas instrucciones aritmeticas
AC	PSW.6	Bandera auxiliar de acarreo Puesta/borrada por el hardware durante instrucciones de adición o substracción para indicar sobre acarreo o sobre préstamo del bit 3.
FO	PSW.5	Bandera 0 Puesta/borrada/probada por el software como una bandera de status defina por el usuario.
RS1	PSW.4	Bit de control de selección del registro de bancos 1 & 0.
RS0	PSW.3	Puesta/borrada por software para determinar el registro de banco en trabajo(ver nota)
OV	PSW.2	Bandera de sobreflujo. Puesta/borrada por hardware durante instrucciones aritméticas para indicar condiciones de sobreflujo.
-	PSW.1	RESERVADO
P	PSW.0	Bandera de paridad Puesta/borrada por hardware cada ciclo de instrucción para indicar status de paridad de los unos localizados dentro del acumulador.

NOTA.- Los contenidos de RS1, RS0 capacita los bancos de trabajo como sigue:

(0,0)-Banco 0 (00H-07H)
(0,1)-Banco 1 (08H-0FH)
(1,0)-Banco 2 (10H-17H)
(1,1)-Banco 3 (18H-1FH)

Mediante la alteración de los contenidos del apuntador de pila puede ser relocalizado en cualquier lugar dentro de la RAM interna.

Finalmente, un registro de 16 bits llamado apuntador de datos (DPTR) sirve como un registro base en saltos indirectos, instrucciones referentes a tablas de constantes y transferencias externas de datos. La mitad superior e inferior de este registro pueden ser manipulados y llamados de forma separada DPH y DPL respectivamente, ó bien: pueden ser utilizados juntos, usando instrucciones especiales para cargar o descargar todos los 16 bits.

2.7.-ESPACIOS DE MEMORIA.

La memoria de programa está separada y es distinta de la memoria de datos. Cada tipo de memoria tiene un mecanismo diferente de direccionamiento, diferentes señales de control y una diferente función.

El arreglo de memoria de programa (ROM y EPROM), tal como un elefante; nunca olvida información aún sin energía eléctrica aplicada. La memoria de programa es usada para proporcionar información necesaria en el tiempo en que la energía eléctrica es suministrada; valores de inicialización, constantes de calibración, tablas de constantes etc., así como el programa por sí mismo. La memoria de programa tiene un bus de 16 bits; sus elementos son direccionados utilizando el contador de programa o por instrucciones que generan direcciones de 16 bits.

En cambio la memoria de datos es como un ratón; si se permite la analogía, pequeño y más rápido que la memoria de programa, y además entra en estado aleatorio en cuanto se aplica energía eléctrica. La memoria RAM dentro del 8031 es usada por variables que están determinadas o podrían cambiar mientras el programa está corriendo.

Una computadora gasta la mayor parte de su tiempo manipulando variables, no constantes y un relativamente pequeño número de ellas. Como ocho bits son suficientes para direccionar

las 128 localidades de RAM interna, las direcciones de los registros de RAM interna son de un byte de ancho. En contraste con la memoria de programa, la memoria de datos solo requiere un valor simple de 8 bits; para especificar una localidad. Como este es el ancho de la ALU y de los diferentes tipos de memoria, esos recursos pueden ser utilizados por los mecanismos de direccionamientos contribuyendo grandemente para la eficiencia operativa de la computadora.

La partición de memoria de programa y de memoria de datos es extendida a la expansión de memoria fuera del encapsulado. Cada una puede ser añadida independientemente, y cada una utiliza la misma dirección y utiliza el mismo bus de datos, pero con diferentes señales de control. La memoria de programa externa es comunicada a el bus externo de datos mediante el control de salida PSEN(Program store enable), en el pin 29 del microcontrolador. La memoria de datos externa es leída mediante el bus de datos utilizando la salida RD, pin 17, y la escritura de datos provistos por el microcomputador mediante la salida WR, pin 16. Mientras que ambos tipos de memoria pueden ser expandidos a más de 64K bytes, la memoria de datos externa puede ser opcionalmente expandido en páginas de 256 bytes para preservar el uso de P2 como un puerto de entrada y salida. Esto es útil con una relativamente pequeña expansion de RAM (tal como la 8155 de Intel) para direccionar periféricos externos.

En adición a los arreglos de memoria descrita existe aún otro espacio de direcciones. Conectados con el bus de datos internos existen una variedad de registros especiales distribuidos en el encapsulado. Algunos de éstos: B,SP,PSW, DPH y DPL han sido anteriormente discutidos. Otros puertos de salida y/o entrada y otros registros de funciones periféricas, serán introducidas en secciones venideras.

Colectivamente estos registros son designados como los registros de funciones especiales de espacio de direccionamiento.

Así, la arquitectura del MCS-51 soporta varios y distintos espacios de direccionamiento, funcionalmente separados al nivel de hardware mediante diferentes mecanismos de direccionamiento, señales de lectura y escritura o ambos:

*Memoria de programa dentro del encapsulado

*Memoria de datos dentro del encapsulado

*Memoria de programa fuera del encapsulado

*Memoria de datos fuera del encapsulado.

*Registros de funciones especiales dentro del encapsulado.

2.8.-PUERTOS DE SALIDA Y ENTRADA.

Las estructura de puertos de entrada y de salida de la familia MCS-51 es extremadamente versátil. El 8031 y el 8751 tienen cada uno 32 pines de entrada y/o salida configurados en cuatro puertos paralelos de ocho bits(P0,P1,P2 y P3). Cada pin podrá sacar o introducir datos bajo el control de software.

En varios modos de operación ó expansión, algunos de estos pines pueden ser utilizados para funciones especiales de entrada y/o salida. Las instrucciones que permiten el acceso a memoria externa utilizan el puerto P0 como un bus multiplexado de datos y direcciones; al principio de un ciclo de memoria externa ocho bits de la dirección son sacados por P0; más tarde los datos entran en los mismos ocho pines. Las instrucciones de transferencia externa de datos que proveen una dirección de 16 bits, y cualquier instrucción que esté accesando memoria de programa externa, saca ocho bits de alto orden por P2 durante el ciclo de acceso. El 8031 siempre usa los pines del puerto P0 y de P2 para direccionamiento externo, pero P1 y P3 están siempre disponibles para puertos de salida y/o entrada.

Los ocho pines del puerto P3 tiene cada uno una función especial. Dos interrupciones externas, dos salidas contadoras, dos líneas de datos seriales y dos controladores temporales estroboscópicos, utilizan los pines del puerto P3, como se describe en la figura 3. Los pines del puerto P3 que son utilizados en funciones especiales pueden usarse para funciones comunes de entrada y salida.

Aún dentro de un puerto simple, las funciones de entrada y salida pueden ser combinadas en diferentes maneras: la salida y entrada pueden ser realizadas utilizando diferentes pines al mismo tiempo, o los mismos pines en diferente tiempo; en paralelo en algunos casos, y en serial en otros; como pines de prueba, o como en el caso del puerto P3 como funciones especiales adicionales.

2.9.-FUNCIONES PERIFERICAS ESPECIALES.

Existen algunas necesidades especiales comunes entre los sistemas computacionales orientados al control, a saber:

* Mantener muestreo del tiempo real.
* Mantener conocimiento de las transiciones de señal
* Medir con precisión el ancho de pulsos de entrada.
* Mantener comunicación con otros sistemas o personas.
* Monitorear cercanamente eventos externos asíncronos.

Hasta ahora, los sistemas basados en microcomputadores necesitaban encapsulados periféricos tales como contadores /temporizadores, USART's para poder manejar estas necesidades. El 8031 integra a todas estas capacidades en el encapsulado.

2.10.-TEMPORIZADORES/CONTADORES.

Existen dos Temporizadores/Contadores de modo múltiple de 16 bits en el 8031, cada uno

consistente de un alto byte y un bajo byte. Estos registros son llamados, TH0, TL0, TH1 y TL1. Cada par puede ser independientemente programado vía software en cualquiera de la docenas de opciones de modos con un registro de modo designado TMOD, figura 4, y controlado por el registro TCON, figura 5.

Los modos de temporizadores pueden ser utilizados para medir tiempos de intervalos, medir anchos de pulso, o iniciar eventos, con una resolución de un microsegundo hasta un intervalo máximo de 65536 ciclos de instrucción (alrededor de 65 milisegundos). Retrasos mayores pueden ser fácilmente acumulados mediante software. Configurado como contador, el mismo hardware acumulará eventos externos a frecuencias desde d.c. hasta 500 khz, con una presición de 16 bits.

2.11.-INTERFASE PUERTO SERIAL.

Cada microcomputadora contiene un puerto serial de gran rapidez que es programable vía software para funcionar en cuatro modos básicos: expansor de registro cambiables de entrada y salida, UART de 8 bits, UART de 9 bits, ó canal interprocesador de comunicaciones. Los modos UART interfasearán dispositivos comunes de entrada y salida (e.g. TRC, teletipos, MODEMS etc.) cuyas razones serán de 122 baudios a 31 kilobaudios. Reemplazando el cristal standard de 12 MHZ con uno de 10.7 MHZ permite una razón de 110 baudios. Cada tipo de paridades, si se desea; pueden ser incluidas con rutinas de software de un bit. Las comunicaciones en sistemas distribuidos toman lugar a una razón de 187 kilobaudios donde se incluye el hardware que realiza automáticamente mensajes de reconocimiento de dirección y datos. Registros cambiadores simples tanto TTL como CMOS proveen expansiones de entrada y salida a bajo costo a una razón de 1 Megabaudio. Los modos de operación del puerto serial son controlados por los contenidos del registro SCON, figura 6.

FIGURA 3.

FUNCIONES ALTERNAS ESPECIALES DEL PUERTO 3

(msb) (lsb)
RD WR T1 T0 INT1 INT0 TXD RXD

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
RD	P3.7	Salida de control lectura de datos. Activo con pulso bajo generado por hardware cuando se lee memoria de datos externa.
WR	P3.6	Salida de control de escritura de datos. Activo con pulso bajo generado por el hardware cuando se escriben datos a memoria de datos externa.
T1	P3.5	Salida externa de temporizador contador 1 ó prueba de pin
T0	P3.4	Salida externa de temporizador/contador 0 ó prueba de pin.
INT1	P3.3	Pin de entrada de interrupción 1. Disparado en bajo nivel o en caida de señal.
INT0	P3.2	Pin de entrada de interrupción 0. Disparada en bajo nivel ó caida de señal.
TXD	P3.1	Pin de transmisión de datos para el puerto serie en modo UART. Salida de reloj en modo de registro cambiado.
RXD	P3.0	Pin de recepción de datos para puerto serie en modo Uart. Pin de entrada/salida modo de registro cambiado.

FIGURA 4.

REGISTRO DE MODO TEMPORIZADOR CONTADOR

(msb) (lsb)
GATE C/T M1 M0 GATE C/T M1 M0

| TIMER 1 | | TIMER 0 |

GATE Control de compuerta.Cuando ésta se configura, el temporizador/contador x es habilitado sólo mientras el pin INTx está en nivel alto el control de bit TRx está habilitado.Cuando está en cero,el temporizador/contador es habilitado cuando el bit de control TRx está puesto en uno.

C/T Selector de temporizador o contador.Puesto en cero para operación del temporizador(entrada desde el reloj interno de sistema). Puesto en cero para la operación de contador(entrada desde el pin de entrada Tx.

M1	M0	MODO OPERATIVO
0	0	Temporizador de MCS-48. TLx sirve como pre-escala de cinco bits
0	1	Temporizador/contador de 16 bits. THx y TLX estan en cascada;no hay pre-escala.
1	0	Temporizador/contador autorecargable de 8 bits. THx mantiene un valor que va a ser recargado dentro de TLx cada vez que exista sobrellujo .
1	1	(Temporizador 0) TL0 es un temporizador contador de ocho bits controlado por el bit estandarde de control Timer 0
1	1	(Temporizador 1) Temporizador/contador 1 parado.

FIGURA 5.

TCON- REGISTRO CONTROL STATUS DE TEMPORIZADOR CONTADOR.

(msb) (lsb)
TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
TF1	TCON.7	Bandera de sobreflujo de temporizador 1. Configurado por hardware en sobreflujo de temporizador/contador. Puesto a cero cuando se procesa la interrupcion.
TR1	TCON.6	Bit de control de corrida de temporizador 1. Pusto a uno o cero por software para encender/apagar el temporizador/contador.
TF0	TCON.5	Bandera de sobreflujo de temporizador 0. Configurado por hardware en el sobreflujo de temporizador/contador. Puesto a cero cuando se procesa una interrupción.
TR0	TCON.4	Bit de control de corrida de temporizador 0. Puesto a cero o uno por software para prender apagar temporizador.
IE1	TCON.3	Bandera de borde de interrupción 1. Puesta a uno por hardware cuando el borde de interrupción externa es detectada. Puesta a cero cuando se procesa una interrupción.
IT1	TCON.2	Bit de control de interrupción 1. Puesto en cero o uno para especificar bajo nivel del borde de caída disparado por interrupciones externas.
IE0	TCON.1	Bandera de borde de interrupción 0 Puesta en uno cuando se detecta borde de interrupción externas.
IT0	TCON.0	Bit de control de interrupción 0. Puesta a cero por software para especificar nivel bajo de borde de caída disparada por interrupciones externas.

2.12.-CAPACIDADES DE INTERRUPCION Y CONTROL.-

Las capacidades de interrupción son consideradas habitualmente como funciones normales de un C.P.U. Está siendo introducida desde el punto de vista de aplicaciones, que las interrupciones estén más cerca de los periféricos y a las interfases del sistema.

Estas funciones periféricas permiten que hardware especial monitoree la interfase de señales de tiempo real, sin molestar al C.P.U. Por ejemplo, imagine datos seriales que están arribando desde un TRC mientras se está transmitiendo datos a otro, y un temporizador/contador está manipulando entradas transitorias de alta velocidad, mientras que otro mide los anchos de pulso de las entradas. Durante todo ésto el C.P.U. está procesando alguna otra situación.

Pero cómo es que el C.P.U. se entera que la recepción, transmisión o pulso terminado. El programador del 8031 tiene tres opciones.

Los registros TCON y SCON contiene bits de status configurados por el hardware cuando un temporizador tiene sobreflujo o cuando la operación en puerto serial está completada. La primera técnica lee el registro de control dentro del acumulador, prueba el bit apropiado, y realiza un salto condicional basado en el resultado.

La segunda técnica consiste, en que el 8031 puede realizar un salto condicional basado en el estado de cualquier bit de control o status o la entrada en un pin en una sola instrucción, una secuencia de cuatro instrucciones escrutan los cuatro sucesos simultáneos mencionados arriba en exactamente 8 microsegundos.

Desafortunamente, el C.P.U. debe aún dejar lo que está haciendo para probar estos bits. Por ejemplo, un gerente no puede realizar su trabajo bien si continuamente está monitoreando el trabajo de sus subordinados: éllos solo deben interrumpir a su gerente sólo cuando necesiten

FIGURA 6.

REGISTRO DE CONTROL/STATUS DEL PUERTO SERIE.

(msb) (lsb)
SM0 SM1 SM2 REN TB8 RB8 TI RI

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
SM0	SCON.7	Bit de control 0 de modo de puerto serie. Puesto a cero o uno por software(ver nota)
SM1	SCON.6	Bit de control 1 de modo de puerto serie. Puesto a cero o uno por software(ver nota)
SM2	SCON.5	Bit de control 2 de modo de puerto serie. Configurado por software para deshabilitar la recepción de estructuras para las cuales el bit 8 es cero.
REN	SCON.4	Bit de control habilitador de receptor. Puesto a cero o uno por software para habilitar/deshabilitar recepción serial de datos.
TB8	SCON.3	Bit 8 de transmisión. Puesto a cero o uno por hardwae para determinar el estado del noveno bit transmitido en modo UART de 9 bits.
RB8	SCON.2	Bit 8 recepción. Puesto a uno o cero por hardware para indicar el estado del noveno bit de datos recibido.
TI	SCON.1	Bandera de interrupción de transmisión. Puesta a uno por hardware cuando un byte es recibido. Puesta a cero por software después del servicio.
RI	SCON.0	Bandera de interrupción de recepción. Puesta a uno por hardware cuando un byte es recibido. Puesto a cero por software después del servicio.

NOTA.- El estado de (SM0,SM1) selecciona:

(0,0)-Cambio de registro para expansión de entrada/salida.
(0,1)-Uart de 8 bits,razón de datos variable
(1,0)-Uart de 9 bits,razón de datos adecuados
(1,1)-Uart de 9 bits,razón de datos variables.

cuidado y consejo. De la misma manera se realiza en un microcomputador; idealmente el C.P.U. no tendría que preocuparse por los periféricos hasta que éstos requieran atención. En ese momento, pospondrá la tarea que estaba realizando, lo suficiente para manipular el dispositivo apropiado, luego regresará al punto donde dejó la tarea descrita.

Esta es la base de la tercera técnica que es la generalmente la más óptima: la interrupción de hardware. El 8031 tiene cinco fuentes de interrupciones: una desde el puerto serie cuando una transmisión/recepción es completada, una desde los temporizadores cuando ocurre el sobreflujo y dos desde los pines de entrada INT0 e INT1.

Cada fuente puede ser independientemente capacitada o descapacitada para permitir la prueba de algunas fuentes en algún momento dado, y cada una puede ser clasificada con una prioridad alta o baja. Una fuente de prioridad alta puede interrumpir rutinas de servicios de baja prioridad. Estas opciones son seleccionadas por la habilitación de interrupción y los registros de control prioritario, IE e IP, figuras 7 y 8.

Cada fuente tiene una dirección particular en la memoria de programa asociada con ella, (tabla 1), empezando en 0003H y continúa en intervalos de ocho bits. Cuando sucede un evento habilitado por interrupciones, el C.P.U. automáticamente ejecuta una llamada a subrutina interna a la correspondiente dirección. Una subrutina de usuario empezando en esta localidad o habiendo saltado a ella puede entonces realizar las instrucciones para servir esa fuente particular. Antes de completar las rutinas de servicio de interrupción, la ejecución retorna a la tarea anterior.

FIGURA 7.

REGISTRO HABILITADOR DE INTERRUPCION IE.-

(msb) (lsb)
EA - - ES ET1 EX1 ET0 EX0

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
EA	IE.7	Bit de control para habilitar todo.Puesto a cero por software para deshabilitar todas las interrupciones, independientemente de el estado de IE.4-IE.0
-	IE.6	reservado
-	IE.5	reservado
ES	IE.4	Bit de control para habilitar el puerto serie.Puesto a cero o uno por software para habilitar deshabilitar interrupciones desde las banderas TI o RI.
ET1	IE.3	Bit de control habilitador del temporizador 1. Puesto a cero o uno por software para habilitar o deshabilitar las interrupciones desde el temporizador/contador 1.
EX1	IE.2	Bit de control habilitador de interrupción externa 1.Puesta en cero o uno por software para habilitar deshabilitar interupciones desde INT1.
ET0	IE.1	Bit de control habilitador del temporizador 0. Puesto a cero o uno por software para habilitar deshabilitar interrupciones desde temporizador/contador 0
EX0	IE.0	Bit de control habilitador de interrupción 0. Puesto a cero o uno por software para habilitar deshabilitar interrupciones desde INT10.

FIGURA 8.

REGISTRO DE CONTROL DE PRIORIDAD DE

INTERRUPCION.-

(msb) (lsb)
-- -- -- PS PT1 PX1 PT0 PX0

SIMBOLO	POSICION	NOMBRE Y SIGNIFICADO
--	IP.7	reservado
--	IP.6	reservado
--	IP.5	reservado
PS	IP.4	Bit de control de prioridad de puerto serie.Puesto a cero o uno por software para especificar prioridad alta o baja de interrupciones para el puerto serie.
PT1	IP.3	Bit de control de prioridad del temporizador 1.Puesta en cero o uno por software para especificar prioridad alta o baja de interrupciones para el temporizador/contador 1.
PX1	IP.2	Bit de control de prioridad de interrupción externa 1.Puesta a cero o uno para especificar interrupciones de alta o baja prioridad desde INT1.
PT0	IP.1	Bit de control de prioridad de temporizador 0. Puesta a cero o uno por software para especificar alta o baja prioridad de interrupciones para temporizador/contador 0.
PX0	IP.0	Bit de control de prioridad de interrupción externa 0. Puesta en cero o uno por software para especificar alta o baja prioridad para interrupciones para INT0.

TABLA I.

FUENTES DE INTERRUPCION 8031 Y VECTORES DE SERVICIO

FUENTE DE INTERRUPCION	DIRECCIONES DECOMIENZO DE LAS RUTINAS DE SERVICIOS
RESET	000H
EXTERNO 0	003H
TEMPORIZADOR/CONTADOR 0	00BH
EXTERNO 1	013H
TEMPORIZADOR/CONTADOR 1	01BH
PUERTO SERIE	023H

2.13.-CONJUNTO DE INSTRUCCIONES Y MODOS DE DIRECCIONAMIENTO.

El conjunto de instruciones del 8031 es extremadamente regular, en el sentido de que la mayoria de las instrucciones pueden operar con variables desde varios y diferentes espacios de direcciones fisicos ó lógicos . Antes de indagar profundamente en el conjunto de instrucciones, es importante enteder los detalles más comunes de modos de direccionamiento.

2.14.-MODO DE DIRECCIONAMIENTO DE DATOS.

El ASM51 consiste de un operador nemonico y de cero a tres operandos separados por comas. En instrucciones de dos operandos, el destino es especificado primeramente, luego la fuente. Muchas operaciones de datos de byte, tales como ADD o MOV inherentemente utilizan el acumulador como una fuente de operando y o para recibir el resultado. Por ejemplo, la instrucción:

ADD A, FUENTE

sumará la variable "fuente" al acumulador, dejando el resultado en el acumulador.

El operando designado como "fuente" puede usar cualquiera de los cuatro comunes modos de direccionamiento:

* Registro- uno de los registros de trabajo en el banco habilitado.

* Directo-una localidad de RAM interna, puerto de entrada y de salida, o registros de funciones especiales.

* Registro indirecto-una localidad de RAM interna, apuntada a un registro de trabajo.

* Datos inmediatos-una constante de ocho bits incorporada en la instrucción.

Los tres primeros modos proveen acceso a la RAM interna y a los espacios de direccionamieto de los registros de hardware, y puede asi mismo ser utilizada como un operando fuente o destino. El último modo accesa memoria de programa y puede ser un

operando fuente.

2.15.-REGISTROS DE DIRECCIONAMIENTO

El programador del 8031 tiene acceso a ocho registros de trabajo, numerados como R0-R7. Los últimos tres bits menos significativos de la instrucción de 'opcode' indica un registro dentro del espacio lógico de la dirección. Así, un código de función y una dirección de operando puede ser combinada para formar una instrucción de un byte. fig 9a.

El ASM51 indica direccionamiento de registro utilizando el símbolo Rn(donde n toma valores de 0 a 7) o con un nombre simbólico previamente definido como un registro por las directivas EQU'ate o SET.

