UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL BUENOS AIRES

Departamento de Electrónica

Cátedra: Técnicas Digitales II - Plan 1995 – Versión 2002. – Edición Previa

CAPITULO 9: Microcontroladores – Parte 1

Autores: Ings. Juan Montenegro y Marcelo Romeo

1. Introducción

1.1 Reseña histórica

En 1978, Intel produjo la primera microcomputadora en un solo circuito (single chip microcomputer). La misma contenía en un encapsulado de 40 patas un rudimentario microprocesador orientado a manipulaciones de bits (privilegiando el manejo de variables de este tipo frente a complejas operaciones de bytes), 27 líneas de entrada / salida, 64 bytes de memoria de datos y 1 kbyte de memoria de programa (ROM para la versión 8048, EPROM para el 8748) y un temporizador de 8 bits.

Esta microcomputadora estaba dirigida al mercado de los sistemas de control en los que se suelen manejar entradas y salidas binarias y se optimizaron las operaciones booleanas entre las mismas.

Debido a su razonable costo, comenzó a desplazar a sistemas electromecánicos con mayores prestaciones, convirtiéndose rápidamente en un estándar en su tipo.

Intel y National presentaron algunas variantes al 8048 como el 8049 con 2 kbytes de ROM y 128 bytes de memoria de datos y el 8050AH con 4 kbytes de ROM y 256 bytes de memoria de lectura / escritura.

En 1980, Intel presentó una versión mejorada del 8048 denominada 8051[1] al cual le incorporó más memoria de programa y de datos, un segundo temporizador con mayor versatilidad, una puerta serie asincrónica e importantes mejoras en su repertorio de instrucciones y arquitectura de registros.

Fue tan importante su penetración en el mercado que continúa siendo el núcleo de los microcontroladores actuales. Podrán disponer de mayor memoria de datos o de programas (con distintas tecnologías) o que se le agreguen más instrucciones o registros, temporizadores con mayores funciones, pero siempre bajo la estructura básica del 8051.

Como las herramientas de desarrollo (compiladores, simuladores, etc.) pueden ser de dominio público (gratuitos) o muy populares a moderado costo (Franklin, Keil) y siguen siendo adaptables a las versiones actuales de microcontroladores, es que en este curso se estudiará el 8051 básico y al concluir el capítulo se hará una presentación de las distintas alternativas comerciales actuales.

[1] Comenzó a denominar a estas microcomputadoras en un solo circuito, microcontroladores.

1.2 Nomenclatura.

Otro punto para tener en cuenta es la notación de las señales activas a nivel bajo (negadas). Según un estándar actualmente difundido en la bibliografía de la materia, por ejemplo para referirnos a la señal que indica escritura de datos externa (Write) y es activa a nivel bajo lo anotaremos “/WR” ó “/WR” (se leerá WRITE negado).

2. Diferencias entre Microcontrolador (MCU) y Microprocesador (MPU).

2.1 Planteo

Para comenzar adentrándonos en el mundo de los microcontroladores (Microcontrolador, por ***Micro Controller Unit***) veremos las características y rasgos principales que los diferencian de los microprocesadores (MPU, por ***Micro Processor Unit***).

Los microprocesadores poseen los mecanismos para arbitrar el uso de por lo menos tres buses diferentes: bus de direcciones, bus de datos y bus de control. Gracias a estos buses, y dependiendo de la forma de manejo de los mapas de memoria (memoria de programa, memoria de datos, registros de entradas y salidas, etc., según algún modelo particular, (por ejemplo Von Neumann o Harvard) el microprocesador administra el flujo de datos entre el universo de dispositivos que pueden conectársele.

Los microprocesadores no poseen memoria interna de tamaño importante (únicamente para registros internos, colas de instrucciones y memorias caché). Tanto la memoria de programa, la de datos como los periféricos se encuentran conectados (externamente al microprocesador) a los buses mencionados anteriormente.