Ejemplo 1-Sumando dos registros

```
;REGARD SUMA LOS CONTENIDOS DEL REGISTRO
;1 A LOS CONTENIDOS DEL REGISTRO 0
REGARD:   MOV   A,R0
          ADD   A,R1
          MOV   R0,A
```

Existen cuatro de los tales registros de trabajo, donde sólo uno de los cuales está activo a la vez. Físicamente, éllos ocupan los primeros 32 bytes de RAM del encapsulado. (Direcciones 0-1FH). Los bits de PSW 4 y 3 determinan que banco está activo. Un reseteo de hardware habilita el banco de registro); para seleccionar un banco diferente, el programador debe modificar los bits 4 y 3 del PSW adecuadamente.

Ejemplo 2-Selección de bancos de memoria alternos.

```
MOV  PSW, #00010000B   ,SELECCIONA BANCO 2
```

2.16.-DIRECCIONAMIENTO DIRECTO DE BYTE.

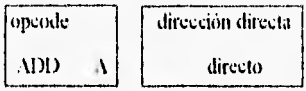
El direccionamiento directo puede accesar cualquier variable dentro del encapsulado o registro de hardware. Un byte adicional especificado en el "opcode" especifica la localidad a ser utilizado(fig 9b).

FIGURA 9.

FORMATOS DE CODIGO MAQUINA PARA DIRECCIONAMIENTO DE DATOS.

a)Direcionamiento de registro

b)Direccionamiento directo



c)Direccionamiento indirecto de registro

d)Direccionamiento inmediato

Dependiendo en el bit de orden más alto del byte de direccionamiento directo, uno de dos espacios de memoria física es seleccionada. Cuando la dirección directa está entre 0 y 127(00H-7FH) uno de los 128 localidades de bajo orden en la RAM de encapsulado es utilizada.

Ejemplo 3-Sumando los contenidos de localidades RAM.

```
;DIRADR SUMA LOS CONTENIDOS DE LA LOCALIDAD
;RAM 41H A LOS CONTENIDOS DE LA LOCALIDAD
;40H
;
DIRADR:   MOV   A,40H
          ADD   A,41H
          MOV   40H,A
```

Todos los puertos de entrada/salida y funciones especiales, los registros de control y status tienen asignado direcciones entre 128 y 255(80H-0FFH). Cuando el byte de dirección directa está entre estos límites el registro correspondiente de hardware está accesado. Por ejemplo, los puertos 0 y 1 tienen asignados direcciones directas tales como 80H y 90H, respectivamente. Una lista completa está presentada en la tabla 2. (pág. 41)

Ejemplo 4.-Sumando datos de entrada del puerto a datos de salida de puerto.

```
;PRTADR SUMA DATOS DE ENTRADA EN EL PUERTO 1
;A LOS DATOS PREVIAMENTE SACADO EN EL PUERTO
;0
;
PRTADR:   MOV     A,P0
          ADD     A,P1
          MOV     P0,A
```

El direccionamiento directo permite a todos los registros de funciones especiales en el 8031 el ser leídos, escritos o utilizados como operandos de instrucciones. En general, este es el único método utilizado para accesar puertos de entrada salida y los registros de funciones especiales. Si el direccionamiento directo es usado con direcciones de registros de funciones especiales más que otros listados anteriormente, el resultado de la operación es indefinido.

2.17.-DIRECCIONAMIENTO INDIRECTO DE REGISTROS.

¿Cómo es posible manejar variables cuyas localidades en RAM están siendo determinadas, computadas o modificadas mientras el programa está corriendo?. Esta situación se presenta cuando se manipulan localidades secuenciales de memoria, entradas indexadas dentro de tablas de RAM, y operaciones de presición múltiple o de cadenas. Registro ó Direccionamiento Directos no pueden ser usados, mientras sus direcciones de operandos estén fijados en el tiempo de ensamblado.

La solución se presenta mediante el direccionamiento de RAM indirecto de registros. R0 y R1 de cada uno de los bancos de registros, pueden operar como indexadores o registros de apuntadores, sus contenidos indican una dirección en RAM. La localidad de RAM interna así direccionada es el actual operando en uso. El bit menos significativo de la instrucción del "opcode" determina cual registro es utilizado como apuntador (figura 9c).

En el ASM51, el direccionamiento indirecto de registros está representada por el signo @ precediendo R0 y R1 o por un símbolo definido por el usuario para ser igual a R0 ó R1.

Ejemplo 5.- Direccionamiento indirecto.

```
;INDADR SUMA LOS CONTENIDOS DE LOCALIDAD
;DE MEMORIA DIRECCIONADAS POR EL REGISTRO
```

```
;A LOS CONTENIDOS DE LOCALIDAD RAM
;DIRECCIONADA POR EL REGISTRO 0
INDADR:     MOV   A,@R0
            ADD   A,@R1
            MOV   @R0,A
```

2.18.-DIRECCIONAMIENTO INMEDIATO.

Cuando un operando fuente es una constante mas que una variable(i.e.-la instrucción usa un valor conocido en el tiempo de ensamblado, entonces la constante puede ser incorporada dentro de la instrucción. Una instrucción de byte adicional especifica el valor usado.(figura 9d).

El valor usado es fijado al tiempo de la manufactura de la ROM o de la programación de la EPROM y no podrá ser alterada durante el tiempo de ejecución. En el lenguaje ensamblador operandos inmediatos están precedidos por el simbolo #. El operando puede ser tanto una cadena numérica, una variable simbólica o un expresión aritmética utilizando constantes.

Ejemplo 6.-Sumando constantes utilizando direccionamiento inmediato.

```
;IMMADR SUMA LA CONSTANTE DECIMAL 12
;A LA CONSTANTE DECIMAL 34, DEJANDO EL
;RESULTADO EN EL ACUMULADOR
IMMADR:     MOV A,#(12+34)
```

2.19.-COMBINACIONES DE MODO DE DIRECCIONAMIENTO.

Los ejemplos anteriores demostraron el uso de los cuatro modos de direccionamiento de datos en instrucciones de dos operandos tales como MOV y ADD, las cuales utilizaron el acumulador como uno de los operandos. Las operaciones ADDC,SUBB,ANL,ORL y

XRL(todas serán adecuadamente discutidas más tarde) pueden ser substitutos de ADD en cada ejemplo. Los primeros tres modos pueden ser tambien para la operación XCH o. en combinación con el modo inmediato de direccionamiento(y un byte adicional),cargado con una constante. Las instrucciones de un solo operando tales como:INC, DEC, DJNZ y CJNE pueden todas operar sobre el acumulador, o pueden especificar los modos de direccionamiento de registro, directo e indirecto. La única excepción se registra con la instrucción DJNZ que no puede usar el acumulador o direccionamiento indirecto.

Las operaciones PUSH y POP no pueden inherentemente direccionar el acumulador como un registro especial. Sin embargo, los tres pueden direccionar directamente el acumulador como uno de los veinte registros de función especial mediante la colocación del símbolo "ACC" en el campo del operando.

2.20.-VENTAJAS DEL DIRECCIONAMIENTO SIMBOLICO.

Como la mayoría de los lenguajes ensambladores o de alto nivel, el ASM51 permite dar nombres apropiados definidos por el usuario. Esto está hecho para lineas de instrucción mediante la colocación de una etiqueta seguida por ":"antes de la instrucción, como en los ejemplos anteriores. Tales símbolos deben empezar con un caracter alfabético y pueden incluir cualquier combinación de letras, números signos interrogación etc. Para nombres muy largos, sólo los primeros 31 caracteres son relevantes.

El ASM51 permite que todas las variables (registros, puertos, direcciones de RAM tanto interna como externa le sean asignadas etiquetas de acuerdo con estas reglas con las directivas de EQUate y SET.

Ejemplo 7.- Direccionamiento simbólico de variables definidas como localidades de RAM.

```
VAR_0   SET 20H
VAR_1   SET 21H
```

```
;SYMB_1 SUMA LOS CONTENIDOS DE VAR 1
;A LOS CONTENIDOS DE VAR 0;
SYMB_1:     MOV  A,VAR_0
            ADD  A,VAR_1
            MOV  VAR_0,A
```

Debe aclararse qute el conjunto de instrucciones del ASM51 tiene pocos nemotécnicos para ser recordados por el programador. Diferentes tipos de datos o modos de direccionamiento son determinados por los operandos especificados, más qte por las variaciones en los nemónicos.

Por ejemplo, el nemónico "MOV" es utilizado por 18 diferentes instrucciones para operar en tres tipos de datos (bit, byte y dirección). Las quince versiones que mueven las variables de byte entre espacios de direcciones lógicas son diagramadas en la figura 10. Cada flecha muestra la dirección de transferencia desde la fuente hacia el destino.

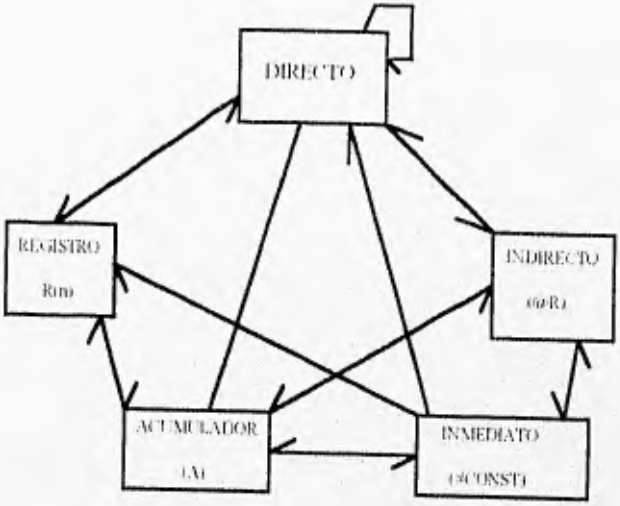
Es de notar, así mismo, que para la mayoría de instrucciones que permiten direccionamiento de registro existe una correspondiente instrucción de direccionamiento directo y viceversa. Esto permite al programador empezar a escribir programas de 8031 como si él ó ella tuviera acceso a 128 registros diferentes. Cuando el programa ha evolucionado lo suficiente hasta el punto donde el programador tiene una idea bastante precisa de qué tan seguido se utiliza una variable, él ó ella podrán localizar los registros de trabajo en cada banco para las variables más utilizadas. Si se utiliza direccionamiento simbólico en la escritura de los programas fuentes, solo las líneas que contienen la definición de símbolos serán necesarias cambiar; el ensamblador entonces producirá las instrucciones apropiadas aunque aún el resto del programa es dejado intacto. Editando solamente las dos primeras líneas del ejemplo 7 reducirán el código producido de 6 bytes a sólo la mitad.

TABLA 2.

DIRECCIONAMIENTO DIRECTO DE REGISTROS DE HARDWARE

REGISTRO	DIRECCION	FUNCION
P0	80H	PUERTO 0
SP	81H	STACK POINTER
		(APUNTADOR DE PILA)
DPL	82H	DATA POINTER (LOW)
		(APUNTADOR DE DATOS BAJO)
DPH	83H	DATA POINTER (HIGH)
		(APUNTADOR DE DATOS ALTOS)
TCON	88H	REGISTRO TEMPORIZADOR
TMOD	89H	REGISTRO DE MODO TEMPORIZADOR
TL0	8AH	TIMER 0 LOW BYTE
		(TEMPORIZADOR 0 BYTE BAJO)
TL1	8BH	TIMER 1 LOW BYTE
		(TEMPORIZADOR 1 DE BYTE BAJO)
TH0	8CH	TIMER 0 HIGH BYTE
		(TEMPORIZADOR 0 ALTO BYTE)
TH1	8DH	TIMER 1 HIGH BYTE
		(TEMPORIZADOR 1 ALTO BYTE)
P1	90H	PUERTO 1
SCON	98H	REGISTRO DE CONTROL DE PUERTO
		SERIE
SBUF	99H	BUFFER DE DATOS DEL PUERTO
		SERIE
P2	0A0H	PUERTO 2
IE	0A8H	REGISTRO HABILITADOR DE
		INTERRUPCION
P3	0B0H	PUERTO 3
IP	0B8H	REGISTRO PRIORIDAD DE
		INTERRUPCION
PSW	0D0H	PALABRA DE STATUS DE PROGRAMA
ACC	0E0H	ACUMULADOR (DIRECCION DIRECTA)
B	0F0H	REGISTRO B

FIGURA 10.



MAPA DE CAMINOS PARA MOVIMIENTO DE BYTES DE DATOS

2.2.1.-UTILIZACION DE LAS FUNCIONES ARITMETICAS-ADD,ADDC,SUBB y DA.

La instrucción ADD suma una variable de byte con el acumulador, dejando el resultado en el acumulador. La bandera de acarreo es puesta en 1 si llegara a existir un sobreflujo desde el bit 7 y será puesta en cero de lo contrario. La bandera AC'V estará en 1 para el sobre acarreo desde el bit 3 para su uso por la instrucción DA descrita posteriormente. La instrucción ADDC' suma los contenidos previos de la bandera de acarreo con dos variables de byte, de otra forma; se comporta igual que ADD.

La instrucción SUBB (substracción con préstamo) substrae una variable de byte indicada y los contenidos de la bandera de acarreo desde el acumulador, y pone los resultados devuelta en el acumulador. La bandera de acarreo sirve como un préstamo durante las operaciones de substracción; cuando un valor grande es substraído de uno pequeño (como restar 5 de 1) el requerimiento de un préstamo dentro del bit de mayor orden, la bandera de acarreo es puesta a uno, de otra forma es puesta a cero.

Cuando se realizan operaciones aritméticas con números binarios con signo, ciertas combinaciones de variables de entrada pueden producir resultados que parecen violar las leyes de las Matemáticas. Por ejemplo, la suma de 7FH(127) consigo mismo produce un resultado de 0FEH, el cual es la representación del complemento a -2. En aritmética normal, dos valores positivos no pueden producir un número negativo cuando se realiza su suma. Similarmente, es comúnmente imposible substraer un número positivo de un negativo y resultar un número positivo, pero en el complemento a dos existen instancias donde ésto puede ocurrir también. Fundamentalmente éstas anomalías ocurren cuando la magnitud de el valor de resultado es muy grande para encajar dentro de los 7 bits permitidos; por éllo no hay representación de un byte en complemento a dos para el número decimal par 254, resultado normal de la soma 127+127.

El procesador del MCS-51 detecta si estas situaciones ocurren e indican tales errores con la bandera OV(sobreflujo). La bandera OV puede ser probada con instrucciones de salto condicional tales como JB y JNB.

A nivel de hardware, la bandera OV es puesta a uno si existe un sobre acarreo de 6 bits, pero no fuera del 7 bit, o un sobre acarreo de 7 bits pero no fuera del 6 bit.

Cuando se suman enteros con signo ésto indica un número negativo producido como la suma de dos operandos positivos, o una suma positiva de dos operandos negativos, en SUBB ésto indica un resultado negativo despúes de substraer un número negativo de un número positivo, o un resultado positivo cuando un número positivo es substraído de un número negativo.

Las instrucciones ADDC y SUBB incorporan los estados previos de la bandera de acarreo para permitir cálculos de precisión múltiples por la repetición de la operación con sucesivos operandos de bytes de alto orden. En cualquier caso, el acarreo debe ser borrado antes de la primera iteración.

Si el dato de entrada para una operación de precisión múltiple es una cadena de enteros sin signo, la bandera de acarreo será puesta a cero si hay un sobreflujo(para ADDC) o un bajo flujo (para SUBB). Con datos con signo en complemento a dos (i.e. el bit más significativo del dato de entrada original indica el signo de la cadena), la bandera de sobre flujo será puesta a uno si el sobreflujo o bajo flujo ocurren.

Ejemplo 8.- Substracción de cadena con detección de sobreflujo con signo.

```
;SUBSTR SUBSTRAE LA CADENA INDICADA POR
;R1 A LA INDICADA POR R0 A LA PRESICION
;INDICADA POR R2.
;CHECAR SI EXISTE SOBREFLUJO CON SIGNO
;
```

```
SUBSTR:    CLR  C    ;PRESTAMO 0
SUBST1:    MOV  A, @R0
           SUBB A,@R1  ;RESTA EL SIG.
           MOV  @R0,A
           INC  R0     ;DESPLAZAR APUNTADORES
           INC  R1
           DJNZ R2, SUBST1  ;SALTO SI ES
                            ;NECESARIO,UNA VEZ
                            ;HECHO,PRUEBA
                            ;SI OCURRE SOBREFLUJO
                            ;EN LA ULTIMA
                            ;ITERACION DE LAZO
           JNB  OV,OV_OK

                            ;RUTINA DE RECOBRO DE
                            ;SOBREFLUJO.
OV_OK:     RET              ;REGRESO
```

La suma decimal es posible mediante la utilización de la instrucción DA en conjunción con ADD y/o ADDC. El valor binario de ocho bits en el acumulador resultado de la adición anterior de dos variables (cada uno puesto forma de BCD de cuatro bits cada uno). Si el contenido de los bits 0-3 del acumulador son mayores de 9 o si la bandera AC había sido configurada: seis son añadidos al acumulador produciendo el apropiado dígito BCD en el nibble de bajo orden. Si la bandera de acarreo es puesta a uno o si los cuatro bits de alto

orden exceden el valor de nueve. estos bits son incrementados a seis. La bandera de acarreo es dejada en uno si originalmente lo estaba o si la adicion de seis produce un acarreo fuera del bit de más alto orden. indicando que la suma de las variables BCD originales es mayor o igual que 100.

Ejemplo 9.-Suma de dos bytes decimales con registros y constantes.

```
;BCDADD SUMA LA CONSTANTE 1.234(DECIMAL) A LOS
;CONTENIDOS DEL PAR DE REGISTROS R3 - R2

BCDADD:   MOV  A,R2
          ADD  A,#34H
          DA   A
          MOV  R2,A
          MOV  A,R3
          ADDC A,#12H
          DA   A
          MOV  R3,A
          RET
```

2.22.-MULTIPLICACION Y DIVISION.

La instrucción "MUL AB" multiplica los valores enteros sin signo de ocho bits mantenidas en el acumulador y el registro B. El byte de bajo orden de un producto de 16 bits es dejado en el acumulador , el byte de orden alto en B.Si los ocho bits de alto orden son todos ceros. la bandera de sobreflujo es puesta en cero, de otra manera se pondrá en uno. El programador puede escrutar la bandera OV para determinar cuando el registro B no es cero y debe ser

procesada.

"DIV AB" divide enteros sin signo de ocho bits localizadas en el acumulador y en el registro B. La parte entera del cociente es puesta en el acumulador, y el residuo es puesto en el registro B. Si el registro B originalmente contenia 00H entonces la bandera de sobreflujo será configurada de tal forma que indicara un error de división, y los valores de resultado serán indefinidos.De otra forma la bandera OV es puesto en cero.

La instrucción de división es también útil para propositos tales como conversión de números de distinta base o para separar campos de bits en el acumulador.

Una subrutina corta puede convertir un entero binario de ocho bits sin signo localizada en el acumulador (entre 0 & 255 a una representación BCD de tres dígitos. El dígito de centenas es almacenado en un registro (HUND) y las centenas y las unidades son almacenadas en código BCD en otro registro(TENONE).

Ejemplo 10.- Usar la instrucción DIV para la conversión de base.

```
;BINBCD CONVIERTE UNA VARIABLE BINARIA DE OCHO
;BITS EN ACC A UN FORMATO DE TRES DIGITOS
;EN CODIGO BCD. LAS CENTENAS ESTARAN EN HUND,
;LAS DECENAS Y UNIDADES EN TENONE
;
        HUND  EQU  21H
        TENONE EQU  22H
;
BINBCD: MOV B,#100      ;DIVIDIR POR 100 PARA
        DIV AB          ;DETERMINAR EL NUM. DE
                        ;CENTENAS
        MOV HUND,A
        MOV A,#10       ;DIVIDIR REMANENTE POR 10
```

```
        XCH A,B         ;PARA DETERMINAR # DE DECENAS
        DIV AB          ;DIGITOS DE DECENAS EN ACC.
                        ;REMANENTES SON DIGITOS DE
                        ;UNIDADES
        SWAP A
        ADD A,B         ;PONER LOS DIGITOS BCD EN ACC
        MOV TENONE,A
        RET
```

La instrucción de división pueden también separar ocho bits de datos en el acumulador en subcampos. Por ejemplo, datos en códigos BCD pueden ser separados en dos nibbles por la división del dato por 16, dejando el nibble superior en el acumulador y el nibble de bajo orden en B. Los dos dígitos pueden ser entonces operados individualmente o en conjunción con el otro. El siguiente ejemplo recibe dos dígitos en código BCD en el acumulador y regreso el producto de dos dígitos individuales en formato BCD en el acumulador.

Ejemplo 11.-Usando la multiplicación en BCD utilizando MPY y DIV.

```
;MULLBCD DESEMPACA DOS DIGITOS EN FORMATO BCD RECIBIDOS
;EN EL ACC, ENCUENTRA SU PRODUCTO, Y REGRESA EL PRODUCTO EN
;FORMATO BCD EN EL ACC.
;
MULBCD:  MOV  B,#10H    ;DIVIDIR LA ENTRADA ENTRE 16
         DIV  AB        ;A&B TIENEN DIGITOS SEPARADOS
                        ;CADA UNO JUSTIFICADO A LA IZQ.
                        ;EN REGISTROS
         MUL  AB        ;A MANTIENE EL PRODUCTO EN
```

```
                        ;FORMATO BINARIO (0-63H)
MOV  B,#10              ;DIVIDIR EL PRODUCTO ENTRE 10
DIV  AB                 ;A MANTIENE # DE DECENAS,B
                        ;MANTIENE EL RESIDUO.
SWAP A
ORL  A,B                ;EMPACAR REGISTROS
RET
```

2.23.-OPERACIONES LOGICAS CON BYTES-ANL,ORL,XRL.

Las instrucciones ANL,ORL Y XRL realizan las operaciones lógicas de AND,OR y OR exclusivo en las dos variables de byte indicadas, dejando el resultado en el primero. Ninguna bandera es afectada.

Estas operaciones pueden usar los mismos modos de direccionamiento como las operaciones aritméticas pero de manera distinta, pues ya que no es necesario operar en el acumulador. Los bytes directamente direccionados pueden ser utilizados como el destino con el uso de tanto el acumulador o una constante como la fuente. Estas instrucciones pueden ser útiles en operaciones tales como ANL,ORL,y XRL, borrado, configurado y complemento, en uno o más bits de RAM, puertos de salida, o registros de control. El patrón de bits a ser afectados es indicado por un byte máscara disponible. Se utiliza el direccionamiento inmediato cuando el patrón a ser afectado es conocido en el tiempo de ensamblado (figura 11); usar las versiones de acumulador cuando el patrón es computado en el tiempo de corrida.

Los puertos de entrada y salida son usualmente usados para datos paralelos en formatos diferentes a los de los ocho bits. Por ejemplo, los cinco bits de bajo orden del puerto 1 pueden sacar un caracter alfabético sin molestar los bits 5-7.

FIGURA 11 Y 12.

FIGURA 11.- PATRON DE INSTRUCCION PARA OPERACIONES LOGICAS EN MODO ESPECIAL DE DIRECCIONAMIENTO.

OPCODE	DIRECCION DIRECTA	MASCARA
ANL	PI	DATOS

FIGURA 12.- FORMATOS DEL CODIGO MAQUINA PARA INSTRUCCIONES DE SALTO

a).-Salto largo (LJMP dir. 16):

OPCODE	DIR15-DIR8	DIR7-DIR0

b).-Salto absoluto (AJMP dir 11):

DIR10-DIR8	OPCODE	DIR7-DIR0

c).-Salto corto (SMJP rel):

OPCODE	OFFSET RELATIVO

Este puede ser un paso, un proceso simple de dos pasos. Primero, borrar los cinco pines de bajo orden con una instruccion ANL, después configurará esos pines correspondiendo a aquellos en el acumulador. (Este ejemplo asume que los tres bits de más alto orden del acumulador son originalmente cero).

Ejemplo 12.-Reconfigurando el tamaño del puerto con instrucciones de byte lógicas.

```
OUT_PX: ANL  P1,#11100000B ;BORRA LOS BITS P1.4-P1.0
        ORL P1,A           ;CONFIGURA PINES P1 CORRES-
                           ;PONDIENTES PARA CONFIGURAR
                           ;LOS BITS DEL ACC.
        RET.
```

En este ejemplo, los bits de bajo orden permanecen en alto nivel pueden parpadear hacia el nivel bajo para un ciclo máquina. Si no es deseable, se puede utilizar una aproximación ligeramente diferente. Primeramente, poner todos los pines correspondientes a los unos del acumulador, entonces se borran los pines correspondientes a los ceros en los bits de bajo orden del acumulador. No todos los bits cambiarán desde su estado original al estado final al mismo instante, pero los bits no hacen ninguna transición intermedia.

Ejemplo 13.-Reconfiguración del tamaño de puerto de entrada y salida sin el parpadeo.

```
ALT_PX:  ORL  P1,A
         ORL  A,#11100000B
         ANL
         RET
```

2.24.-CONTROL DE PROGRAMA: SALTOS, LLAMADAS Y REGRESOS.

El microcontrolador de la familia 8031 tiene tres instrucciones de salto. Cada una provoca que la ejecución del programa salte incondicionalmente a alguna otra dirección. Cada una difiere en cómo el código de máquina representa las direcciones de destino.

La instrucción LJMP(salto largo) codifica una dirección de 16 bits en el segundo y tercer byte de instrucción(figura 12a); el destino puede ser cualquier localidad en el espacio de memoria de programa de 64Kilobytes.

La instrucción de dos bytes AJMP(salto absoluto) codifica su destino direccionando los bits de 10 a 8 formando un campo de tres bits en el opcode y direcciona los bits de 7 a 0 formando el segundo byte(figura 12b). Los bits de direcciones 15-12 están sin cambios desde los contenidos del P.C.(PROGRAM COUNTER); así la instrucción AJMP solo puede ser usada cuando el destino es conocido y se sabe que está dentro del mismo bloque de 2K. De otra manera el ASM51 marcará error.

Un tipo diferente de instrucción de dos bytes para salto es legal con cualquier dirección cercana, sin importar las fronteras de los bloques de memoria o páginas. La instrucción SJMO(salto corto) codifica el destino con un programa de dirección contadora relativa en el segundo byte(figura 12c). El C.P.U. calcula el destino en el tiempo de corrida por la adición de un valor de desplazamiento de ocho bits con signo al P.C.(PROGRAM COUNTER) incrementado. Valores negativos de offset causarán saltos arriba de 128 bytes hacia atrás; valores positivos arriba de 127 bytes hacia adelante.

(SJMP con 00H en el offset del código máquina procederá con la instrucción siguiente).

En el esfuerzo de mantener un mínimo de instrucciones nemotécnicas, existe una forma genérica para identificar las tres instrucciones de salto. El ASM51 reconoce el nemónico JMP como una pseudo instrucción transladándola a instrucciones máquina LJMP,AJMP ó

SJMP, dependiendo en la dirección de destino.

Como SJMP, todos los saltos condicionales utilizan el direccionamieto relativo. JZ(salto si cero) y JNZ(salto si no cero) monitorean el estado de el acumulador como se implica por su nombre, mientras JC(salto en acarreo) y JNC(salto en no acarreo) prueban si las banderas de acarreo están puestas. Todas son instrucciones de dos bytes, con el mismo formato como en la figura 12e. JB(salto en bit),JNB(salto en no bit) y JCB(salto en bit y luego borra el bit) pueden probar cualquier bit de status o cualquier pin de entrada con una instrucción de tres bytes: el segundo byte especifica cual bit está bajo prueba y el tercero dá el valor relativo de offset.

Existen dos instrucciones de llamada de subrutina. LCALL(llamada larga) y ACALL(llamada absoluta). Cada incremento en el PC, a el primer byte de la siguiente instrucción, entonces empuja a la pila (el primer byte inferior). Salvando ambos bytes de incremento aumenta el apuntador de pila a dos. La dirección de comienzo de la subrutina está codificada en el mismo modo que LJMP y AJMP. La forma genérica de la operación de llamada es el nemónico CALL, el cual ASM51 transladará dentro LCALL o ACALL como sea apropiado. La instrucción de regreso saca los bytes de bajo y alto orden del contador de programa sucesivamente desde la pila, decrementando el apuntador de pila por dos. La ejecución de programa continua en la dirección previamente insertada; el primer byte de la instrucción que sigue inmediatamente a la llamada.

Cuando una solicitud de interrupción es reconocida por el hardware de 8031, dos cosas pasan. El control del programa es vectorizado automáticamente a una rutina de servicio de interrupción empezando a direccionar, mediante el forzar al CPU a procesar una LCALL en vez de la siguiente instrucción. Esto automáticamente guarda la dirección de regreso en la pila.

Seguidamente, la lógica de interrupción es deshabilitada desde la aceptación de cualquier otra interrupción desde la misma o más baja prioridad. Después de completar la rutina de servicio de interrupción, ejecutando una instrucción RETI(regreso de interrupción) regresará la ejecución a el punto donde el programa de respaldo fue interumpido -justo como RET- mientras la restauración de la lógica de interrupcion a el previo estado.

2.25.-INSTRUCCIONES DE OPERAR Y SALTO.

Dos grupos de instruciones combinan una operación de byte con un salto condicional basado en los resultados. CJNE(compara y salto si no son iguales)compara dos operandos de dos bytes y ejecuta el salto si no son iguales. La bandera de acarreo es puesta siguiendo las reglas de la substracción; si el valor entero sin signo de el primer operando es menor que aquél en el segundo es entonces configurado; de otra manera, está borrado. Sin embargo, ningún operando es modificado.

La instrucción CJNE provee el efecto de tipo caso. Es decir, esta instrucción puede ejecutar repetidamente, la comparación de la variable de código a una lista de valores de casos especiales; el segmento de código que sigue una instrucción será ejecutada solo si los operandos son de la misma especie. Comparando el acumulador o un registro a una serie de constantes es un camino conveniente para checar un manejo especial o condiciones de error; si ninguno de los casos se asemeja a los casos, el programa continuará su funcionamiento normal.

Un ejemplo tipico podría ser un dispositivo procesador el cual recibe un código ASCII através de un puerto serie y conduce a una impresora térmica. Una rutina estandarde traduce los caracteres de impresión a patrones de bits, pero los caracteres de control tales como DEL, CR, LF, BEL, ESC o SP deben invocar rutinas especiales correspondientes. Cualquier otro caracter con un código ASCII menor que 20H debe ser

transladado dentro de el valor NUL , 00H y procesado con los caracteres de impresión.

Ejemplo 14.-Tipos de casos usando CJNE.-

```
CHAR   EQU  R7                  ;CODIGO VARIABLE DE CARACTER
INTERP:     CJNE CHAR #7FH,INTP_1
                                ;(RUTINA ESPECIAL DE REBOOT CODE)
            RET
INTP_1:     CJNE CHAR #70H,INTP_2
                                ;(RUTINA ESPECIAL DE CODIGO CAMPANA)
            RET
INTP_2:     CJNE CHAR,#0AH,INT_3
                                ;(RUTINA ESPECIAL DE CODIGO LINE FEED)
            RET
INTP_3:     CJNE CHAR,#0DH,INT_4
                                ;(RUTINA ESPECIAL PARA CODIGO DE RETURN)
            RET
INTP_4:     CJNE CHAR,#1BH,INTP_5
                                ;(RUTINA PARA EL CODIGO DE ESCAPE)
            RET
INTP_5:     CJNE CHAR,#20H,INTP_6
                                ;(RUTINA PARA EL CODIGO DE ESPACIO)
            RET
INTP_6:     JC   PRINTC         ;SALTO SI CODIGO  20H
            MOV  CHAR,#0        ;REEMPLAZAR CARACTERES DE CONTROL
                                ;CON CODIGO NULO
PRINC:                          ;PROCESO ESTANDARDE DE IMPRESION
```

RET

La instrucción DJNZ(decremento y salto si no es cero) decrementa el registro o dirección directa indicada y salta si el resultado no es cero, sin afectar ninguna bandera. Esto provee un modo simple para ejecutar un bucle de programa un número dado de veces, añadir un moderado tiempo de retardo (de 2 a 512 ciclos máquina) con una simple instrucción. Por ejemplo, un bucle de retardo de software de 99 microsegundos puede ser añadido al código forzando un estado lógico bajo en un pin de entrada y/o salida con sólo dos instrucciones.

Ejemplo 15.-Insertando un retraso de software con DJNZ

```
CLR   WR
MOV   R2,#49
DJNZ  R2,$
SETB  WR
```

El signo de pesos en este ejemplo es un caracter especial que significa "la dirección de esta instrucción". La cual es útil en la eliminación de etiquetas en la misma linea de código fuente.

Las instrucciones CJNE y DJNZ(como todos los saltos condicionales) utilizan el direccionamiento de contador de programa relativo para la dirección de destino.

2.26.-OPERACIONES CON LA PILA DEL APUNTADOR(Stack Pointer)-PUSH Y POP.

La instrucción PUSH incrementa el apuntador de pila en uno, luego transfiere los contenidos de una simple variable de byte indicada (solamente en direcionamiento directo) en la localidad direccionada por el apuntador de pila en la RAM interna. Inversamente POP copia los contenidos de la localidad de la RAM interna direccionada por el apuntador de pila a una variables de byte indicada, entonces decrementa el apuntador de pila en uno.

(El direccionamiento de pila sigue las mismas reglas, y direcciona las mismas localidades como un registro indirecto).

Las rutinas de servicio de interrupción no deben cambiar cualquier variable o registro de hardware modificado por el programa principal, o el programa no podría resumir adecuadamente.

(Tal cambio podría parecer un espontáneo error aleatorio).

Los componentes usados o alterados por una rutina de servicio (Acumulador, PSW, etc) deben ser salvados y restaurados a sus valores previos antes de regresar de un servicio de rutina. Las instrucciones PUSH y POP proveén un modo eficiente y conveniente de salvar los estados de los registros en la pila.

Ejemplo 16.-Uso de la pila para salvar categóricamente en las interrupciones.-

```
LOC_TMP EQU          ;RECUERDA LA LOCALIDAD DE CONTADOR
        ORG 0003H    ;SALTO A LA RUTINA DE SERVICIO
                     ;DE INTERRUPCION ACTUAL(LOCALIZADA
                     ;EN CUALQUIER LUGAR)
        ORG LOC_TMP  ;RESTAURA LA LOCALIDAD DEL
                     ;CONTADOR
SERVER: PUSH PSW
        PUSH ACC     ;SALVAR LOS CONTENIDOS DEL
                     ;ACUMULADOR
                     ;NOTESE EL DIRECCIONAMIENTO DIRECTO
        PUSH B       ;SALVA EL REGISTRO B
        PUSH DPL     ;SALVA EL APUNTADOR DE DATOS
        PUSH DPH
```

```
MOV   PSW,#00001000B ;SELECCIONA EL REGISTRO DE
                     ;BANCOS 1
POP   DPH            ;RESTAURA LOS REGISTROS EN ORDEN
                     ;INVERSO
POP   DPL
POP   B
POP   ACC
POP   PSW            ;RESTAURA PSW Y RESELECIONA LOS
                     ;BANCOS
                     ;DE REGISTROS ORIGINALES
RETI      ;REGRESA AL PROGRAMA PRINCIPAL Y
          ;RESTAURA LA INTERRUPCION LOGICA
```

Si el registro SP tenia 1FH cuando la interrupción fue detectada, entonces mientras la rutina de servicio estaba en progreso la pila mantendria los registros mostrados en la figura 13. el SP (apuntador de pila) mantendria 26H.

El ejemplo muestra la situación general: si en la rutina de servicio no se altera el registro B ni el apuntador de datos, por ejemplo, las instrucciones salvar y restaurar los registros no serán necesarias.

La pila puede pasar parámetros a y desde las subrutinas. La subrutina puede indirectamente direccionar los parámetros derivados de los contenidos del apuntador de pila.

Una ventaja aqui es la simplicidad. Las variables no necesitan ser localizadas para parámetros específicos. un número potencialmente largo de parámetros pueden ser pasados, y diferentes llamadas de programas pueden usar diferentes técnicas para determinar o manejar las variables.

Por ejemplo, la subrutina siguiente lee un parametro localizado en la pila por la llamada de programa, utiliza los bits de bajo orden para accesar una tabla local de datos constantes manteniendo los patrones de bit para controlar los embobinados de un motor de pasos de cuatro fases; y guarda el apropiado patrón de bit de regreso en la misma posición en la pila antes de regresar. El acumulador es dejado sin cambio.

Ejemplo 17.-Pasando parámetros de variables a las subrutinas utilizando la pila.

```
NXTPOS:     MOV R0,SP
            DEC R0          ;PARAMETRO DE ACCESO A LOCALIDAD
                            ;PUESTO DENTRO.
            DEC R0
            XCH A,@R0       ;LEE PARAMETROS DE ENTRADA Y SALVA
                            ;LOS CONTENIDOS DE ACUMULADOR
            ANL A,#03H      ;ENMASCARA TODO EXCEPTO LOS 2 BITS
                            ;DE BAJO ORDEN
            ADD A,#2        ;PERMITE VALOR DE OFFSET DESDE MOVC
                            ;A LA TABLA.
            MOVC A,@A+PC    ;LEE LA ENTRADA DE LA TABLA LOOK-UP
            XCH A,@R0       ;PASA DE REGRESO EL VALOR TRANS-
                            ;LADADO Y RESTAURA EL ACUMULADOR
            RET             ;REGRESA AL PROGRAMA ANTERIOR
STPTBL:
            DB 01101111B    ;POSICION 0
            DB 01011111B    ;POSICION 1
            DB 10011111B    ;POSICION 2
            DB 10101111B    ;POSICION 3
```

El programa de regreso puede alcanzar esta subrutina con diferentes secuencias de llamadas, todas las cuales empujan un valor antes de la llamada de rutina y saca el resultado después. Un motor en el puerto 1 puede ser inicializando por la colocación de la posición deseada (cero) en la pila antes de llamar a la subrutina y sacar los resultados directamente un puerto después.

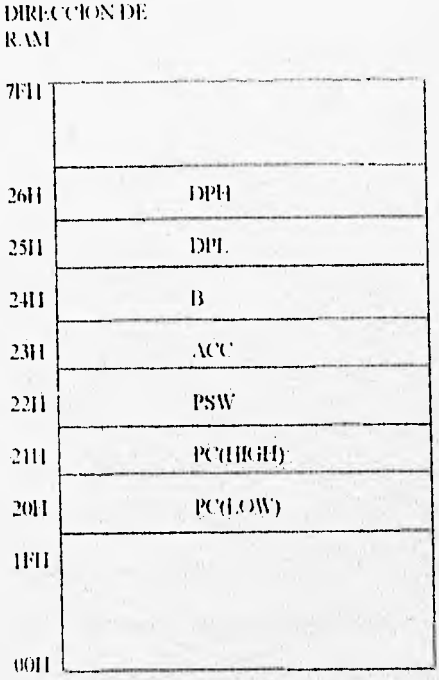
Ejemplo 18.-Enviando y recibiendo los parámetros de datos via la pila.

```
            CLR   A

            PUSH  ACC
```

FIGURA 13.

CONTENIDOS DEL STACK POINTER(APUNTADOR DE PILA)
DURANTE UNA INTERRUPCION



```
        CALL  NXTPOS
        POP   PI
```

Si la posición de el motor está determinada por los contenidos de la variable POSMI(un byte en la RAM interna) y la posición de un segundo motor en un puerto 2 está determinado por los datos de entrada en el nibble de bajo orden del puerto 2, una secuencia de seis instrucciones puede actualizarlas a ambos.

Ejemplo 19.-Cargando y descargando la pila desde los puertos de entrada y salida.

```
POSMI: EQU   51
       PUSH  POSMI
       CALL  NXTPOS
       POP   PI
       PUSH  P2
       CALL  NXTPOS
       POP   P2
```

2.27.-INSTRUCCIONES RELATIVAS AL APUNTADOR DE DATOSDE LA TABLA DE DATOS CONSTANTES.--MOV, INC, MOVC, JMP.

El apuntador de datos puede ser cargado con un valor de 16 bits utilizando la instrucción MOV DTPR, #data 16. Los datos utilizados son puestos en la segunda y tercera instrucción de bytes, el bit de alto orden primero. El apuntador de datos es incrementado por INC DPTR. Un incremento de 16 bits es realizado mediante, un sobre flujo desde el byte bajo que acarreará dentro del byte de alto orden. Ninguna instrucción afecta alguna bandera.

Las instrucciones MOVC(mueve una constante)(MOVC A,@A+DPTR y MOVC A,@A+PC) leen dentro del acumulador los bytes de datos desde el espacio de dirección lógica de la

memoria de programa. Ambos utilizan una forma de direccionamiento indexado: el actual añade los contenidos sin signo de ocho bits del acumulador con el registro del apuntador de datos de 16 bits, y utiliza los resultados de la suma como una dirección desde la cual el byte es traído. Una adición de 16 bits es realizada; un acarreo fuera desde los bits de bajo orden pueden ser propagados através de los bits de alto orden, pero los contenidos del DTPR no son alterados.

La forma tardia utiliza el contador de programa incrementado como el valor base en vez de DPTR(figura 14).

Otra vez, ninguna versión afecta las banderas.

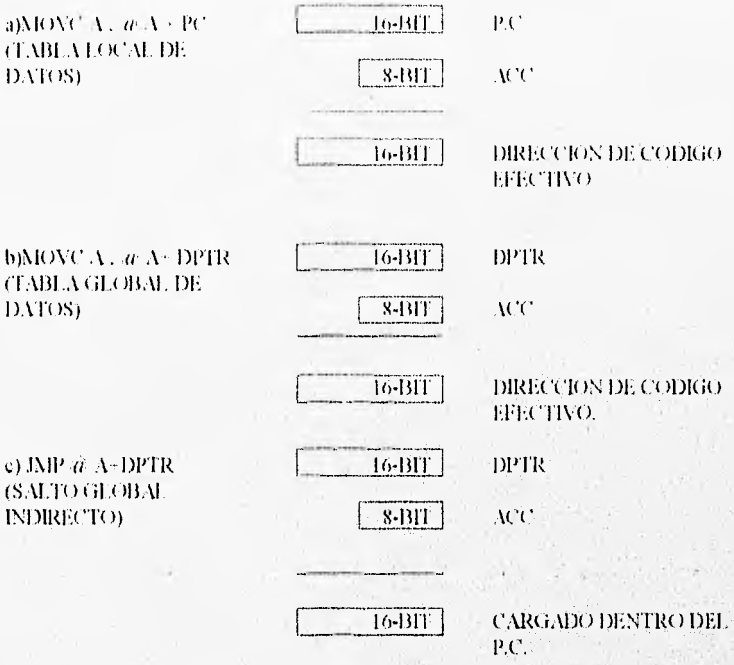
Cada una puede ser parte de una secuencia de tres pasos para accesar las tablas de datos constantes en la ROM. Para utilizar la versión relativa de DPTR, se carga el apuntador de datos con la dirección de comienzo de una tabla de datos constantes; cargando el acumulador con el indice de la entrada deseada, y ejecutar MOVC A,@A+DPTR. La tabla puede ser localizada en cualquier parte en la memoria del programa. El apuntador de datos puede ser cargado con una constante para tablas cortas, O para permitir estructura de datos más complicados, o tablas con más de 256 entradas, los valores para DPH y DPL puede ser computado o modificado con el conjunto de instrucciones estandardes aritméticas.

La versión PC-relativo tiene la ventaja de no afectar el apuntador de datos. Otra vez una secuencia de datos constantes toma tres pasos; carga el acumulador con el indice; compensar el offset desde la instrucción de datos constantes para empezar de la tabla por la adición de un números de bytes separándolos de el acumulador, entonces ejecutar la instrucción MOVC A,@A+PC.

Veamos a una situación no trivial donde esta situación puede ser utilizada. Algunas aplicaciones almacenan tablas de datos constantes multidimensionales para patrones de

FIGURA 14.

INSTRUCCIONES DE LA OPERACION MOVC



matriz de punto, parámetros de calibración no lineales, y en un vector lineal en memoria de programa. Para sacar datos de la tabla, variables representando índices de matrices deben ser convertidos a las deseadas direcciones de entradas de memoria. Para una matriz de dimensiones MxN empezando en la dirección BASE y los respectivos índices I e J, la dirección del elemento (I,J) es determinado por la fórmula

Dirección de entrada = BASE + (NxI) + J

El código mostrado debajo puede accesar cualquier arreglo de hasta 255 entradas(i.e. un arreglo de 11x21 con 231 elementos). Las entradas son definidas utilizando la directiva de DATA BYTE (DB), y será contenida en el código objeto como parte de la propia rutina de acceso.

Ejemplo 20.-Uso del MPY y del apuntador de datos para accesar entradas desde una tabla look-up dimensional en ROM.

```
;MATRIX1  CARGAR CONSTANTE LEIDA DESDE UNA TABLA DE DATOS DE DOS
;DIMENSIONES EN LA MEMORIA DE PROGRAMA DENTRO DEL ACUMULADOR
;UTILIZANDO UNA INSTRUCCION LOCAL DE LOOK-UP,
;MOVC A,@A+PC. EL NUMERO TOTAL DE ENTRADAS ES ASUMIDA CHICA, I.E.
;HASTA 250 ENTRADAS. LA TABLA UTILIZADA EN ESTE EJEMPLO ES DE 11X21
;LA DIRECCION DE ENTRADA DESEADA ES DADA POR LA FORMULA, [(BASE
;ADDRESS) + (21 x INDEXI) + (INDEXJ) ]
INDEXI  EQU R6    ;PRIMERA COORDENADA DE ENTRADA (0-10)
INDEXJ  EQU 23H  ;SEGUNDA COORDENADA DE ENTRADA (0-20)
MATRIX1:
        MOV A,INDEXI
        MOV B,#21
```

```
        MUL AB
        ADD A,INDEXJ
        ;PERMITE INSTRUCCION DE BYTE  ENTRE MOVC Y
        ;ENTRADA (0,0)
        INC  A
        MOVC A, @A+PC
        RET
BASE1:
        DB 1;(ENTRADA 0,0)
        DB 2;(ENTRADA 0,1)
;
        DB 21;(ENTRADA 0,20)
        DB 22;(ENTRADA 1,0)
;
        DB 42;(ENTRADA 1,20)
;
        DB 231;(ENTRADA 10,20)
```

Existen varias y diferentes maneras para enlazar a las secciones de código determinado o seleccionado al tiempo de corrida. (La dirección simple de destino incorporado dentro de saltos condicionales e incondicionales están, por supuesto, determinada al tiempo de ensamblaje). Cada una tiene avances para diferentes aplicaciones.