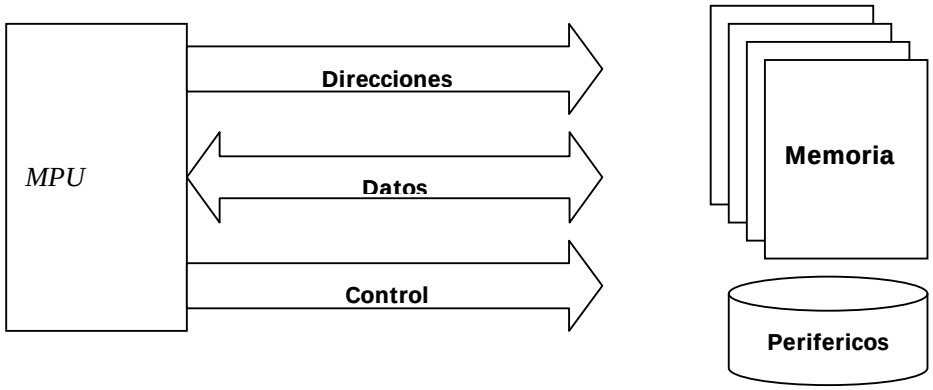


Fig. 1. Arquitectura de una microcomputadora genérica basada en un Microprocesador.

Para este caso simplificado suponemos que dentro del grupo de los periféricos se encuentran controladores y dispositivos diversos (discos, controladores de acceso directo a memoria, controladores de interrupciones, etc.).

Ahora bien, veremos que el panorama cambia para el caso de los microcontroladores, pues estos poseen en forma integrada (dependiendo del tipo y modelo): memoria de datos, de programa, temporizadores, puertos de entrada y salida, puertos serie, conversores A/D, D/A, etc.

Para lograr el ordenamiento del flujo de datos y sus señales de control los **microcontroladores** poseen buses internos y que no son accesibles desde el exterior. En general lo que encontraremos accesible desde el exterior serán puertos de entrada y salida, puertos serie y paralelo, etc.

En la **Fig. 2** podemos apreciar un bosquejo de la estructura interna de un microcontrolador genérico. Allí apreciamos la existencia de una CPU integrada con el resto de los periféricos, la cual posee registros, una unidad aritmética y lógica, decodificador de códigos de operación, etc.

Los puertos que llamamos de “Entrada / Salida” son para propósitos generales, y pueden configurarse a nivel de bit, es decir, se pueden usar 8 bits de salida o configurar individualmente los bits que conforman dicho puerto como entradas y/o salidas para funciones totalmente diferentes, de esta forma podemos vislumbrar la existencia de la que algunos tienden a denominar la filosofía del bit.

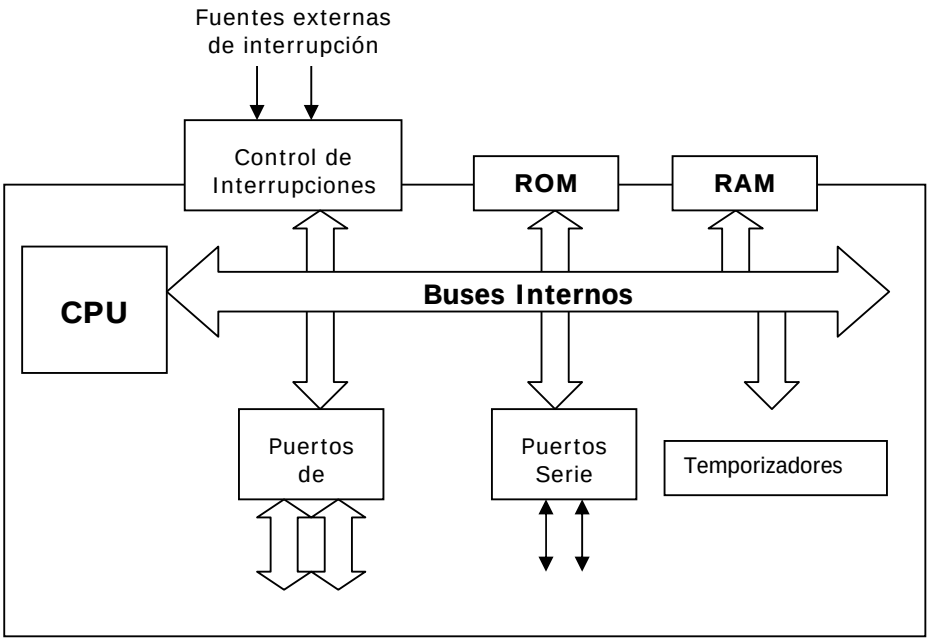


Fig. 2 Estructura interna de un Microcontrolador genérico

2.2 Filosofía del bit.

Podemos decir que en el mundo de los microprocesadores la unidad de información es el ancho de palabra (*Word Width*). Por ejemplo en un microprocesador 8086 es de dos bytes, en un 80486 es de cuatro bytes, etc. Esto implica que los registros internos del Microprocesador, la arquitectura de su bus de datos y los códigos de operación están preparados para manejar con cierta comodidad conjuntos de bits agrupados en "Words" o a lo sumo en bytes.

En los Microcontroladores en cambio, la unidad de manejo de datos es el bit. Encontraremos entonces instrucciones que nos permiten interactuar directamente con una pata determinado de un puerto de Entrada / Salida. También por lo general, los registros internos de la CPU del microcontrolador y algunas posiciones de memoria de datos interna son accesibles a nivel de bit, esto es, disponemos de un código de operación para cambiar el estado de un bit en particular de una posición de memoria de datos[2].

Esta característica particular de los Microcontroladores es llamada "direccionamiento a nivel de bit" (Bit Addressable), y de esta forma, a lo largo de este capítulo, nos referiremos con frecuencia a las operaciones a nivel de bit.

Pero esta "filosofía del bit" no solo implica el cambio de estado de bits pertenecientes a puertos o posiciones de memoria, sino que también existen códigos de operación para procesar a nivel lógico y aritmético determinados conjuntos de bits y lo que es muy importante, tomar decisiones en base a los resultados.

Por ejemplo si un determinado bit que representa el estado de un pata de una puerta configurada como entrada toma el valor "1" entonces una instrucción nos permitirá saltar a otra parte del programa que pondrá en "0" (o en "1") otro bit configurado como salida. En nemónicos:

```
            JB      ENT1, SACA_DATO   ; Si el bit ENT1 vale 1, se saltará a ejecutar el tramo
            ......                    ; de programa SACA_DATO
SACA_DATO:  CLR     BIT_SALIDA        ; Se pone en "0" el bit definido como BIT_SALIDA
```

[2] Por ese motivo, entre otros, se dice que algunas familias de microcontroladores incorporan en su interior un procesador booleano

Compárese la simpleza de este programa frente a lo que sería su equivalente para el 8088.

```
            MOV   DX,ENTRADA.
            IN    AL,DX              ; Lee un byte completo de entrada
            TEST  AL,ENT1            ; Aisla el bit que se desea analizar por medio de una máscara
            JNZ   SACA_DATO          ; Si el bit considerado vale "1", salta a SACA_DATO
            .....
SACA_DATO:  MOV   DX,SALIDA          ; Apuntamos a la puerta de salida
            IN    AL,DX              ; Se lee el estado de la puerta de salida
            OR    AL,BIT_SALIDA             ; Se pone en "1" el único bit que se desea cambiar
            OUT   DX,AL              ; Se lo escribe sobre la puerta de salida.
```

Es evidente que para la operación con variables del tipo bit, es mucho más sencillo operar con un microcontrolador que con un microprocesador. También debe destacarse que para las operaciones que requieran alguna complejidad matemática o lógica a nivel de bytes o palabras, es conveniente el empleo de un microprocesador.

2.3 Árbol genealógico y aplicaciones de los microcontroladores.

En este punto haremos una breve presentación de algunas familias de microcontroladores (las de uso más frecuente), algunas aplicaciones y algunas curiosidades.

2.3.1 Familias Intel:

Como planteamos en la introducción, podemos mencionar como referencia histórica, a la familia MCS®-48 compuesta principalmente por el 8048AH.