El más común en un salto condicional de N-caminos basados en algunas variables, con todos los potenciales destinos conocidos en el tiempo de ensamblado. Una de un número pequeño de rutinas es seleccionado de acuerdo a los valores de un variable índice determinada mientras el programa está corriendo. El más eficiente camino para resolver el problema es con MOVC y una instrucción de salto indirecto, utilizando una pequeña tabla de un valor offset de byte en ROM para indicar la dirección de comienzo relativa de varias rutinas.

JMP @A+DPTR es una instrucción que realiza un salto indirecto a una dirección determinada durante la ejecución del programa. La instrucción adiciona un valor de ocho bits sin signo en el acumulador con los contenidos en el apuntador de datos de 16 bits, tal como MOVC A,@A+DPTR. La suma resultante es cargada dentro de un contador de programa y es utilizada como la dirección para ir por instrucciones subsecuentes. Otra vez, una adición de 16 bits es realizada; un acarreo desde los ocho bits de bajo orden puede propagarse através de los bits de alto orden. En este caso, ni los contenidos del acumulador ni los del apuntador de datos son alterados.

El ejemplo de subrutina abajo lee un byte de RAM dentro del acumulador desde una de las cuatro direcciones alternas, como seleccionadas por los contenidos de una variable llamada MEMSEL. La dirección de un byte a ser leído es determinado por los contenidos de R0(y opcionalmente R1). Podría buscarse en una aplicación de impresión, donde cuatro modelos diferentes de impresoras todas utilizan el mismo código ROM pero utilizan diferentes tipos y tamaños de buffer de memoria para diferentes velocidades y opciones.

Ejemplo 21.- N-Caminos de ruta e instrucciones de salto computado vía JMP @ ADPTR.

```
MEMSEL EQU  R3
;
JUMP 4:     MOV  A,MEMSEL
            MOV  DPTR,#JMPTBL
            MOVC A,@A+DPTR
            JMP  @A+DPTR
JMPTBL:     DB   MEMSP0-JMPTBL
            DB   MEMSP1-JMPTBL
            DB   MEMSP2-JMPTBL
            DB   MEMSP3-JMPTBL
```

```
MEMSP0:  MOV  A,@RO  ;LEER DESDE RAM INTERNA
         RET
MEMSP1:  MOVX A,@RO  ;LEER DESDE LOS 256 BYTES DE RAM EXTERNA
         RET
MEMSP2:  MOV  DPL,RO
         MOV  DPH,R1
         MOVX A,@DPTR;LEER DESDE 64K DE RAM EXTERNA
         RET
MEMSP3:  MOV  A,R1
         ANL  A,#07H
         ANL  PL,#11111000B
         ORL  PL,A
         MOVX A,@RO  ;LEER DESDE 4K BYTES DE RAM EXTERNA
         RET
```

Note que esta aproximación es disponible cada vez que el tamaño del salto más la longitud de las rutinas alternas es menor que 256 bytes. La tabla de salto y las rutinas pueden estar localizadas en cualquier parte del programa de memoria, independiente de las páginas de 256 bytes de memoria de programa.

Para aplicaciones donde arriba de 128 destinos deben ser seleccionados, todo lo cual reside en la misma página de 2K de memoria de programa que puede ser alcanzada por las instrucciones de salto absoluto, la siguiente técnica puede ser utilizada. En el arriba mencionado ejemplo de impresión terminal, esta secuencia puede analizar 128 códigos diferentes para los caracteres ASCII que arriban vía el puerto serie del 8031

Ejemplo 22.-N-caminos de ruta con 128 destinos opcionales.

```
OPTION  EQU  R3
;
JMP128:     MOV  A,OPTION
            RL   A.          ;MULTIPLICAR POR 2 PARA TABLA DE
                             ;SALTO DE 2 BYTES
            MOV  DPTR,#INSTBL ;PRIMERA ENTRADA EN TABLA DE
                             ;SALTO
            JMP  @A+DPTR  ;SALTO DENTRO DE TABLA DE SALTO
;
INSTBL:     AJMP PROC00  ;CONSECUTIVO 128
            AJMP PROC01  ;INSTRUCCION AJMP
            AJMP PROC02
            AJMP PROC7E
            AJMP PROC7F
```

El destino en la tabla de salto (PROC00-PROC7F) no son todas necesariamente las rutas únicas. Un largo número de códigos especiales de control puede cada uno ser procesado con su única rutina, con los caracteres de impresión remanentes todos causantes de un rutaje a una rutina especial para entrar el caracter dentro de la fila de impresión de salida.

En esas raras situaciones donde aún las 128 opciones son insuficientes, o donde las rutinas de destino pueden cruzar una frontera de página de 2k, el acercamiento superior puede ser modificado levemente como se muestra abajo.

Ejemplo 23.- Camino de 246 rutas utilizando direccionamiento utilizando tablas de look-up.

```
RTEMP  EQU  R7
;
JMP256:     MOV  DPTR,#ADRTBL:PRIMERA ENTRADA EN LA TABLA
```

```
                        ;DE DATOS
        MOV   A,OPTION
        CLR   C
        RLC   A         ;MULTIPLICA POR 2 PARA 2 BYTES DE
                        ;TABLA DE SALTO.
        JNC   LOW128
        INC   DPH
LOW128: MOV   RTEMP,A   ;SALVA ACC PARA LEER ALTO BYTE
        MOVC  A,@A+DPTR ;LEE BYTE BAJO DESDE TABLA DE
                        ;SALTO
        XCH   A,RTEMP
        INC   A
        MOVC  A,@A+DPTR ;CONSIGUE EL BYTE DE BAJO ORDEN DE
                        ;TABLA
        PUSH  ACC
        MOV   A,RTEMP
        MOVC  A,@A+DPTR ;CONSIGUE BYTE DE ALTO ORDEN DESDE
                        ;TABLA
        PUSH  ACC
;LAS DOS INSTRUCIONES PUSH DE ACC HAN PRODUCIDO UN
;REGRESO DE DIRECION EN LA PILA QUE CORRESPONDE A LA DIRECCION DE
;COMIENZO DESEADA Y PUEDE SER ALCANZADA POR POPPING LA PILA
;DENTRO DEL PC.
        RET
;
ADRTBL  DW    PROC00    ;ARRBA DE 256 DATOS CONSECUTIVOS
```

```
        DW  PROC01    ;PALABRAS INDICANDO DIRECCION DE
                      ;COMIENZO
        DW  PROCFF
;
;
;   CODIGO FANTASMA DIRECCIONES DEFICION NECESITADAS
;   POR LOS DOS EJEMPLOS ANTERIORES
PROC00  NOP
PROC01  NOP
PROC02  NOP
PROC7E  NOP
PROC7F  NOP
PROCFF  NOP
```

2.28.-INSTRUCCIONES DE OPERACIONES BOLEANAS.

Direccionamieto directo de bit.-

Un número de instrucciones operan en variables boleanas de un bit, utilizando el modo de direccionamiento directo de bit comparable al direccionamiento directo de byte. Un byte adicional pegado al "opcode" especifica la variable boleana, un pin I/O o bit de control usado. El estado de cualquiera de estos bits puede ser probado como falso y verdadero con el salto condicional JB(salto en bit) y JNB(salto en no bit). La instrucción JBC(salto en bit y borra) combina una prueba de verdadero con un borrado incondicional.

Como en el direccionamiento directo de byte, el bit 7 de el byte de dirección conmuta entre dos espacios físicos de dirección. Los valores entre 0 y 127 (00H-7FH) define bits en localidades de RAM interna 20H a 2FH(figura 15a); los bytes de direcciones entre 128 y 255(80H-0FFH) definen bits en el espacio de registro de direcciones de 2 x "funciones

FIGURA 15.

a)DIRECCION DE BIT DE RAM

7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	59	58	57	56	55	54	53	52
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH 18H	BANCO 3							
17H 10H	BANCO 2							
0FH 08H	BANCO 1							
07H 00H	BANCO 0							

b)DIRECCION DE BIT DE REGISTRO DE HARDWARE.-

DIR. DIRECTA DE BYTE	BITS DE DIRECCION								SIMBOLO DE REGS. DE HARD-WARE.
0FFH	F7	F6	F5	F4	F3	F2	F1	F0	B
0F0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	--	--	--	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	--	--	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
098H	9F	9E	9D	9C	9B	9A	99	98	SCON
090H	97	96	95	94	93	92	91	90	P1
088H	8F	8E	8D	8C	8B	8A	89	88	TCON
080H	87	86	85	84	83	82	81	80	P0

especiales(figura 15b). Si ninguno 2 x funciones especiales corresponde a la dirección de bit directo utilizado el resultado de la instrucción es indefinida.

Los bits así direccionados pueden tener buenas propiedades. Ellos pueden poner, borrar o complementar con dos instrucciones de dos byte SETB,CLR ó CPL. Los bits pueden ser movidos a y desde la bandera de acarreo con MOV. Las funciones lógicas ANL y ORL pueden ser realizadas entre el acarreo y también el bit direccionado o su complemento.

2.29.-INSTRUCCIONES DE MANIPULACION DE BIT-MOV.

El nemónico MOV puede ser utilizado para cargar un bit direccionable dentro de la bandera de acarreo ("MOVC' C,bit) o para copiar el estado del acarreo a tal bit ("MOV bit,C"). Estas instrucciones son frecuentemente utilizadas para implementar algoritmos de entrada y salida serial vía software o para adaptar estructura de puerto de entrada/salida.

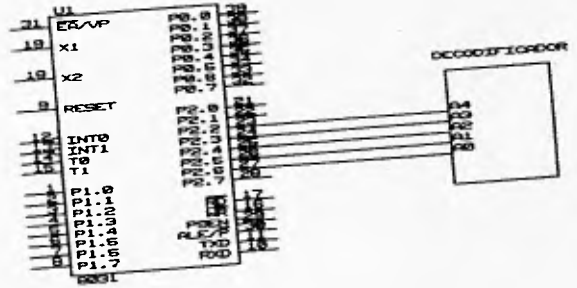
Es algunas veces deseable alterar el orden de los pines de entrada y salida debido a las consideraciones concernientes a las tablillas de circuito impreso. Cuando se interfacea el 8031 a un dispositivo inmediatamente adjunto con pines de entrada, tales como un decodificador columna del teclado, los pines correspondientes son definidamente no alineados.(figura 16).

Existe un intercambio en las interconexiones con tanto los entrelazados de las pistas de circuitos impresos o através software. Lo cual es extremadamente difícil o imposible de realizar con arquitectura de computadora de orientación de byte. El 8031 tiene un conjunto único de instrucciones booleanas que son simples para mover bits individuales entre localidades arbitrarias.

Ejemplo 24.- Reordenando la configuración de puertos de entradas y salidas.

```
OUT_P2:
    RRC  A     :MOVER ACC0 ORIGINAL DENTRO DE CY
```

FIGURA 16



ERROR ENTRE PUERTOS DE SALIDA Y EL DECODIFICADOR

Size	Document Number	REV
A	FIGURA 16	
Date: February 4, 1998	Sheet 1 of 1	

```
MOV  P2.6,C;PONER ACARREO EN EL PIN P26
RRC  A    ;MOVER ACC1 ORIGINAL A CY
MOV  P2.5,C;PONER ACARREO EN PIN P25
RRC  A    ;MOVER ACC2 ORIGINAL A CY
MOV  P2.4,C;MOVER ACARREO A PIN P24
RRC  A    ;MOVER ACC3 ORIGINAL A CY
MOV  P2.3,C;PONER ACARREO A PIN P23
RRC  A    ;MOVER ACC4 ORIGINAL A CY
MOV  P2.2,C;PONER ACARREO A PIN P22
RET
```

2.30.-RESOLVIENDO ECUACIONES LOGICA DE COMBINACIONAL.

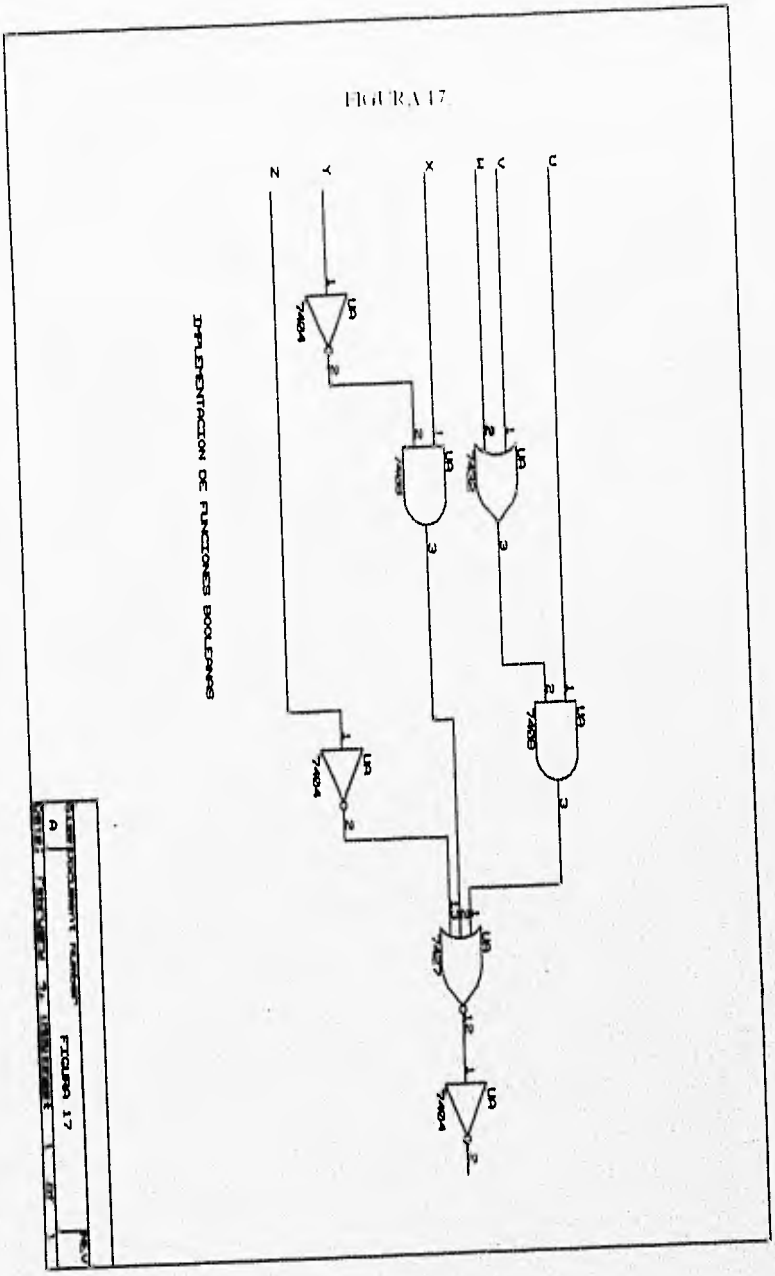
Virtualmente todos los diseñadores de hardware están familiarizados con el problema de resolver funciones complejas utilizando lógica combinacional. Las tecnologías implicadas pueden variar enormemente, desde múltiples contactos de relevadores lógicos ,tubos de vacío, TTL, CMOS hasta aproximaciones esotéricas como con fluidos, pero en cada caso el objetivo es el mismo; una función boleana (falso verdadero) es computada de un número de variables boleanas.

La figura 17 muestra los diagramas lógico para una función arbitraria de seis variables nombradas desde U hasta Z utilizando símbolos lógicos estandardes y símbolos de relevadores lógicos. Cada una es una solución de la ecuación:

$$Q = (U\,(V+W)) + (X\,Y) + Z$$

Esta ecuación puede ser reducida utilizando mapas de Karnaugh ó técnicas algebraicas, ese no es el propósito del ejemplo.

FIGURA 17


IMPLEMENTACION DE FUNCIONES BOOLEANAS
U1
7404
U2
7408
U3
7432
U4
7408
U5
7404
U6
7404
U7
7427
U8
7404
Z
Y
X
V
U
FIGURA 17

La mayoría de las computadoras digitales pueden resolver ecuaciones de este tipo con instrucciones lógicas estandardes y saltos condicionales. Sin embargo, tales soluciones de software parecerían algo inapropiadas debido a la gran cantidad de rutas que pueden existir en el programa.

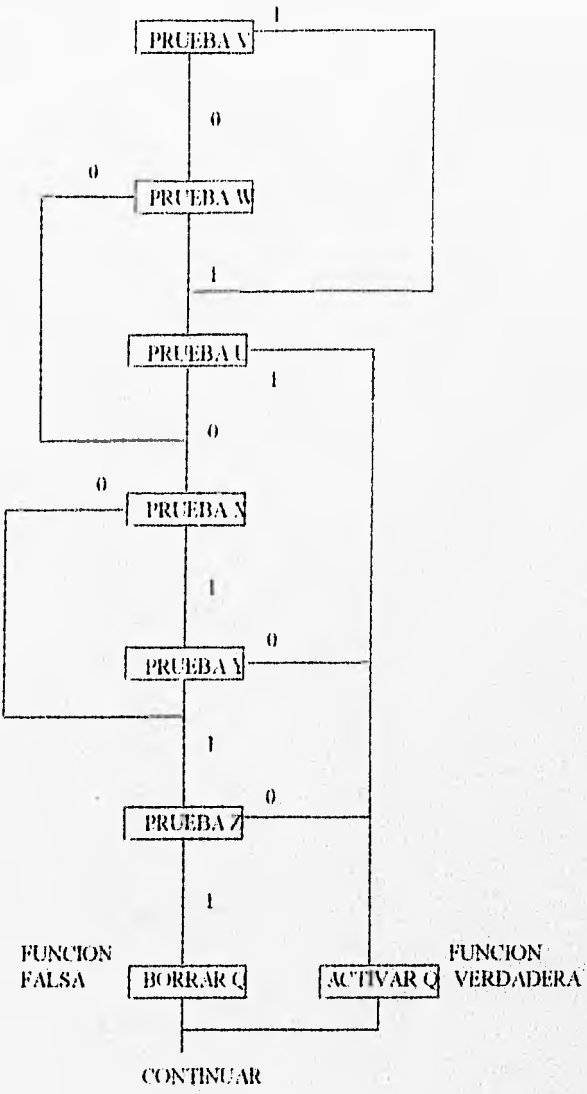
Si se asume que U y V son pines de entradas que siendo leídas por diferentes puertos de entradas, W y X son bits de status para dos controladores periféricos (leídas como puertos de entrada y salida) e Y y Z son banderas de software puestas o borradas anteriormente en el programa. El final resultante debe ser escrito a un pin de salida o en algún tercer puerto. Por el bien de la comparación implementaremos esta función con salida de software desde tres subconjuntos apropiados del conjunto de instrucciones del MCS-51. Las primeras implementaciones siguen el diagrama de flujo mostrada en la figura 18. El flujo del programa embarcaría en una ruta y probará y viajará por las distintas rutas del algoritmo, dejando tanto la salida verdadera y no verdadera llamada ASAP. Estas salidas escriben entonces el puerto de salida con el dato previamente escrito en el mismo puerto con el bit de resultado respectivamente uno ó cero.

En el primer caso, se asume que no hay instrucciones para direccionar bits individuales otros que más que banderas especiales como el acarreo. Esto es típico de muchos microprocesadores antiguos y computadores mainframes diseñados para el truncamiento de números. Los nemónicos del MCS-51 son utilizados aquí, ya que en la mayoría de las otras máquinas el asunto sería aún más poblada por su uso de nemónicos de operación específica como INPUT,OUTPUT,LOAD,STORE etc.. en vez de la universalmente usada MOV.

Ejemplo 25.- Solución de software para función lógica de la figura 17, utilizando solamente instrucciones lógicas de byte ancho.

```
;BFUNC1 RESUELVE UNA FUNCION LOGIC DE SEIS VRIABLE POR ;CARGA Y
```

FIGURA 18.


PRUEBA V
1
0
0
PRUEBA W
1
PRUEBA U
1
0
0
PRUEBA N
1
0
PRUEBA Y
1
0
PRUEBA Z
1
FUNCION
FALSA
BORRAR Q
ACTIVAR Q
FUNCION
VERDADERA
CONTINUAR

```
;ENMASCARAMIENTO DE LOS BITS APROPIADOS EN EL ;ACUMULADOR,
;ENTONCES SE EJECUTA EL SALTO CONDICIONAL; BASADO EN LA CONDICION
;DE CERO.
;(ACERCAMIENTO USADO POR LAS ARQUITECTURAS ORIENTADAS A BYTE)
;LOS VALORES DE BYTE Y MASCARA CORRESPONDEN RESPECTIVAMENTE ;A
LA DIRECCION DE BYTE Y POSICION DE BIT.
;
OUTBUF  EQU  22H   ;ESTADO MAPA DE PIN DE SALIDA
;
TESTV:      MOV  A,P2
            ANL  A,#00000100B
            JNZ  TESTU
            MOV  A,TCON
            ANL  A,#00100000B
            JZ   TESTX
TESTU:      MOV  A,P1
            ANL  A,#00000010B
            JZ   SETG
TESTX:      MOV  A,TCON
            ANL  A,#00001000B
            JZ   TESTZ
            ANL  A,20H
            ANL  A,#00000001B
            JZ   SETG
TESTZ:      MOV  A,21H
            ANL  A,00000010B
```

```
        JZ   SETG
CLRQ:   MOV  A,OUTBUF
        ANL  A,#11101111B
        JMP  OUTG
SETG:   MOV  A,OUTBUF
        ORL  A,#00001000B
OUT:    MOV  OUTBUF,A
        MOV  P3,A
        RET
```

Cada secuencia movimiento/máscara/instrucción de salto condicional debe ser reemplazado por una instrucción de bit simple de prueba gracias al direccionamiento de bit directo. Pero el algoritmo estará igualmente convolucionado.

Ejemplo 26.- Solución de software para funciones lógicas de la figura 17 usando instrucciones de un bit de prueba.

```
;BFUNC2 RESUELVE UNA FUNCION LOGICA DE SEIS VARIABLES
;PROBANDO CADA BIT.
;SIMBOLOS UTILIZADOS EN EL DIAGRAMA LOGICO SIGNADO A LAS
;CORRESPONDIENTES DIRECCIONES DE BIT DEL 8031.
;
U       BIT   P1.1
V       BIT   P2.2
W       BIT   TF0
X       BIT   IE1
Y       BIT   20H,0
Z       BIT   21H,1
```

```
Q       BIT   P3.3
;
TEST_V  JB    V,TEST_U
        JNB   W,TEST_X
TEST_U  JB    U,SET_G
TEST_X  JNB   X,TEST_Z
        JNB   Y,SET-G
TEST_Z  JNB   Z,TEST_G
CLR_G   CLR   Q
        JMP   NXTTST
SET_Q   SETB  Q
NXTTST  ;(CONTINUACION DEL PROGRAMA)
```

Una implementación más elegante y eficiente utiliza las funciones lógicas ANL y ORL para generar la función de salida utilizando código de linea derecha. Esas instrucciones realizan las correspondientes operaciones entre la bandera de acarreo(acumulador boleano) y el bit de direccionamiento, dejando el resultado en el acarreo.

Formas alternas de cada instrucción (especificadas en el lenguaje ensamblador por el colocar un slash antes del nombre del bit) utiliza el complemento del estado del bit como el operando de entrada. Estas instrucciones pueden ser enlazadas juntas para simular una compuerta de múltiple entrada lógica. Una vez acabado, la bandera de acarreo contiene el resultado, el cual puede ser movido directamente al destino o al pin de salida. No es necesario algún diagrama de flujo- es un simple código directamente de los diagramas lógicos en la figura 17.

Ejemplo 27.-Solución de software a las funciones lógicas de la figura 17, utilizando las instrucciones lógicas y variables booleanas patentadas del MCS-51 .

```
;BFUNC3 RESUELVE UNA FUNCION LOGICA DE 6 VARIABLES UTILIZANDO
;INSTRUCCIONES LOGICAS DE LINEA RECTA EN LAS VARIABLES BOOLEANAS
;EN MCS-51.
        MOV  C,V
        ORL  C,W    ;SALIDA DE LA COMPUERTA OR
        ANI  C,U    ;SALIDA DE PUNTA Y COMPUERTA
        MOV  FO,C   ;SALVAR EL ESTADO INTERMEDIO
        MOV  C,X
        ANI  C,Y    ;SALIDA DEL FONDO Y COMPUERTA
        ORL  C,FO   ;INCLUIR EL VALOR SALVADO ANTERIORMENTE
        ORL  C,Z    ;INCLUIR LA ULTIMA ENTRADA VARIABLE
        MOV  Q,C    ;SALIDA DE RESULTADO COMPUTADO.
```

Simplicidad considerable. Rapidez, flexibilidad, confianza, fácilidad de diseño y de compilación.

Las características booleanas son útiles y únicas suficientemente para garantizar un aplicación completa.

2.31.-FUNCIONES PERIFERICAS DISPONIBLES EN EL ENCAPSULADO.

OPERACIONES Y FUNCIONES.

Los resultados de la versatilidad del puerto de entrada/salida desde una estructura cuasi-bidireccional ilustrada en la figura 19(esta es efectivamente la estructura de los puertos 1,2, y 3 para operaciones normales de entrada/salida.

En el puerto 0 el resistor R2 es deshabilitado excepto durante operaciones de bus multiplexado, proporcionando esencialmente salidas de colector abierto.

Una salida lacht de bit asociado con cada pin es actualizado por las instruciones de

FIGURA 19.

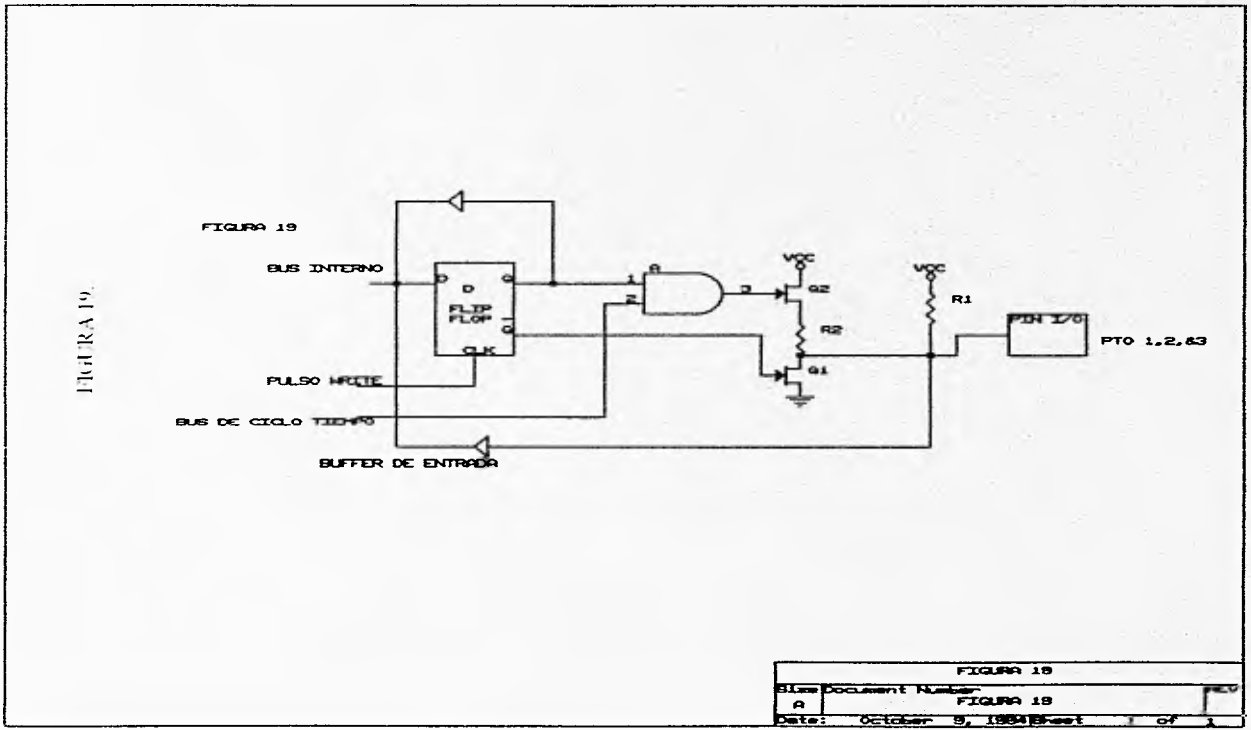


FIGURA 19
BUS INTERNO
D
FLIP FLOP
Q
CLK
PULSO
BUS DE CICLO
BUFFER DE ENTRADA
Q2
Q1
R2
R1
VCC
VCC
PIN I/O
PTO 1,2,&3
FIGURA 18
Size
A
Document Number
FIGURA 18
REV
Date: October 9, 1984
Sheet 1 of 1

direccionamiento directo cuando el puerto es el destino. El estado de lacht es comunicando a el mundo exterior por R1 y Q1, los cuales pueden conducir entradas estandardes TTL.(En términos de la familia TTL, Q1 y R1 parecen una salida de colector abierto con un resistor de pull-up a Vcc). R2 y Q2 representan un dispositivo de carga activa habilitado momentáneamente cuando un 0 de salida previa cambia a 1. Esto contunde al pin de salida a el nivel de 1 más rápido más que las cargas pasivas, improvisando un tiempo de subida significativo si el pin está conduciendo una carga capacitiva. Note que la carga activa es solamente activada en las transiciones de 0 a 1 en el lacth de salida.

Las operaciones utilizando un puerto de salida o un pin como el operando fuente utiliza este nivel lógico en el pin mismo, más que los contenidos de las salidas de lacth. Este nivel es afectado por la computadora misma y cualquier dispositivo que este conectado externamente al pin. El valor leído es esencialmente la función de conjunción (OR) de Q1 y el dispositivo externo. Si el dispositivo externo tiene alta impedancia, tal como una entrada de una compuerta lógica ó una salida de tres estados en el tercer estado, entonces leyendo un pin reflejará el nivel lógico previo de la salida. Para utilizar un pin para entrada, la correspondiente salida de latch debe ser configurada. El dispositivo de salida puede entonces conducir el pin con tanto una señal lógica alta o baja. Así el mismo puerto puede ser utilizado tanto como para entrada y salida por la escritura de unos a todos los pines utilizados como operaciones de entrada y salida, e ignorando todos los pines utilizados como salida en una operación de entrada.

En una instrucción de un operando (INC,DEC,DJNZ y la instrucción boleana) la salida latch más que el nivel de pin de entrada es usado como el dato fuente.

Similarmente, instrucciones de dos operandos utilizando el puerto como una fuente y como destino(ANL,ORL,XRL) utilizando el latch de salida. Esto asegura que los bits de latch

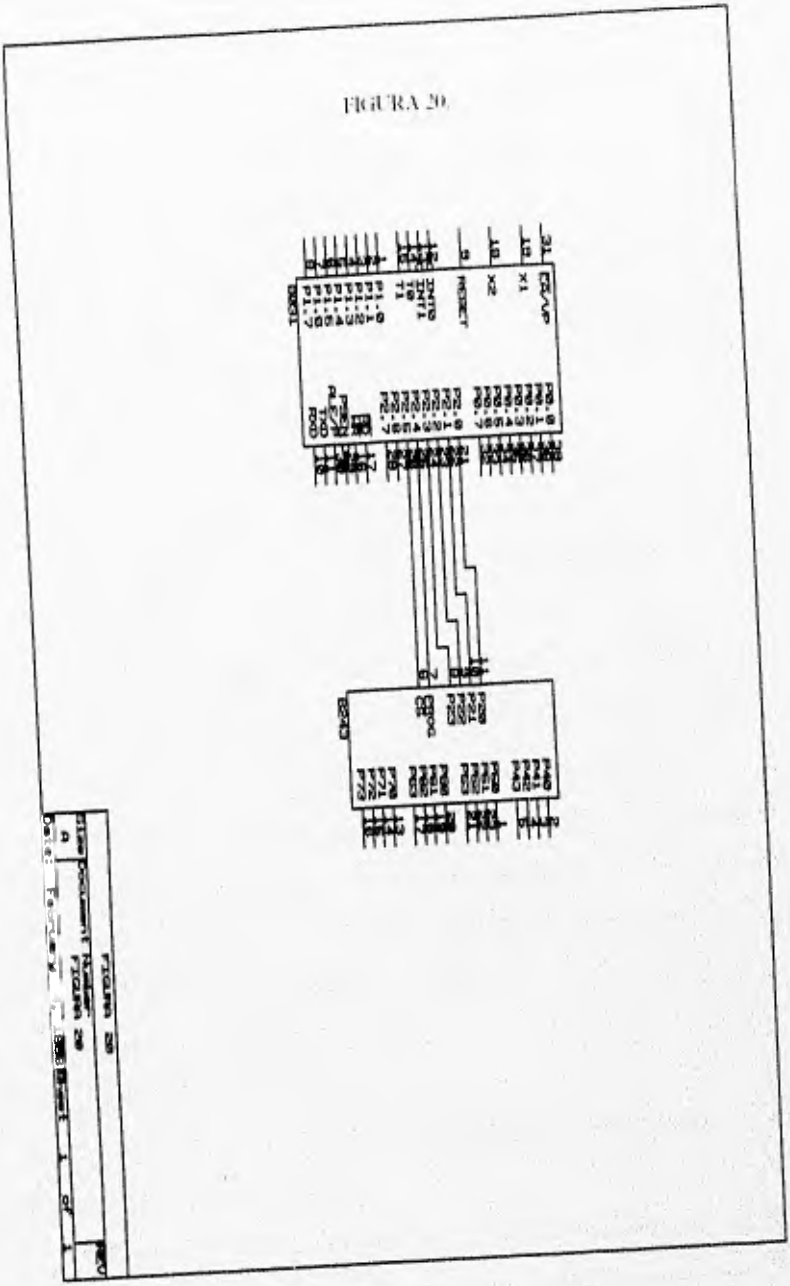
correspondientes a los pines usados como entradas no serán borradas en el proceso de ejecución de estas instrucciones.

La operación booleana JBC prueba el bit latch de salida, más que el pin de entrada, en decidir si realizar el salto o no. Como las sabias operaciones lógicas, operaciones boleanas que modifican pines individuales de un puerto deja otros bits del latch de salida sin cambio. Un buen ejemplo de cómo estos modos pueden ser tomados conjuntamente desde la interfase del procesador huesped esperado por un expansor de entrada y salida del 82431. Aunque aún el 8031 no incluye instrucciones para interfasar con un 8243, las partes pueden ser interconectadas(figura 20) y el protocolo puede ser emulado con el siguiente programa:

Ejemplo 30.-Combinando salidas paralelas, entradas y señales de control en puerto 2

```
;INB243 DATOS DE ENTRADAS DESDE UN EXPANSOR DE ENTRADA Y ;SALIDA
8243 CONECTADOS A P23-P20
;P25 Y PP24 SIMULA CS Y PROG
;P27-P26 UTILIZADO COMO ENTRADA
;PUERTO PARA SER LEER EN ACUMULADOR
INB243:   OR A,#11010000B
          MOV P2,A              ;CODIGO INSTRUCCION SALIDA
          CLR  P2,4             ;CAIDA DE EXTREMO DE PROG.
          ORL  P2,#00001111B    ;CONFIGURAR PARA ENTRADA
          MOV     A,P2          ;LEER DATOS DE ENTRADA
          SET     P2,A          ;REGRESAR A PROG ALTO
          SETB    P2,5          ;DESELECCIONAR CHIP
```

FIGURA 20



2.32.-APLICACIONES DEL PUERTO SERIE Y EL TIMMER.

El configurar el puerto serie del 8051 para una razón de datos y protocolos requeridos esencialmente tres pequeñas secciones de software. En el momento de encendido o de reseteo del hardware las palabras de control del puerto serie y del Timmer deben ser inicializado a los valores apropiados. Software adicional es también necesario en la rutina de transmisión para cargar el registro de datos del puerto serial y en la rutina de recepción para descargar los datos según vayan arribando.

Lo anterior es ilustrado mayormente por un ejemplo cualquiera. Asuma que el 8031 comunicará a un TRC operando a 2400 bits por segundo. Cada caracter es transmitido como datos de siete bits, bit de paridad y un bit de entrada. Esto resulta en una razón de caracter de 2400/10=240 caracteres por segundo.

Para clarificar, las subrutinas de recepción y transmisión son conducidas por status de software escrutando código más que interrupciones. El puerto serial debe ser inicializado a un modo UART de 8 bits.(M0.M1=01) habiliatdos par recibir todos los mensajes (M2=0.REN=1). La bandera indicando que el registro transmisor está libre para más datos serán artificialmente configurados en orden para permitir la salida de software conocer el registro de salida está disponible. Esto puede ser configurado con una instrucción.

Ejemplo 29.- Modo puerto serie y bits de control

```
;SPINIT INICIALIZA PUERTO SERIE
;PARA MODO UART DE 8 BITS
;Y CONFIGURAR LA BANDERA DE TRANSMITIR DISPONIBILIDAD
;
SPINT MOV   SCON,#01010010B
```

El Timmer 1 será utilizado en un modo de auto-carga como un generador de razón de datos. Para lograr una razón de datos de 2400 baudios, el timer debe dividir el reloj interno de 1

MHZ por 32 x (la razón de datos deseada):

$$\frac{1x10E6}{(32)(2400)}$$

que equivale 13.02 lo cual redondeando es igual a 13 ciclos de instrucciones. El timer debe recargar el valor -14 ó 0F3H.(El ASM51 aceptará tanto decimales con signo como representaciones hexadecimales).

Ejemplo 30.-Inicilizando el modo del timer y bits de control.

```
;T1INT INICIALIZA EL TIMER 1 PARA
;AUTOCARGA  EN 32*2400 HZ
;UTILIZADA COMO CONTADOS DE 16 BITS
;
T1NT        MOV  TCON,#10100010B
            MOV  TH1,#-13
            SETB  TR1
```

Una subrutina simple para transmitir el caracter pasado en el acumulador debe computar primeramente el bit de paridad, insertado dentro del byte de dato, esperar hasta que el transmisor está disponible, sacar el caracter y regresar. Esto es cercanamente tan fácil de hacer como se dice.

Ejemplo 31.-Código de salida UART, añadiendo paridad, transmisor de carga.

```
;SP_OUT AÑADE PARIDAD AL ACUMULADOR Y TRANSMITIR CUANDO EL
;PUERTO SERIAL ESTE LISTO.
;
;
SP_OUT:     MOV C,P
            CPL   C
            MOV  ACC 7,C
            JNB   TI,$
```

```
        CLR   TI
        MOV   SBUF,A
        RET
```

Una rutina simple para esperar hasta que un caracter es recibido, poner la bandera de acarreo si hay un error de paridad impar, y regresar al código marcado de siete bits en el acumulador es igualmente corto.

Ejemplo 34.- Código pra recepción via UART y verificación de paridad.

```
;SP_IN ENTRA EL SIGUIENTE CARACTER DESDE EL PUETO SERIAL.
;     COLOCAR ACARREO SI EXISTE ERROR DE PARIDAD
;
SP_IN:JNB   RI,$
        CLR   RI
        MOV   A,SBUF
        MOV   C,P
        CPL   C
        ANL   A,#7FH
        RET
```

Sumario.-

Los diseñadores de sistemas de microcomputadores apreciarán el 8051 como básicamente un solución en un encapsulado para muchos problemas que previamente requerían computadoras a nivel de tarjeta madre. Diseñadores de control de tiempo real hallarán que la ejecución de alta velocidad, periféricos en el encapsulado, y las capacidades de interrupción que es vital para enfrentar los obstáculos de productos previos que requerían diseños lógicos discretos. Y los diseñadores de controles industriales serán capaces de

convertir diagramas de escalera directamente desde prueba y verdadero de TTL ó diseños de relevadores lógicos para software. gracias a las únicas capacidades de procesamiento boleano.

CAPITULO TRES

CAPITULO III.-

3.1.-PROPUESTA DE LOS EJERCICIOS PRACTICOS CON EL

EL MICROCONTROLADOR.-

Este capítulo tiene por objeto presentar los diagramas y los programas sugeridos para la realización de un laboratorio de tales microcontroladores. Se proponen 5 prácticas de diferente grado de dificultad. Además se hablará de los temas relativos a los dispositivos adicionales que se utilizarán en conjunto con el microcontrolador mencionado.

Para la realización de las prácticas no se sugiere el formato de las mismas pues se deja en libertad a los realizadores de las mismas el formato a seguir.

Se propone que tales prácticas se apliquen a la materia de Electrónica digital, pues se puede decir que al fin y al cabo el microcontrolador no es sino un sistema mínimo dentro de un sólo dispositivo.

Cabe mencionar además que un microcontrolador se dirige hacia las aplicaciones concretas en donde, el espacio y el número de los componentes es mínimo; además los cambios o ampliaciones futuras del sistema son casi nulos.

3.2.-DESCRIPCION DE LOS PINES DEL 8051

Refiérase a la figura 21.

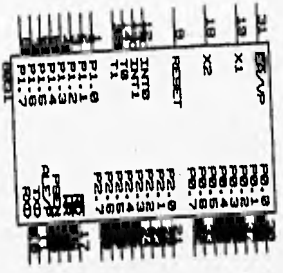
NEMONICO	CONEX.	TIPO	NOMBRE Y FUNCION
VSS	20	ENT.	TIERRA 0V referencia
P0.0-P0.7	39-32	E/S	PUERTO 0.-Es un puerto bidireccional con salidas en colector abierto. Cuando el puerto tiene 1's escritos, las salidas están flotadas y pueden servir como entradas en alta impendancia. El puerto 0 es también multiplexada para obtener el dato y la parte baja de la direccion.
P1.0-P1.7	1-8	E/S	PUERTO 1.- Es un puerto cuasibidireccional,cuando se escriben 1's en el puerto, el puerto puede ser utilizado como entrada.
P2.0-P2.7	21-28	E/S	PUERTO 2.- Es un puerto cuasi-bidireccional con fijadores de nivel internos (pull-up). Cuando se escriben 1s sobre el puerto, las lineas pueden ser utilizadas como entradas o salidas. Como entradas, las lineas que son ex--ternamente colocadas en la po--sición baja proporcionarán una corriente hacia el exterior. El puerto 2 es utilizado además para direccionar memoria ex--terna. Este puerto, emite el byte mas alto de la dirección durante la busqueda de datos en la memoria del programa externo y durante el acceso a las memorias de datos externos que usan direccionamientos de

			16 bits. Durante el acceso a una memoria de datos externa que usa direcciones de 8 bits, el puerto 2 emite el contenido del registro correspondiente a este puerto. que se encuentra en el espacio de funciones especiales.
P3.0-P3.7	10-17	E/S	PUERTO 3.-Es un puerto cuasi-bidireccional con fijadores internos (pull-up). Cuando se escriben 1's sobre el puerto, las lineas pueden ser utilizadas como entradas o salidas. Como entradas, las lineas que son externamente colocadas en la posicion baja proporcionarán una corriente hacia el exte--- rior. El puerto 3 es utilizado además para producir señales de control de dispositivos ex--- ternos como son los siguientes:
	10	E	RxD(P3.0):Puerto serie de entrada
	11	S	TxD(P3.1):Puerto serie de salida.
	12	E	INT0(P3.2)Interrupcion externa.
	13	E	INT1(P3.3)Interrupcion externa.
	14	E	T0(P3.4)Entrada externa timer 0.
	15	E	T1(P3.5)Entrada externa timer 1.
	16	S	WR(P3.6)Habilitador de escritura para memoria externa de datos.

	17	S	RD(P3.7)Habilitador de lectura para memoria externa de datos.
RST	9	E	Reset.- Una entrada alta en ésta linea durante dos ciclos de ma---- quina.mientras el oscilador está corriendo, detiene el dispositivo. Un resistor interno conectado a Vss permite un alto en la fuente usando solamente un capacitor externo a Vcc.
ALE	30	E/S	Address latch enable.- Un pulso positivo de salida permite fijar el byte bajo de la dirección du-- rante el acceso a una memoria externa. En operación normal, ALE es emitido en un rango constante de 1/6 de la frecuencia del oscilador y puede ser usada para cronometrar. Note que un pulso del ALE es omitido durante cada acceso a la memoria de datos externos.
PSEN	29	S	Program Store Enable.- Habilita-- dor de lectura para memoria de programas externos. Cuando el 8031/8051 está ejecutando un codigo de una memoria de pro-- gramas externos, PSEN es acti-- vada dos veces por cada ciclo de máquina, excepto cuando se accesa la memoria de datos ex-- terna que omiten las dos activa-- vaciones del PSEN externos. PSEN tampoco es activada cuando se usa la memoria de programas internos.
EA	31	E	External access enable.- EA debe mantenerse externamente en posicion baja para habilitar el mecanismo que elige el codigo de las localizaciones de la me-- moria de programas externos,

			0000H y 0FFFH. Si EA se mantiene en posicion alta, el dispositivo ejecuta los programas que se encuentran en la memoria interna ROM, a menos que el contador del programa contenga una direccion mayor a 0FFFH.
XTAL 1	19	E	Crystal 1.- Es la entrada del cristal para el circuito oscilador (generador del reloj interno) que amplifica e invierte la entrada.
XTAL2	18	S	Crystal 2.-Es la salida del amplificador oscilador inversor

FIGURA 21



DESCRIPCION DE LAS PRACTICAS SUGERIDAS.-

3.3.-PRACTICA NUMERO I.-

Simulacion de una UART (Universal Asynchronous Receiver Transmitter)

Objetivo.-

-Adquirir experiencia en el diseño e implementacion de sistemas digitales conformados alderedor del microcontrolador 8031.

-Conocer la relacion existente entre un microcontrolador, la memoria y los elementos de entrada/salida con los que normalmente trabaja este.

Descripcion.-

Lo que en esta práctica se pretende realizar es una simulaciónde una UART(Universal Asynchronous Receiver Transmitter) mediante el microcontrolador. Si bien se reconoce que tal función la puede realizar un circuito integrado por si solo, el énfasis en utilizar el microcontrolador en esta primera práctica es el ofrecer un primer acercamiento para producir una familiaridad suficiente, y así; poder proseguir las siguientes prácticas propuestas en este trabajo.Una UART(Universal Asynchronous Receiver Transmitter) es básicamente un registro de cambio, es decir; convierte salida o entrada en formato paralelo de bytes a formato serial. Lo contrario también aplica.

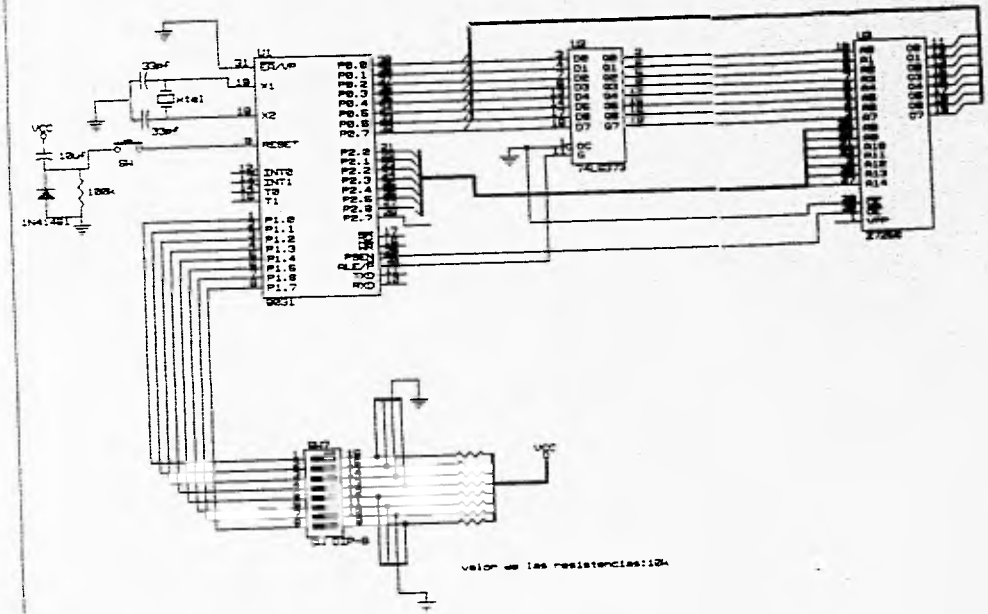
Una UART tiene tres secciones, a saber; un receptor,un transmisor y un controlador. El receptor se aplica en la recepción de una entrada serial desde un periférico y una señal de reloj. Este receptor convierte la señal en formato serial a formato en paralelo y coordina la señal con el microprocesador.

La señal en formato paralelo es entoces colocada en el bus de datos y es leido por el

procesador. El transmisor es alimentado con la señal en paralelo que le envía el procesador y el bus de datos. Convierte el formato paralelo a el formato serie y envía esta señal al puerto de interface. El controlador recibe información de control desde el procesador y se asegura que la operación se realize en forma apropiada.

La UART maneja el comienzo, el paro y los bits de paridad para que la operación asincronica se lleve a cabo de manera adecuada. Durante la transmisión desde la computadora a un periférico la UART los bits tanto de comienzo, de paro y los de paridad en los bits de datos. Cuando la UART recibe caracteres desde un periférico, remueve los bits tanto de comienzo como de paro y checa los de paridad. La UART es comunmente usada con modems y en algunas ocasiones trabaja con las salidas de impresoras seriales.

El diagrama de la primera práctica se muestra a continuación, cabe hacer notar que, en el diagrama mencionado; se detallan las caracteristicas de los elementos implicados en la realización de la práctica. Una vez más se hace énfasis, que el formato de la realización del trabajo escrito del reporte de dicha práctica se deja al libre criterio del que realiza el trabajo.



PRACTICA NUM. 2

SISTEMA MINIMO 1

PROGRAMA 1.

```
;*****************************************
;UNIVERSIDAD NACIONAL AUTONOMA DE MEXCIO
;FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN
;PROGRAMA DE SIMULACION DE UART
;SISTEMA MINIMO NUMERO 1
;*****************************************

        ORG 000H
        MOV SCON,#00000000B
        MOV SBUF,P1
LOOP:   MOV C,P1.0      ;MUEVE DATO DEL PIN 0 DEL PTO.1 AL CARRY
        MOV TXD,C       ;MUEVE EL CARRY AL PIN TXD
        MOV C,P1.1      ;MUEVE DATO DEL PIN 1 DEL PTO. 1 AL CARRY
        MOV TXD,C
        MOV C,P1.2
        MOV TXD,C
        MOV C,P1.3
        MOV TXD,C
        MOV C,P1.4
        MOV TXD,C
        MOV C,P1.5
        MOV TXD,C
        MOV C,P1.6
        MOV TXD,C
        MOV C,P1.7
        MOV TXD,C
        SJMP LOOP       ;SALTA INCONDICIONALMENTE A LOOP.
        END
        END[]
```

3.4.-PRACTICA II.-

Manejo de un exhibidor alfanumérico y un teclado matricial.

Objetivo.- Desarrollar un sistema con exhibidor alfanumérico y teclado

En la mayoria de los sitemas a desarrollar, resulta indispensable el uso de un exhibidor, el cual, nos muestra mediante mensajes escritos las demándas del aparato, los requerimientos, los mandatos externos, las señalizaciones, las alarmas etc. El uso de un exhibidor alfanumérico nos proparciona la solución a esta necesidad, facilitando el manejo y aprendizaje del aparato ó dispositivo que ha sido implementado.

En esta práctica se propone utilizar el Modulo de cristal liquido de dos lineas por 16 caracteres por linea, el cual es fabricado por la compañia AND.

Este exhibidor puede interconectarse directamente con el bus de datos de cualquier microprocesador o microcontrolador, gracias a que tiene un bus de datos con tres estados. Además contiene una memoria Ram que le permite almecenar hasta 128 caracteres y una memoria Rom con 160 caracteres matriciales de 5x27 puntos y 30 caracteres de 5x10 puntos.

La ventana del exhibidor permite ver 32 caracteres a la vez, en dos lineas de 16 caracteres cada una. (Ver figura del exhibidor).

El procedimiento para inicializar el exhibidor es el siguiente:

1.-Primeramente se establece el tipo de interfase a la cual el exhibidor se va a conectar, en este caso, se trata de un microcontrolador con un bus de datos de 8 bits, el cual se conecta directamente.

La primera palabra de control que se envia al exhibidor es el numero 38H, el cual significa

lo siguiente:

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
38H	0	0	0	0	1	1	1	0	0	0

Se envia la palabra de control al exhibidor(RS=0 y R/W=0). los bits DB4 y DB5 especifican el tamaño del bus. y el bit DB3 el número de lineas del exhibidor.

Se espera un lapso de tiempo de 40 microsegundos antes de enviar la siguiente instrucción.

NOTA: Cada instrucción, toma un cierto tiempo de ejecución que va de 40 microsegundos a 1.64 milisegundos.

2.-Se limpia toda la memoria del exhibidor y se regresa la pantalla del exhibidor a su posición incial.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
01H	0	0	0	0	0	0	0	0	0	1

Esta instrucción toma un tiempo de 1.64 milisegundos.

3.-Se establece el movimiento del cursor hacia la derecha de la pantalla del exhibidor permanece lija con la entrada de los caracteres.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
06H	0	0	0	0	0	0	0	1	1	0

Esta instrucción toma un tiempo de 40 milisegundos

4.-Se prende la pantalla del exhibidor, se activa el cursor indicando la posición del próximo caracter de entrada y se desactiva el parpadeo.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0EH	0	0	0	0	0	0	1	1	1	0

Se posiciona el cursor en el primer caracter y la primera linea.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
80H	0	0	1	0	0	0	0	0	0	0

Esta instrucción toma un tiempo de 40 microsegundos

6.- A partir de aquí se puede comenzar a enviar los caracteres que se desean exhibir dejando un espacio entre cada uno de éllos de 40 microsegundos como mínimo y con RS = 1.

Por ejemplo se enviarán las letras A y B. por lo tanto se escribirá el siguiente código ASCII para la letra A, el cual se ejecuta en 40 microsegundos.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
41H	1	0	0	1	0	0	0	0	0	1

Ahora para la letra B.

CODIGO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
42H	1	0	0	1	0	0	0	0	1	0.

EXHIBIDOR LCD AND 1481
VCC
1 4 7
2 5 8
3 6 9
0 *

PROGRAMA 2

```
;*************************************************************
;PROGRAMA PARA EL MANEJO DEL EXHIBIDOR ALFANUMERICO
;UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
;FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN IZCALLI EDO. DE
;MEXICO.
;SISTEMA MINIMO 2
;*************************************************************
            ORG 0100H
            ;LAS DIRECCIONES DEL EXHIBIDOR SON:
            ;8000H PARA CONTROL DEL EXHIBIDOR
            ;8001H PARA EXHIBICION DE CARACTER
            ;APUNTADAS POR LOS REGISTROS R0 Y P2.
EXHIBE:     MOV DPTR , #CONTEX ;CONTROL DEL EXHIBIDOR.
            MOV R0 , #00H
            MOV P2 , #80H
            LCALL XCBDOR
            ;ENVIA LOS CARACTERTES DE CONTROPL AL EXHIBIDOR

            INC DPTR
            LCALL XCBDOR
            ;SE ENVIA EL CARACTER 01 DE CONTROL
            ;(LIMPIA PANTALLA) EL CUAL REQUIERE
            ;1.64 mSEG PARA SU EJECUCION.

            LCALL LIMPIA
            ;CARGA EL APUNTADOR CON EL PRIMER MENSAJE

BIEN:       MOV DPTR , #BIENV
            ;SE ENVIA A LA RUTINA QUE EXHIBE EN LAS DOS
            ;LINEAS EN UNA VEZ.
            LCALL DOBLEX
            LCALL TIME
            ;SE ENVIA EL CARACTER 01 DE CONTROL
            ;(LIMPIA PANTALLA), EL CUAL REQUIERE
            ;1.64 mSEG. PARA SU EJECUCION.
            MOV R0 , #00
            LCALL LIMPIA
            ;SE ENVIA UN SOLO MENSAJE A LA PRIMERA
            ;LINEA A PARTIR DE LA COLUMNA 5.
            MOV DPTR , #MEDIO
            MOV A , #85H      ;ESCRIBIR EN LA 1era.
            LCALL COEXH       ;LINEA (5ta COLUMNA)
```

```
                LCALL TIME
                LCALL LIMPIA

TEREXH:         SJMP BIEN
                ;*****************************************************
        ;*********SUBRUTINA DE CONTROL Y PRE.*****************
        ;**********SENTACION DEL EXHIBIDOR  *****************
        ;*****************************************************

DOBLEX:         MOV A, #80H     ;ESCRIBE EN LA 1era.
                ACALL COEXH     ;LINEA (1era. COLUMNA)
                INC DPTR
                MOV A, #0C0H    ;ESCRIBE EN LA 2da.
                ACALL COEXH     ;LINEA (1era COLUMNA)
FEXH:           RET
COEXH:          MOV R0 , #00H
                MOV P2 , #80H
                MOVX @R0 , A
                ACALL QARNTA
                INC R0
                ACALL XCBDOR
                RET
XCBDOR:         CLR A
                MOVC A,@A+DPTR
                JZ TERMIN
                MOVX @R0 , A
                ACALL QARNTA    ; TIEMPO 40 uSEGS.
                INC DPTR
                SJMP XCBDOR
TERMIN:         RET
                ;*************************************************
        ;*********SUBRUTINA DE TIEMPO DE 40 uS*************
        ;*************************************************
QARNTA:         MOV R7 , #20
TIEMPO:         DJNZ R7 , TIEMPO
                RET
                ;*************************************************
        ;*********SUBRUTINA QUE ENVIA EL CARACTER********
        ;*********01 DE CONTROL, Y ADEMAS CONSUME********
        ;*********LOS 1.64 mSEG. PARA SU EJECUCION*******
        ;*************************************************
```

```
LIMPIA:         MOV DPTR , #CLEAR
                LCALL XCBDOR
                MOV R6 , #40
TI1600:         LCALL QARNTA     ;TIEMPO DE 40 uSEG.
                DJNZ R6 , TI1600
                RET
                ;************************************************
      ;***********SUBRUTINA DE TIEMPO 2 SEG  *********
      ;************************************************

TIME:     MOV R1 , #02
E2:       MOV R0 , #90
E3:       MOV R6 , #99
E4:       MOV R7 , #50
WAIT:     DJNZ R7 , WAIT
          DJNZ R6 , E4
          DJNZ R0 , E3
          DJNZ R1 , E2
          RET


      ;************************************************
      ;*********TABLA DE MENSAJES DEL EXHIBIDOR**********
      ;************************************************
                ;ORG 1000H
CONTEX:         DB 38H
                DB 01H
                DB 06H
                DB 0EH
INIEXH:         DB 80H
                DB 00H
CLEAR:          DB 01H
                DB 00H
BIENV:          DB 'BIENVENIDOS AL '
                DB 00H
                DB 'SISTEMA UNAM-8031 '
                DB 00H
MEDIO:          DB 'ENMEDIO '
                DB 00H
                RET
                END
                END
```

```
;*************************************************
;**  PROGRAMA PARA TOMAR DATOS DEL TECLADO
;**  Y ENVIARLOS AL EXHIBIDOR
;**   DIRECCIONES DEL EXHIBIDOR
;**   8000H = CONTROL DEL EXHIBIDOR
;**   8001H = CONTROL DEL DATO
;**   DIRECCION DEL TECLADO = 9000H
;**   EL DA ACTIVA LA INTO DE MICROCONTROLADOR
;**   SISTEMA MINIMO 2
;*************************************************


          ORG 000H
TEC:      EQU 9000H
EXHI:     EQU 8000H
          LJMP TECLAD
          ORG 03H
          SETB 20H.0
          MOV DPTR,#TEC
          MOVX A,@DPTR
          ANL A,#0FH
          ADD A,#30H
          RETI

          ;PROGRAMA PRINCIPAL

          ORG 100
TECLAD:   MOV TCON,#01H
          MOV DPTR,#CONTRL
          MOV R0,#00H
          ACALL SUBEXH
          MOV DPTR,#TEXTO
          ACALL SUBEX1
          MOV A,#0C0H
          ACALL POSCUR
          MOV DPTR,#TEXT1
          ACALL SUBEX1
ESPTEC:   JNB 20H.0,ESPTEC
          CLR 20H.0
          MOV DPTR,#8001H
          MOVX @DPTR,A
          MOV R7,#20H
```

```
TEX:    DJNZ R7,TEX
        MOV A,#10H  ;MUEVE EL CURSOR A LA IZQUIERDA
        ACALL POSCUR


        ;SE REPITE EL DATO CONTINUAMENTE
        SJMP ESPTEC
        ;***********************************************
        ;***** SUBRUTINA DE EXHIBICION  **************
        ;***********************************************
SUBEX1:     MOV R0,#01H
SUBEXH:     MOV P2,#80H
SUBEX:      CLR A
        MOVC A,@A+DPTR
        JZ FINEXH
        MOVX @R0,A
        MOV R2,#10H
LAZEX2: MOV R1,#0FFH
LAZEXH: DJNZ R1,LAZEXH
        DJNZ R2,LAZEX2
        INC DPTR
        SJMP SUBEXH
FINEXH: RET

POSCUR: MOV DPTR,#EXHI
        MOVX @DPTR,A
        MOV R1,#0FFH
LAZPOS: DJNZ R1,LAZPOS
        RET

        ;TEXTOS
CONTRL: DB 38H,01H,02H,06H,0FH,80H,00H
        DB 00H
TEXTO:  DB 'OPRIMA UNA TECLA'
        DB 00H
TEXT1:  DB 'TECLA ------>'
        DB 00H
        END
        ENDI1
```

3.5.-PRACTICA NUMERO III.-
Objetivo:

Interconexionando un motor de pasos con el microcontrolador

8051.

En muchas aplicaciones de control automático, es necesario el accionamiento de válvulas o de sistemas de engranes con una exactitud y precisión muy altas. En Robótica, son indispensables estas características, donde las manos y los brazos mecánicos deben ejecutar movimientos de gran precisión. Existen muchas otras ramas de la electrónica donde la utilización de dispositivos de posicionamientos mecánicos son indispensables.

Un motor de pasos resuelve en gran medida este problema, ya que su principio de funcionamiento le permite realizar pequeños movimientos (pasos), con gran exactitud y repetibilidad.

Un motor de pasos es un motor eléctrico cuyo eje gira una cantidad especifica por cada pulso de entrada que recibe, lo cual permite el control de posición, velocidad y sentido de dirección. El número de pasos varía según sea la aplicación que se requiera. Existen en el mercado desde 0.1 a 120 grados. Los ángulos más comunes son de 1.8,2.0,2.5,5.0,15 y 30 grados, que respectivamente dan 200,180,144,72,24 y 12 pasos por revolución. Los motores de pasos son alimentados con fuentes de corriente directa y manejados con circuitería lógica.

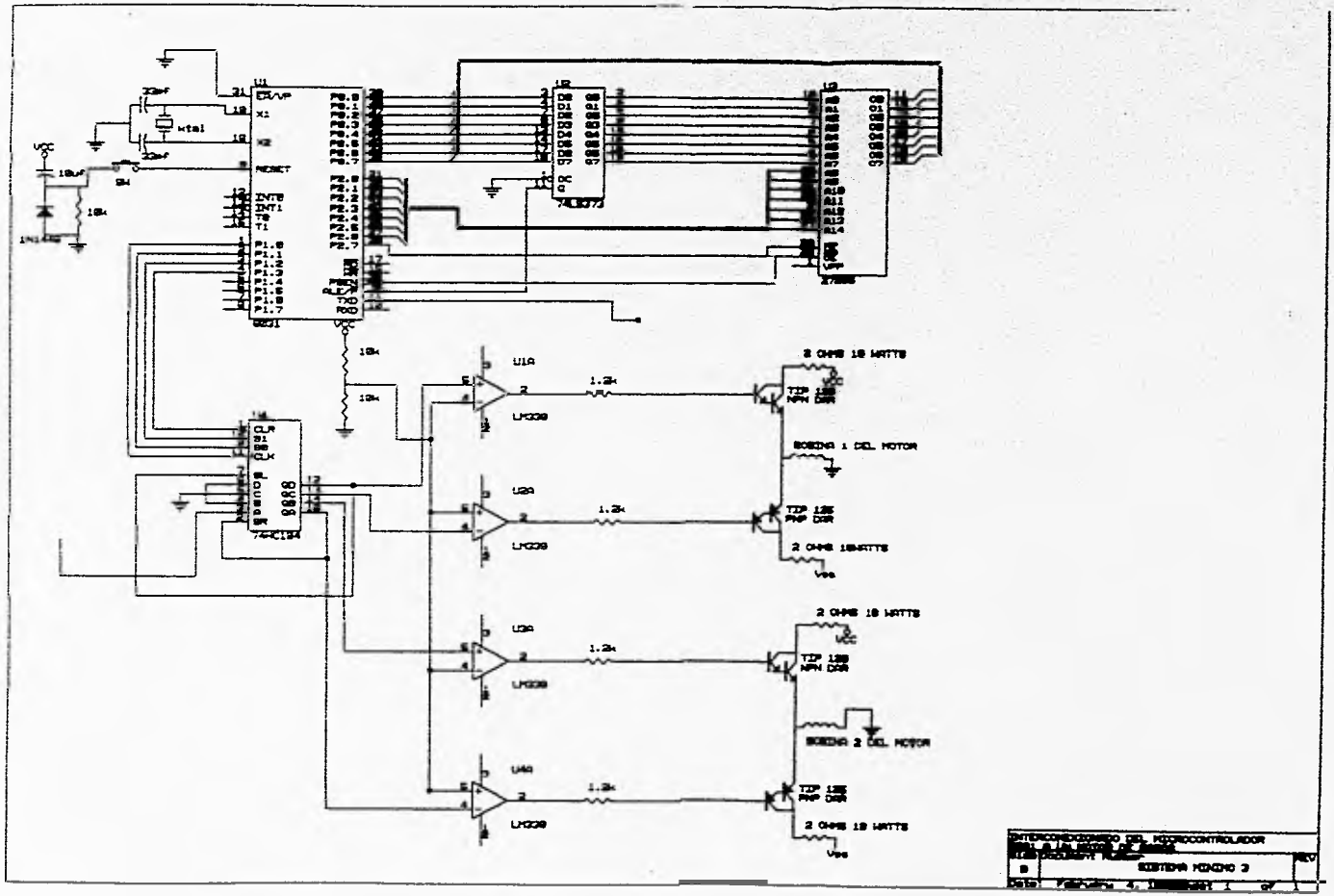
El circuito que se propone para la activación de estas bobinas es el esquema de la práctica. Cuando el transistor 1 es activado, una parte de la bobina queda energizada positivamente y la otra negativamente. Cuando el transistor 1 se desactiva y el transistor 2 se activa, sucede lo inverso y, de esta manera se invierten las polaridades. Este circuito se presenta en cada una de las bobinas.

Cabe mencionar que el circuito 74194 el cual es un registro de corrimiento de 4 bits, utiliza la siguiente logica de control:

TABLA DE ESTADOS LOGICO

CLR	S0	S1	CLK	DEFINICION
1	X	X	X	Deshabilita el circuito.
1	0	0	X	No cambian las salidas.
1	0	1	α	Desplaza a la der. en serie.
1	1	0	α	Desplaza a la izq. en serie.
1	1	1	α	Carga el dato en paralelo.

En el esquema del circuito de la práctica se puede apreciar que se utilizan las lineas de PL0 a PL3, para poder controlar el CLR,S0,S1 y CLK respectivamente. La primera linea CLR esactiva el circuito. es decir; ninguna de las dos bobinas se encuentran energizadas. Mediante S0=1,S1=1 y un pulso transitorio positivo por el clk, el dato paralelo es cargado. Un 1 es cargado en una sola de las lineas de salida y las demás con 0, con el fin de que solo sea energizada una bobina por paso y en un solo sentido. El sentido de giro del motor se debe al valor de S1 y S0.


U1
X1
X2
RESET
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
74LS373
74HC154
LM339
1.2k
2 OHMS 10 WATTS
TIP 122
NPN DAR
TIP 127
PNP DAR
BOBINA 1 DEL MOTOR
BOBINA 2 DEL MOTOR
Vcc

PROGRAMA 3

```
;*****************************************************
;PROGRAMA PARA EL CONTROL DE MOTORES DE PASOS
;SE UTILIZA EL T0 COMO BASE DE TIEMPO
;T0=10,000 M SEG.
;100 PASOS POR SEGUND, 1 REVOLUCION =2 SEG.
;PRIMERO GIRARA DURANTE 10 SEGUNDOS A LA DERECHA; DESPUES
;PARARA DURANTE DOS SEGUNDOS; A CONTINUACION, GIRARA HACIA
;LA IZQUIERDA REPETIRA LA SECUENCIA INDEFINIDAMENTE.
;SISTEMA MINIMO 3
;UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
;FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN
;*****************************************************

        ORG 00H
        LJMP MOTOR
        ORG 0BH
        MOV TH0,#0D8H ;SE RECARGA EL T0 CON
        MOV TL0,#0F0H  ; LA BASE DE T=-10,000 MSEG.
        RETI
MOTOR:  MOV TH0,#0D8H  ;SE CARGA CON LA BASE
        MOV TL0,#0F0H  ;DE TIEMPO 10,000 MSEG.
        CLR P1.3       ;SE DESACTIVA EL MOTOR
        SETB P1.3      ;SE VUELVE A ACTIVAR
        CLR P1.0       ;SE CARGA EL VALOR INICIAL
        SETB P1.0      ;DE CORRIMIENTO "0001"

        ;******EL MOVIMIENTO SE REALIZA A LA DERECHA*******
        ;**************    S0=1 Y S1=0    ****************


        MOV TMOD, #01H   ;SE ESTABLECE T0 COMO
        MOV TCON, #10H   ;TEMPORIZADOR  EN MODO 1
        MOV IE, #82H     ;SE PERMITE LA INTER. T0
REPITE: CLR P1.1       ;S1=0
        SETB P1.2      ;S0=1

        ;SE LLAMA A LA RUTINA DE PASOS
        ACALL PSS

        ;SE LLAMA A LA RUTINA DE PARO
        ACALL PARO
```

```
        ;***** EL MOVIMIENTO SE REALIZARA A LA IZQUIERDA*******
        ;*****          S0=0 Y S1=1                 *******

        CLR P1.2  ;S0=0
        SETB P1.1 ;S1=1

        ;SE LLAMA A LA RUTINA DE PASOS
        ACALL PSS

        ;SE LLAMA A LA RUTINA DE PARO
        ACALL PARO

        ;SE REPITE LA SECUENCIA DE MANERA INDEFINIDA
        SJMP REPITE

        ;****** RUTINA DE MOVIMIENTO DEL MOTOR***************


PSS:    MOV R3,#07H       ;R2R3 COMO CONTADORES DE
PAS1:   MOV R2,#0D0H      ;200 PASOS=10 VUELTAS
PASO:   JNB TCON.5, PASO  ;ESPERA 10 MSEG
        CLR P1.0          ;SE ENVIA UN PULSO DE
        DJNZ R2,PASO
        DJNZ R3,PAS1
        RET

        ;****** RUTINA DE PARO DEL MOTOR.**************


PARO:   CLR P1.3    ;SE DESACTIVA EL MOTOR
        MOV R4, #20     ;CONTADORES PARA DOS 2 SEG.
        MOV R5, #00         ;DE PARO TOTAL DEL MOTOR
        MOV R6, #00
TIEMPO: DJNZ R6,TIEMPO
        DJNZ R5,TIEMPO
        DJNZ R4,TIEMPO
        SETB P1.3   ; SE VUELVE A ACTIVAR EL MOTOR
        CLR P1.0        ; SE CARGA EL VALOR INICIAL
        SETB P1.0       ; DE CORRIEMIENTO "0001"
        RET
        END
        ENDLI
```

3.6.-PRACTICA NUMERO IV.-

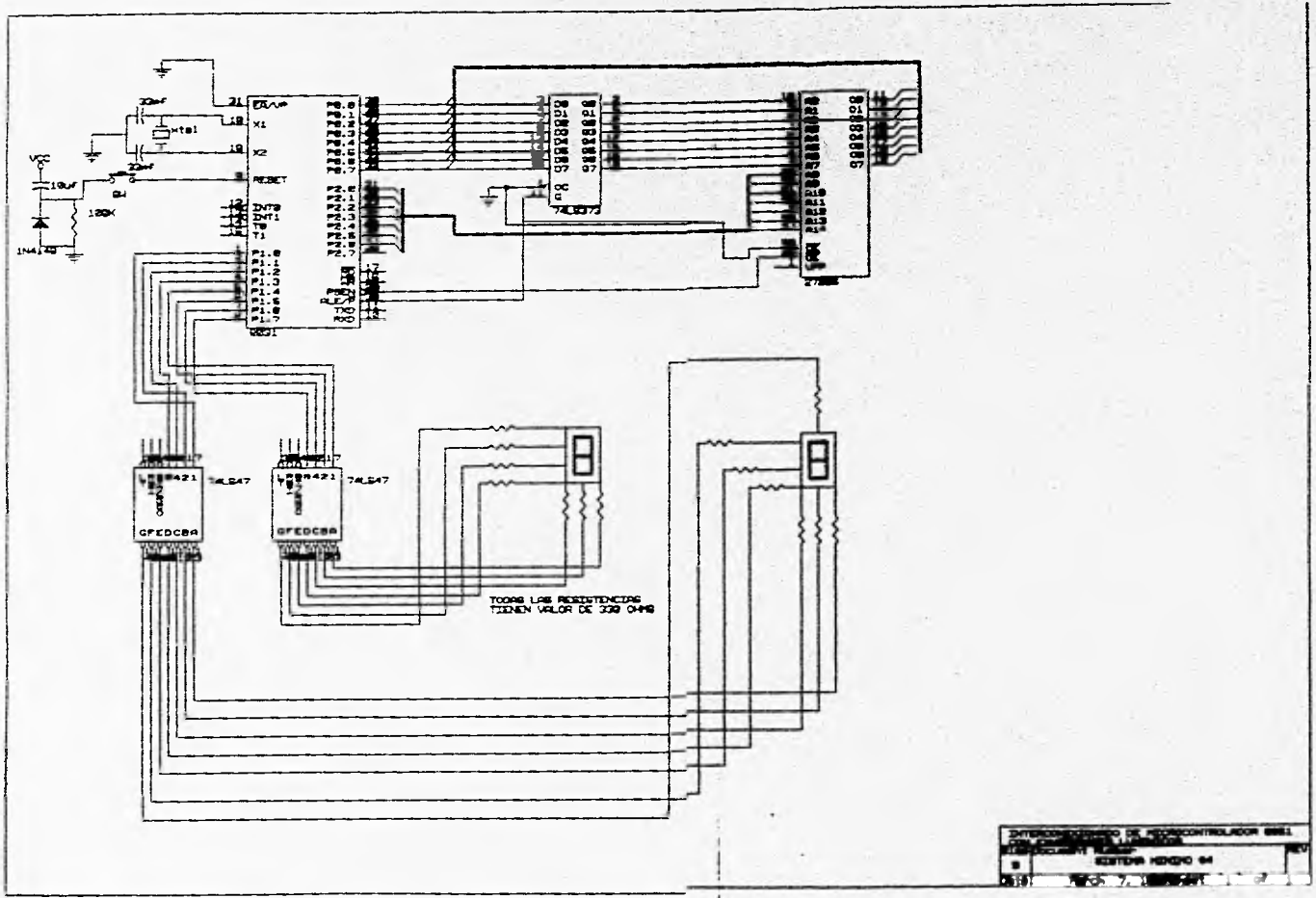
Objetivo:

Interconexionando el microcontrolador 8051 con exhibidores luminosos

En todos los sistemas digitales, un dispositivo que resulta esencial es el exhibidor luminoso pues permiten observar de manera fehaciente la operacion del microcontrolador, permitiendo asi que el operador humano sepa con certeza la condicion de operación del sistema mínimo.

Existen en el mercado una variedad amplia de tales exhibidores. La selección de tal o cual exhibidor va de acuerdo a las necesidades y requerimientos particulares de nuestro sistema propuesto.

El sistema que se muestra tiene como función el realizar la cuenta desde cero a 99. A continuación se muestra el diagrama y su programa respectivo. Es menester enfatizar que se recurre a muchas rutinas donde la principal tarea de éstas es adaptar el tiempo de operación del sistema al tiempo de los exhibidores, es decir para que la operación fuera apreciable por el operador humano; lo cual se logra haciendo que el microcontrolador gaste su tiempo haciendo disminuyendo valores numéricos dentro de sus registros de RAM internos.


74LS47
74LS47
TODAS LAS RESISTENCIAS
TIENEN VALOR DE 330 OHMS

PROGRAMA 4.

```
;*************************************************************
;UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
;FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN
;SISTEMA MINIMO 4
;CONTADOR DE 0 A 99 UTILIZANDO EXHIBIDORES
;*************************************************************


            ORG 00H
            AJMP WCTA
INICIO:     INC R0
            MOV A,B
            ANL A,R0
            ADD A,R3
            MOV P1,A    ;SE MANDA NUMERO DE UNIDAD
            JNB P1.3, TOTO
            JNB P1.0, TOTI
            MOV R0,#00H
            SJMP TOT
TOTO:       SJMP FIN
TOTI:       SJMP FIN
TOT:        NOP
            MOV R7,#0FFH     ;SE GENERA RUTINA DE TIEMPO
TIME:.      MOV R5,#0FFH
            MOV R6,#0FFH
YUI:        DJNZ R5,YUI
WAGE:       DJNZ R6,WAGE
            MOV R5,#0FFH
            MOV R6,#0FFH
REST:       DJNZ R5,REST
COME:       DJNZ R6,COME
            MOV R5,#0FFH
            MOV R6,#0FFH
GONE:       DJNZ R5,GONE
HEDO:       DJNZ R6,HEDO
            DJNZ R7,TIME
            INC R1
            MOV R2,A
            MOV A,B
            ANL A,R1
            SWAP A
            ADD A,R0
```

```
        MOV R3,A
        JNB P1.3, PORT
        JNB P1.0, PORT
        JNB P1.7, PORT
        JNB P1.4, PORT
        MOV P1,A
        SJMP WCTA
PORT:   MOV P1,A    ; SE ENVIA NUMERO DE DECENA.
FIN:    SJMP TIEMPO
WCTA:   MOV P1,#00H      ;SE MANDA CEROS AL PUERTO 1
        MOV B,#00001111B
        MOV R1,#00H      ;SE LIMPIA LOS REGISTROS R1,R2,R3 Y A
        MOV R2,#00H
        MOV R3,#00H
        MOV A,#00H
        MOV R7,#0FFH
TIEMPO: MOV R5,#0FFH     ;SE GENERAN LAS RUTINAS DE TIEMPO
        MOV R6,#0FFH
OPUS:   DJNZ R5,OPUS
POIS:   DJNZ R6,POIS
        MOV R5,#0FFH
        MOV R6,#0FFH
JOI:    DJNZ R5,JOI
UHI:    DJNZ R6,UHI
        MOV R5,#0FFH
        MOV R6,#0FFH
MNB:    DJNZ R5,MNB
BHU:    DJNZ R6,BHU
        DJNZ R7,TIEMPO
        LJMP INICIO      ;SALTO A SUBRUTINA CONTADORA.
        END
        END
```

3.7.-PRACTICA NUMERO V.-

Objetivo:

Impresión de datos mediante el microcontrolador 8051, haciendo uso de la interfase paralelo tipo centronics.

En muchos sistemas de adquisición y control de procesos es necesario tener por escrito una hoja de resultados, que nos indique los valores de las variables medidas o características del proceso, por ejemplo; si hubo algun artefacto o nivel sobrepasado, el tiempo en que se realizó tal o cual evento, el número de muestras, etc.

Por tal motivo una solución, es el envio de los resultados obtenidos por nuestro sistema, hacia una computadora, en la cual mediante un programa específico, se adecúe la información, para posteriormente ser enviada hacia una impresora obteniéndose la hoja deseada.

Desafortunadamente, el tiempo que se pierde enviar los datos hacia la computadora, aunado a la necesidad de tener una computadora libre cada vez que se requieran imprimir los resultados, hace de ésto, un medio tediosos y algunas veces molesto para los usuarios.

Por tal motivo, en este proyecto se propone interfasar directamente la impresora al sistema basado en el microcontrolador 8051.

CONNECTOR DB25

PROGRAMA 5.

```
;*************************************************************
;UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
;FACULTAD DE ESTUDIOS SUPERIORES DE CUAUTITLAN
;PROGRAMA PARA IMPRESION DE DATOS UTILIZANDO UN CONECTOR
;PARALELO TIPO CENTRONICS.
;UTILIZA CUATRO TERMINALES DE P1 Y UN FIJADOR 74374 COMO PERIFERICO
;DE SALIDA DE DATOS. DIR=C000H
;       P1.0=SELECTOR
;       P1.1=PAPER END(NO HAY PAPEL)
;      P1.2=BUSY(OCUPADO)
;       P1.3=STROBE(VALIDACION)
;       INTO=ACKNOWLEDGE(RECONOCIMIENTO)
;*************************************************************

            ORG 00H
            AJMP IMPRES
            ORG 03H
            SETB 20H.0
            MOVX @R0,A
            CLR P1.3    ;SE ESTABLECE EL STROBE
            SETB P1.3
            RETI

      ;*************************************************************
      ;ESTE PROGRAMA CONTROLARA LA IMPRESORA.
      ;PRIMERO SALTARA 4 LINEAS, DESPUES ESCRIBIRA:
      ;"BIENVENIDOS AL SISTEMA UNAM-FESC VOLVERA A SALTAR 2 LINEAS,
      ;AVANZARA UN ESPACIO(TAB) Y ESCRIBIRA :
      ;"PARA IMPRIMIR LOS CARACTERES HAY QUE ENVIAR SU CODIGO ASCII".
      ;POR ULTIMO SALTARA 2 LINEAS DARA UNA ESPACIO(TAB) Y ESCRIBIRA:
      ;"123456789"


IMPRES:     SETB IE.7
            SETB IE.0
            MOV DPTR,#TABLA
            MOV P2,#0C0H
            ACALL LF
            ACALL LF
            ACALL HT
```

```
        ACALL IPRE
        ACALL LF
        ACALL LF
        MOV DPTR,#PRLE
        ACALL HT
        ACALL IPRE
        ACALL LF
        ACALL LF
        ACALL HT
        MOV DPTR,#SELE
        ACALL IPRE
        ACALL LF
        ACALL LF
        ACALL HT
        MOV DPTR,#NUMER
        ACALL IPRE
TERMIN: SJMP TERMIN
LF:     MOV A,#0DH
        MOVX @R0,A
        CLR P1.3    ;SE ESTABLECE EL STROBE
        SETB P1.3
        MOV A,#0AH
OCUPO:  JB P1.2,OCUPO    ;OCUPADA LA IMPRESORA
RECO:   JNB 20H.0,RECO   ;ESPERA RECONOCIMIENTO
        CLR 20H.0
        RET
HT:     MOV A,#0DH
        MOVX @R0,A
        CLR P1.3
        SETB P1.3
        MOV A,#09H
OCUP:   JB P1.2,OCUP     ;OCUPADA LA IMPRESORA
REC:    JNB 20H.0,REC    ;ESPERA RECONOCIMIENTO
        CLR 20H.0
        RET

IPRE:   MOVX A,@DPTR
        JZ FIN
CARGA:  MOVX @R0,A
        CLR P1.3    ;SE ESTABLECE EL STROBE
        SETB P1.3
ESPLIN: JNB P1.0,ESPLIN  ;ESPERA QUE SE PONGA
```

```
                    ;EN LINEA LA IMPRESORA

NHPAP:      JB P1.1,NHPAP   ; NO HAY PAPEL
OCUPA:      JB P1.2,OCUPA   ;OCUPADA LA IMPRESORA
RECO:       JNB 20H.0,RECO  ;ESPERA RECONOCIMIENTO
            CLR 20H.0
            INC DPTR
            MOVX A,@DPTR
            JNZ ESPLIN
FIN:        RET

            ORG 1000H
TABLA:      DB 'BIENVENIDOS AL SISTEMA UNAM/FESC'
            DB 00H
PRLE:       DB 'ESTA ES UNA PRUEBA DE IMPRESION'
            DB 00H
SELE:       DB 'PARA IMPRIMIR LOS CARACTERES HAY'
            DB 'QUE ENVIAR SU CODIGO ASCII'
            DB 00H
NUMER:      DB 30H
            DB 31H
            DB 32H
            DB 33H
            DB 34H
            DB 35H
            DB 36H
            DB 37H
            DB 38H
            DB 39H
            DB 00H
            END
            END
```

APENDICE A

APENDICE A.

COMO GRABAR MEMORIAS EEPROM UTILIZANDO EL GRABADOR DE MEMORIAS ROM MASTER .

(PARA LLEVAR A CABO ESTA OPERACION SE RECOMIENDA SEGUIR TODOS LOS PROCEDIMIENTOS SUGERIDOS PARA EVITAR DAÑOS EN LAS MEMORIAS EEPROM DEBIDO A LA PRESENCIA DE ELECTRICIDAD ESTATICA).

1. Asegurarse de que se posee el arrhivo de la información que se quiere grabar el forma binaria , es decir que el archivo tenga terminación .bin.
Este formato se obtiene utilizando un programa que intercambia del formato hexadecimal al formato binario, llamado hexbin. exe.
(dentro de las opciones de este programa se utiliza el formato de INTEL).

2. Para entrar al programa Rom Master estando en el directorio raiz se teclea cd rom master oprimir enter, después teclear cd bin y oprimir enter. Teclear rom master,oprimir enter y esperar despliegado del programa.

3. Se podrá observar una serie de opciones en la parte superior de la pantalla , mediante el cursor posicionarse en la opción de manufacturer. Oprimir enter . La computadora desplegará en el monitor el nombre del fabricante que tenga seleccionado con anterioridad. Si el fabricante no corresponde a la memoria eeprom que se va a grabar se borra dicho nombre y se oprime enter. Se desplegara un conjunto de nombres de fabricantes que se encuentran instalados en la memoria de la computadora de donde se escogerá el tipo de fabricante que corresponda al tipo de memoria que se desee usar.

4. A continuación la computadora presentara el tipo de memoria anteriormente preseleccionado, si el tipo de memoria no corresponde al tipo de memoria que se desea grabar , se procede a borrar el tipo de memoria y presionar enter . Se desplegará un conjunto de diferentes tipos de memoria de los cuales se escogerá el tipo de memoria en cuestión.

5. Presionar escape una vez para regresar al menú de opciones iniciales . Posicionarse mediante el cursor a la opción device y presionar enter.

A continuación se coloca la memoria en zócalo programador , teniendo cuidado en el sentido de la ranura de la memoria, (nocht) . Bajar la palanca de seguridad del zócalo programador para que la memoria quede firmemente asentada .

6. En la pantalla del monitor se verá que el cursor esta posicionado en la primera de un conjunto de opciones (program), mediante el cursor posicionarse en la opción de BLANK CHECK y presionar enter. A continuación se vera que la computadora ejecuta una serie de operaciones y se observa que el led del zócalo programador se enciende indicando un status de ocupado (busy) . Si la memoria en cuestión esta limpia de datos aparecerá el mensaje de succsess.

De lo contrario aparecerá el mensaje de failure, lo cual indica que la memoria necesita colocarse en el borrador de eeprom's o que esta inservible.

7. Oprimir escape , ir a la opción denominada file . Ir a la opción load y después posicionarse en la opción de load,presionar enter y volverlo a presionar para posicionarse en la opción A:Binary file. Presionar enter, colocarse en la opción File name, presionar enter e introducir el lugar y nombre del archivo que contiene el programa que va a ejecutar el microcontrolador.Cuando se finalice de introducir

dicho nombre, presionar enter.

8. A continuación, posicionarse en la opción de Initiate load con lo cual el archivo que contiene el programa que va a correr en el microcontrolador se posiciona en el Buffer del programador y se encontrara listo para ser insertado-programado en la memoria eeprom.

9.Presionar F5 para regresar a la opción de Program, y una vez en dicho lugar se presiona enter para comenzar la programación de la memoria eeprom.Una vez finalizado el proceso seguramente se obtendrá un status de success en a Operación consecuencia se tendrá una memoria eeprom lista para ser utilizada en la aplicación.

APENDICE B

APENDICE B

COMO SIMULAR PROGRAMAS PARA EL SIMULADOR DEL MICROCONTROLADOR 8031

Supóngase que se cuenta con un programa llamado BETA.ASM(es importante mencionar que los programas para el microcontrolador que se utiliza, los programas deberán siempre tener terminación ASM.)

1.Se procede a ensamblar el programa BETA.ASM mediante el comando xasm51 BETA.ASM,(asumiendo que el programa beta está en el mismo subdirectorio que xasm51.exe).Este proceso generara los archivos BETA.PRN donde se especifican el código objeto y los errores, y el archivo BETA.HEX con el opcode del programa.

Al terminar el proceso, el ensamblado no se podrá continuar si existen errores, los cuales son señalados al final del proceso.En tal caso habrá que regresar al editor de textos en formato ASCII que se utilizó para generar el texto del programa y corregir los errores.

2.Para realizar la simulación del programa se teclea el comando AVSIM51, donde el programa pedirá una de las opciones de los diferentes microcontroladores que maneja, escogiéndose la opción C; la cual indica que se utiliza el 8031.

3.Una vez dentro del programa se realiza lo siguiente:

"Cargar el programa de BETA.HEX con el comando LOAD y PROGRAM.

"Definir el espacio de memoria a usar con el comando SET, MEMORY-MAP,ramdom Acces y asignando 00H y FFFH para la parte baja y alta respectivamente.

"Seleccionar la zona de memoria en la pantalla através de Dump Area y 1 indicando que es memoria Absolute y finalmente tecleando la dirección de inicio en forma hexadecimal.

"Ir a la pantalla a cambiar el PC a la dirección de inicio del programa.

Para correr el programa instrucción por instrucción se utiliza la tecla de F10 y para regresar un paso se usa F9. Para cancelar un comando se usa CTRL-C.

MANUAL DE CONJUNTO DE INSTRUCCIONES DEL MCS - 51

Table 11. Instruction Opcodes in Hexadecimal Order

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

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Mnemonic	Operands	Hex Code	Number of Bytes	Mnemonic	Operands
XRL	A,@R0	99	1	SUBB	A,R1
XRL	A,@R1	9A	1	SUBB	A,R2
XRL	A,R0	9B	1	SUBB	A,R3
XRL	A,R1	9C	1	SUBB	A,R4
XRL	A,R2	9D	1	SUBB	A,R5
XRL	A,R3	9E	1	SUBB	A,R6
XRL	A,R4	9F	1	SUBB	A,R7
XRL	A,R5	A0	2	ORL	C,/bit addr
XRL	A,R6	A1	2	AJMP	code addr
XRL	A,R7	A2	2	MOV	C,bit addr
JNZ	code addr	A3	1	INC	DPTR
ACALL	code addr	A4	1	MUL	AB
ORL	C,bit addr	A5		reserved	
JMP	@A + DPTR	A6	2	MOV	@R0,data addr
MOV	A, # data	A7	2	MOV	@R1,data addr
MO	a addr, # data	A8	2	MOV	R0,data addr
	# data	A9	2	MOV	R1,data addr
	, # data	AA	2	MOV	R2,data addr
MO	, # data	AB	2	MOV	R3,data addr
MOV	R1, # data	AC	2	MOV	R4,data addr
MOV	R2, # data	AD	2	MOV	R5,data addr
MOV	R3, # data	AE	2	MOV	R6,data addr
MOV	R4, # data	AF	2	MOV	R7,data addr
MOV	R5, # data	B0	2	ANL	C,/bit addr
MOV	R6, # data	B1	2	ACALL	code addr
MOV	R7, # data	B2	2	CPL	bit addr
SJMP	code addr	B3	1	CPL	C
AJMP	code addr	B4	3	CJNE	A, # data,code addr
ANL	C,bit addr	B5	3	CJNE	A,data addr,code addr
MOVC	A,@A + PC	B6	3	CJNE	@R0, # data,code addr
DIV	AB	B7	3	CJNE	@R1, # data,code addr
MOV	data addr, data addr	B8	3	CJNE	R0, # data,code addr
MOV	data addr,@R0	B9	3	CJNE	R1, # data,code addr
MOV	data addr,@R1	BA	3	CJNE	R2, # data,code addr
MOV	data addr,R0	BB	3	CJNE	R3, # data,code addr
MOV	data addr,R1	BC	3	CJNE	R4, # data,code addr
MOV	data addr,R2	BD	3	CJNE	R5, # data,code addr
MOV	data addr,R3	BE	3	CJNE	R6, # data,code addr
MOV	data addr,R4	BF	3	CJNE	R7, # data,code addr
MOV	data addr,R5	C0	2	PUSH	data addr
MOV	data addr,R6	C1	2	AJMP	code addr
MOV	data addr,R7	C2	2	CLR	bit addr
MOV	DPTR, # data	C3	1	CLR	C
ACALL	code addr	C4	1	SWAP	A
MOV	bit addr,C	C5	2	XCH	A,data addr
MOVC	A,@A + DPTR	C6	1	XCH	A,@R0
SUBB	A, # data	C7	1	XCH	A,@R1
SUBB	A,data addr	C8	1	XCH	A,R0
SUBB	A,@R0	C9	1	XCH	A,R1
SUBB	A,@R1	CA	1	XCH	A,R2
SUBB	A,R0	CB	1	XCH	A,R3

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

INSTRUCTION DEFINITIONS

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345 H. After executing the instruction

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL

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

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A) ← (A) + (direct)

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
(A) ← (A) + ((Ri))

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A) ← (A) + #data

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC
(A) ← (A) + (C) + (R_n)

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC
(A) ← (A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC
(A) ← (A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC
(A) ← (A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP
(PC) ← (PC) + 2
(PC$_{10-0}$) ← page address

ANL <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL P1,#01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	r r r r

Operation: ANL
(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL

(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL

(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(direct) ← (direct) ∧ (A)

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct) ← (direct) ∧ #data

ANL C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag, otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC.7 = 1, and OV = 0:

```
MOV  C,P1.0     ;LOAD CARRY WITH INPUT PIN STATE
ANL  C,ACC.7    ;AND CARRY WITH ACCUM. BIT 7
ANL  C,/OV      ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL
(C) ← (C) ∧ ¬(bit)

CJNE <dest-byte>,<src-byte>,rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
        CJNE  R7,#60H, NOT_EQ
        ...   ...          ; R7 = 60H.
NOT_EQ: JC    REQ_LOW      ; IF R7 < 60H.
        ...   ...          ;
```

sets the carry flag and branches to the instruction at [illegible]. By testing the carry flag, this instruction determines whether R7 is greater or le[illegible]

If the data being presented to Port 1 is also 34H, then the instruction,

```
WAIT: CJNE  A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (A) <> (direct)
THEN
        (PC) ← (PC) + relative offset

IF (A) < (direct)
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE A, # data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (A) <> data
THEN
        (PC) ← (PC) + relative offset

IF (A) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE Rn, # data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (Rn) <> data
THEN
        (PC) ← (PC) + relative offset

IF (Rn) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE @Ri, # data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF ((Ri)) <> data
THEN
        (PC) ← (PC) + relative offset

IF ((Ri)) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 0 0

Operation: CLR
(A) ← 0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation: CLR
(C) ← 0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

CLR

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

```
CPL  A
```

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL
(A) ← ¬(A)

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

```
CPL  P1.1
CPL  P1.2
```

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation: CPL
(C) ← ¬(C)

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL
(bit) ← ¬(bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC  A,R3
DA    A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence.

```
ADD   A,#99H
DA    A
```

will leave the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 − 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA

```
-contents of Accumulator are BCD
IF    [[(A3-0) > 9] V [(AC) = 1]]
      THEN(A3-0) ← (A3-0) + 6
           AND

IF    [[(A7-4) > 9] V [(C) = 1]]
      THEN (A7-4) ← (A7-4) + 6
```

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC   @R0
DEC   R0
DEC   @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) − 1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) − 1

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct) ← (direct) − 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 X 18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV
$(A)_{15-8}$
$(B)_{7-0}$ ← (A)/(B)

DJNZ *<byte>,<rel-addr>*

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ  40H,LABEL_1
DJNZ  50H,LABEL_2
DJNZ  60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
          MOV     R2,#8
TOGGLE:   CPL     P1.7
          DJNZ    R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation: DJNZ

```
(PC) ← (PC) + 2
(Rn) ← (Rn) - 1
IF  (Rn) > 0 or (Rn) < 0
    THEN
          (PC) ← (PC) + rel
```

DJNZ direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation:
```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) − 1
IF (direct) > 0 or (direct) < 0
   THEN
        (PC) ← (PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC  @R0
INC  R0
INC  @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation:
```
INC
(A) ← (A) + 1
```

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct) ← (direct) + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC  DPTR
INC  DPTR
INC  DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

```
JB   P1.2,LABEL1

JB   ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation: JB

```
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
        (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC  ACC.3,LABEL1
JBC  ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation: JBC
(PC) ← (PC) + 3
IF (bit) = 1
THEN
(bit) ← 0
(PC) ← (PC) + rel

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL 2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation: JC
(PC) ← (PC) + 2
IF (C) = 1
THEN
(PC) ← (PC) + rel

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB  P1.3,LABEL1
JNB  ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF  (bit) = 0
    THEN (PC) ← (PC) + rel
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC  LABEL1
CPL  C
JNC  LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC) + 2
IF  (C) = 0
    THEN  (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   LABEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. addre.

Operation:

```
JNZ
(PC) ← (PC) + 2
IF    (A) ≠ 0
      THEN  (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC A
JZ    LABEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

```
JZ
(PC) ← (PC) + 2
IF    (A) = 0
      THEN  (PC) ← (PC) + rel
```

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

```
LCALL   SUBRTN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LCALL

(PC) ← (PC) + 3
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

```
LJMP   JMPADR
```

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV  R0,#30H   ;R0 <= 30H
MOV  A,@R0     ;A <= 40H
MOV  R1,A      ;R1 <= 40H
MOV  B,@R1     ;B <= 10H
MOV  @R1,P1    ;RAM (40H) <= 0CAH
MOV  P2,P1     ;P2 #0CAH
```

leaves the value 30H in R0, 40H in the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) in location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

*MOV A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A) ← (direct)

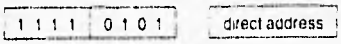
MOV A,ACC is not a valid instruction.

MOV direct,A

Bytes: 2

Cycles: 1

Encoding:



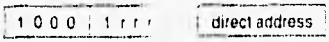
Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding:



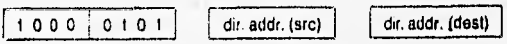
Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding:



Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2

Cycles: 2

Encoding:

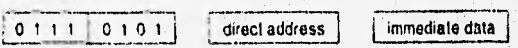
Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:



Operation: MOV
(direct) ← #data

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← #data

MOV @RI,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @RI,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((RI)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV   P1.3,C
MOV   C,P3.3
MOV   P1.2,C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 0 0 0 1 0	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding:

1 0 0 1 0 0 1 0	bit address

Operation: MOV
(bit) ← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR,#1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	immed. data15-8	immed. data7-0

Operation: MOV
(DPTR) ← $\#data_{15-0}$
DPH ☐ DPL ← $\#data_{15-8}$ ☐ $\#data_{7-0}$

MOVC A,@A + <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC   A
        MOVC  A,@A + PC
        RET
        DB    66H
        DB    77H
        DB    88H
        DB    99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A + DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A + PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVX <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 805' Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX  A,@R1

MOVX  @R0,A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR) ← (A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

```
MUL   AB
```

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL
$(A)_{7-0} \leftarrow (A) \times (B)$
$(B)_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP
$(PC) \leftarrow (PC) + 1$

ORL <dest-byte>,<src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL A,R0

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte. The mask byte may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

ORL P1,#00110010B

will set bits 5, 4, and 1 of output Port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) ∨ (Rn)

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

Operation: ORL
(A) ← (A) V (direct)

ORL A,@RI

Bytes: 1

Cycles: 1

Encoding:

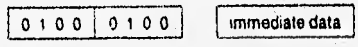
Operation:

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:



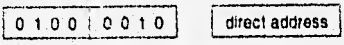
Operation: ORL
(A) ← (A) V #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding:



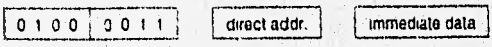
Operation: ORL
(direct) ← (direct) V (A)

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:



Operation: ORL
(direct) ← (direct) V #data

ORL C,< src-bit >

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```
MOV   C,P1.0     ;LOAD CARRY WITH INPUT PIN P10
ORL   C,ACC.7    ;OR CARRY WITH THE ACC. BIT 7
ORL   C,/OV      ;OR CARRY WITH THE INVERSE OF OV.
```

ORL C,bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL
(C) ← (C) V (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL
(C) ← (C) V $(\overline{bit})$

POP direct

Function: Pop from stack.

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP   DPH
POP   DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP   SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation:
POP
(direct) ← ((SP))
(SP) ← (SP) − 1

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH   DPL
PUSH   DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation:
PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

RET

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

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

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 1	0 0 1 0

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

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
RL   A
```

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 0 1 1

Operation: RL
$(A_{n+1}) \leftarrow (An)$ $n = 0 - 6$
$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

```
RLC   A
```

leaves the Accumulator holding the value 8BH (10001010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 0 1 1

Operation: RLC
$(An + 1) \leftarrow (An)$ $n = 0 - 6$
$(A0) \leftarrow (C)$
$(C) \leftarrow (A7)$

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
RR  A
```

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 1 1

Operation: RR
(An) ← (A_{n+1}) n = 0 - 6
(A7) ← (A0)

RRC A

Function: Rotate Accumulator Right through Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

```
RRC  A
```

leaves the Accumulator holding the value 62 (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 0 1 1

Operation: RRC
(An) ← (An + 1) n = 0 - 6
(A7) ← (C)
(C) ← (A0)

SETB <bit>

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB  C
SETB  P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation:
SETB
(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation:
SETB
(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB

(A) ← (A) − (C) − (Rn)

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) − (C) − (direct)

SUBB A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) − (C) − ((Ri))

SUBB A,#data

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB
(A) ← (A) − (C) − #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

```
XCH   A,@R0
```

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation:
XCH
(A) ⇄ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation:
XCH
(A) ⇄ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation:
XCH
(A) ⇄ ((Ri))

XCHD A,@Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

```
XCHD  A,@R0
```

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.

Bytes: 1

Cycles: 1

1 1 0 1	0 1 1 i

Encoding:

Operation: XCHD
$(A_{3-0}) \rightleftarrows ((Ri_{3-0}))$

XRL <dest-byte>,<src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
XRL  A,R0
```

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

```
XRL  P1,#00110001B
```

will complement bits 5, 4, and 0 of output Port 1.

DESCRIPCION DEL CONJUNTO DE INSTRUCCIONES DEL MCS-51

OPERACIONES ARITMETICAS.

ADD A,Rn	SUMAR UN REGISTRO AL ACUMULADOR.
ADD A.direct	SUMAR BYTE CON ACUMULADOR.
ADD A@RiS	SUMAR RAM INDIRECTA AL ACUMULADOR.
ADD A,#data	SUMAR DATO INMEDIATO AL ACUMULADOR.
ADDC A,Rn	SUMAR A UN REGISTRO CON ACARREO.
ADDC A,direct	SUMAR BYTE DIRECTO A REGISTRO A CON BANDERA DE ACARREO.
ADDC A,@Ri	SUMAR RAM INDIRECTA A REGISTRO A CON BANDERA DE ACARREO.
ADDC A,#data	SUMAR DATO INMEDIATO CON REGISTRO A CON BANDERA DE ACARREO.
SUBB A,Rn	SUBSTRAER REGISTRO DESDE A CON PRESTAMO.
SUBB A,direct	SUBSTRAER BYTE DIRECTO DESDE A CON PRESTAMO.
SUBB A,@Ri	SUBSTRAER RAM INDIRECTO DESDE A CON PRESTAMO.
SUBB A,#data	SUBSTRAER DATO INMEDIATO DESDE A CON PRESTAMO.
INC A	INCREMENTO DEL ACUMULADOR.
INC Rn	INCREMENTO DEL REGISTRO.
INC direct	INCREMENTO DE BYTE DIRECTO.
INC @Ri	INCREMENTO DE RAM INDIRECTO.

DEC AD	DECREMENTO DEL ACUMULADOR.
DEC Rn	DECREMENTO DEL REGISTRO.
DEC direct	DECREMENTO DE BYTE DIRECTO.
DEC @Ri	DECREMENTO DE RAM INDIRECTO.
INC DPTR	INCREMENTO DEL APUNTADOR DE DATOS.
MUL AB	MULTIPLICA LOS CONTENIDOS DEL REGISTRO A POR LOS DEL B.
DIV AB	DIVIDE A POR B.
DA A	AJUSTE DECIMAL DE ACUMULADOR.
OPERACIONES LOGICAS	
ANL A,Rn	FUNCION AND ENTRE REGISTRO Y ACUMULADOR.
ANL A,direct	FUNCION AND ENTRE BYTE DIRECTO Y ACUMULADOR.
ANL A@Ri	FUNCION AND ENTRE RAM INDIRECTA Y ACUMULADOR.
ANL A#data	FUNCION AND ENTRE DATA INMEDIATO Y ACUMULADOR.
ANL direct,A	FUNCION AND ENTRE ACUMULADOR Y BYTE DIRECTO.
ANL direct,#data	FUNCION AND ENTRE DATO INMEDIATO Y BYTE DIRECTO.
ORL A,Rn	FUNCION OR ENTRE REGISTRO Y ACUMULADOR.
ORL A,direct	FUNCION OR ENTRE BYTE DIRECTO Y ACUMULADOR.

ORL A,@Ri	FUNCION OR ENTRE RAM INDIRECTO Y ACUMULADOR.
ORL A,#data	FUNCION OR ENTRE DATO INMEDIATO Y ACUMULADOR.
ORL direct,A	FUNCION OR ENTRE ACUMULADOR Y BYTE DIRECTO.
ORL direct,#data	FUNCION OR ENTRE DATO INMEDIATO Y BYTE DIRECTO.
XRL A,Rn	FUNCION OR EXCLUSIVA ENTRE REGISTRO Y ACUMULADOR.
XRL A,direct	FUNCION OR EXCLUSIVA ENTRE BYTE DIRECTO Y ACUMULADOR.
XRL A,@Ri	FUNCION OR EXCLUSIVA ENTRE RAM INDIRECTA Y A.
XRL A,#data	FUNCION OR EXCLUSIVA ENTRE DATO INMEDIATO Y A.
XRL direct,A	FUNCION OR EXCLUSIVA ENTRE ACUMULADOR Y BYTE DIRECTO.
XRL direct,#data	FUNCION OR EXCLUSIVA ENTRE DATO INMEDIATO Y DIRECTO.
CLR A	BORRAR CONTENIDO DEL ACUMULADOR.
CPL A	COMPLEMENTAR ACUMULADOR.
RL R	ROTACION DEL ACUMULADOR A LA IZQUIERDA.
RLC A	ROTACION A LA IZQUIERDA ATRAVEZ DE LA BANDERA DE ACARREO.
RR A	ROTACION DEL ACUMULADOR A LA DERECHA.

RRC A	ROTACION DERECHA DE A,ATRAVES DE LA BANDERA DE ACARREO.
SWAP A	CAMBIAR LOS NIBLES DENTRO DEL ACUMULADOR.

TRANSFERENCIA DE DATOS.

MOV A,Rn	MOVER EL REGISTRO AL ACUMULADOR.
MOV A,direct	MOVER EL BYTE DIRECTO AL ACUMULADOR.
MOV A,@Ri	MOVER RAM INDIRECTA AL ACUMULADOR.
MOV A,#data	MOVER DATO INMEDIATO AL ACUMULADOR.
MOV Rn,A	MOVER EL ACUMULADOR AL REGISTRO.
MOV Rn,direct	MOVER BYTE DIRECTO AL REGISTRO.
MOV Rn,#data	MOVER DATO INMEDIATO AL REGISTRO.
MOV direct,A	MOVER EL ACUMULADOR AL BYTE
MOV direct,Rn	MOVER EL REGISTRO AL BYTE DIRECTO.
MOV direct,direct	MOVER EL BYTE DIRECTO AL DIRECTO.
MOV direct,@RiS	MOVER RAM INDIRECTA AL BYTE DIRECTO.
MOV direct,#data	MOVER DATO INMEDIATO AL BYTE DIRECTO.
MOV @Ri,A	MOVER EL ACUMULADOR AL RAM INDIRECTA.
MOV @Ri,direct	MOVER BYTE DIRECTO AL RAM INDIRECTO.
MOV @Ri#data	MOVER DATO INMEDIATO AL RAM INDIRECTO.
MOV DPTR,#data16	CARGA EL APUNTADOR DE DATOS CON UNA

	CONSTANTE DE 16BITS.
MOVC A,@A+DPTR	MOVER EL CODIGO DEL BYTE RELATIVO AL DPTR Y AL A.
MOVC A,@A+PC	MOVER EL CODIGO DEL BYTE RELATIVO AL PC Y AL A.
MOVX A,@Ri	MOVER EL RAM EXTERNO(8 BITS DE DIRECCION) AL A.
MOVX A,@DPTR	MOVER EL RAM EXTERNO (16 BITS DE DIRECCION) AL A.
MOVX @Ri,A	MOVER A AL RAM EXTERNO(8 BITS DE DIRECCION).
MOVX @DPTR,A	MOVER A AL RAM EXTERNO (16 BITS DE DIRECCION).
PUSH direct	INTRODUCIR BYTE DIRECTO DENTO DE LA PILA(STACK)
POP direct	SACA EL BYTE DIRECTO DE LA PILA (STACK).
XCH A,Rn	CAMBIAR CONTENIDO DE REGISTRO CON ACUMULADOR.
XCH A,direct	CAMBIAR BYTE DIRECTO CON ACUMULADOR.
XCH A,@Ri	CAMBIAR RAM INDIRECTO CON A.
0XCHD A,@Ri	CAMBIAR DIGITO DE BAJO ORDEN DE RAM. CON A.

MANIPULACION DE VARIABLES BOOLEANAS.

CLR C	LIMPIAR BANDERA DE ACARREO.
CLR bit	LIMPIAR BIT DIRECTO.
SETB C	CONFIGURAR LA BANDERA DE ACARREO.

SETB bit	CONFIGURAR BIT DIRECTO.
CPL C	COMPLEMENTAR LA BANDERA DE ACARREO.
ANL C,bit	FUNCION AND ENTRE BIT DIRECTO Y BANDERA DE ACAREO.
ANL C, bit	FUNCION AND ENTRE BIT DIRECTO Y ACARREO.
ORL C,bit	FUNCION OR ENTRE BIT DIRECTO Y BANDERA DE ACARREO.
ORL C, bit	COMPLEMENTO OR ENTRE BIT DIRECTO Y ACARREO.
MOV C,bit	MOVER BIT DIRECTO A BANDERA DE ACARREO.
MOV bit,C	MOVER BANDERA DE ACARREO A BIT DIRECTO.

CONTROL DE PROGRAMA Y MAQUINA

ACALL addrll	LLAMADA ABSOLUTA SUBORDINADA.
LCALL addr16	LLAMADA LARGA SUBORDINADA.
RET	REGRESO DE SUBORDINADO.
RETI	REGRESO DE INTERRUPCION.
AJMP addrll	SALTO ABSOLUTO.
IJMP addr16	SALTO LARGO.
SJMP rel	SALTO CORTO(EN DIRECCION RELATIVA).
JMP @A+DPTR	SALTO RELATIVO INDERECTO A EL DPTR.
JZ rel	SALTO SI EL ACUMULADOR ES CERO.
JNZ rel	SALTO SI EL ACUMULDOR NO ES CERO.
JC rel	SALTO SI LA BANDERA DE ACCARREO ES

	ESTABLECIDA.
JNC rel	SALTO SI NO HAY BANDERA DE ACARREO.
JB bit,rel	SALTO SI BIT DIRECTO ES CONFIGURADO.
JNB bit,rel	SALTO SI BIT DIRECTO NO ES CONFIGURADO.
JBC bit,rel	SALTO SI BIT DIRECTO ES CONFIGURADO Y BORRADO.
CJNE A,direct,rel	COMPARACION DIRECTA CON REGISTRO A Y SALTO SI NO SON IGUALES.
CJNE A#data,rel	COMPARACION INDIRECTA CON REGISTRO A Y SALTO SI NO SON IGUALES.
CJNE Rn,#data,rel	COMPARACION INDIRECTA CON REGISTRO Y SALTO SI NO SON IGUALES.

NOTAS RESPECTO AL MODO DE DIRECCIONAMIENTO DE DATOS:

Rn-REGISTRO DE TRABAJO (R0-R7).

direct-CUALESQUIER DE LAS 128 LOCALIDADES DE RAM INTERNA, CUALESQUIER PUERTO DE ENTRADA Y SALIDA, REGISTRO DE CONTROL O DE STATUS.

@Ri-LOCALIDAD DE RAM INTERNA INDIRECTA DIRECCIONADA POR EL REGISTRO R0 O R1.

#data-CONSTANTE DE OCHO BITS INCLUIDA EN ALGUNA INSTRUCION.

#data16-CONSTANTE DE 16 BITS INCLUIDAS EN LAS INSTRUCCIONES DE 2&3 BYTES DE LARGO.

bit-CUALQUIERA DE LAS 128 BANDERAS, CUALQUIER PIN DE ENTRADA O

SALIDA O CUALQUIER BIT DE CONTROL O DE STATUS.

NOTAS RESPECTO A LOS MODOS DE DIRECCIONAMIENTO DE PROGRAMA.

addr16-DIRECCION DESTINO PARA LAS INSTRUCCIONES LCALL Y LJMP LA CUAL PUEDE ESTAR EN CUALQUIER LUGAR DENTRO DEL ESPACIO DE DIRECCIONES DEL PROGRAMA DE MEMORIA.

addr11-DIRECCION DESTINO PARA LAS INSTRUCCIONES ACALL Y AJMP QUE ESTARA DENTRO DE LOS MISMOS 2 KILOBYTES DE MEMORIA DE PROGRAMA COMO EL PRIMER BYTE DE LA SIGUIENTE INSTRUCCION.

rel-LA INSTRUCCION SJMP Y TODOS LOS SALTOS CONDICIONALES INCLUYEN UN OFFSET DE 8 BITS.

Bibliografía:

1.-Lógica digital y diseño de computadores.
Morris Mano
Prentice Hall

2.-Sistemas digitales
Tocci
Prentice hall

3.-Circuitos electrónicos discretos e integrados.
Schilling-Belove
Alfaomega.

4.-Embedded applications
Intel
1991

5.'Peripherials
Intel
1990

6.-Memory
Intel
1990

7.'Fast and ls ttl data
q3/92 dl121 rev 5
Motorola.