Los microcontroladores de esta familia poseen una CPU de 8 bits, 27 líneas de Entrada / Salida, un Temporizador / contador (*Timer/Counter*) programable. Dependiendo del modelo (8050,8049,8048) poseen respectivamente 256, 128 o 64 bytes de RAM estática. Existen en versiones ROM, EPROM, Flash y sin ROM (Romless). Esta última es un caso particularmente interesante y es una característica habitual de la mayoría de las familias. Alguno de los microcontroladores, no poseen memoria para almacenamiento de código ejecutable, entonces se reservan algunas líneas de Entrada / Salida para armar los tres buses necesarios (direcciones, datos y control), de esta forma el programa puede ser almacenado en cualquier memoria externa (ROM, EPROM, E^2PROM, FLASH, NVRAM, etc.) y accedido por estos buses.

Más recientemente encontramos a la familia MCS®-51, quizás la más prolífica en su momento, ya que un gran número de fabricantes de hardware y software implementaron sistemas sobre la base de esta familia de microcontroladores y una gran cantidad de herramientas de desarrollo como compiladores de lenguaje "C", simuladores, emuladores en tiempo real[3] ICE's (In Circuit Emulators), etc. Debido a que este apunte está orientado en particular al estudio de esta familia, no la trataremos en detalle por ahora, lo que podemos agregar es que no existe compatibilidad con su antecesor (MCS®-48) a nivel software o hardware, si bien existe algunos programas (bastante primitivos, por cierto) para convertir código de MCS®-48 a MCS®-51.

2.3.2 Familias Motorola:

Al igual que en el campo de los microprocesadores, existe una batalla comercial y un mercado que se reparten fundamentalmente entre Intel y Motorola en lo que a Microcontroladores se refiere. Motorola posee una familia de microcontroladores 6805 de bajo costo, muy popular y con el

[3] Estos dispositivos son sumamente útiles y costosos para el desarrollo de hardware y software. Consiste en un elaborado sistema de adquisición de datos digitales con memorias estáticas de alta velocidad que muestrean en cada flanco de reloj el estado de cada línea del microprocesador *en tiempo real.* Posteriormente se podrá mostrar el desensamblado del programa real ejecutado o como mapas de bits o como contenido de memoria. También se dispondrá de comparadores de alta velocidad que detendrán la operación (breakpoints) del microcontrolador al acceder a una determinada posición de memoria o puerto de E/S.
También el programa compilado en una PC se podrá cargar en una memoria RAM interna del emulador, detectar fallas o problemas en el programa y modificar manualmente algunos bytes del mismo sin necesidad de grabar una EPROM o FLASH ganando mucho tiempo en el proceso de depuración.

que también se han desarrollado infinidad de aplicaciones y herramientas, quizás con mas variedad de integrantes en su familia que los Intel.. Esta familia cuenta con versiones ROM, EPROM, E^2PROM y flash, con una particularidad muy interesante, poseen memoria en cantidades no convencionales (192, 768 bytes en lugar de cantidades más razonables como 64, 128 o 256 bytes). A diferencia de los Intel se los consigue en versiones de tamaño reducido (encapsulado DIP 16 por ejemplo), para aplicaciones donde no se requieren demasiadas líneas de Entrada / Salida.

Una de las particularidades de esta familia es que sus integrantes son autocontenidos, es decir que se deberán utilizar exclusivamente los recursos internos ya que no disponen de buses de expansión y por ende no pueden conectarse memorias ni de datos ni de programa externas.

También fueron precursores en la simplificación del proceso de grabado de las versiones flash o EPROM. Ya en las primeras hojas de datos de la familia 6805 se proponía un simple circuito (un par de transistores, algunos resistores y LEDs y un zócalo) que se encargaba de copiar una EPROM (grabada con un programador estándar) a la memoria EPROM interna del microcontrolador.

Otra familia importante y más reciente es la de los Microcontroladores 68HC11, razonablemente compatible con sus antecesores, incorporando a diferencia que sus hermanos menores, conversores A/D multiplexados, versiones OTP (One Time Programmable), mayor cantidad de memoria para programas y datos, circuito de Watchdog[4], más cantidad de temporizadores y más funcionales, etc.

Una de las características más destacables para los usuarios de Motorola es la *ortogonalidad de su repertorio de instrucciones*. Esto quiere decir que no existen registros dedicados (por ejemplo en Intel para hacer una operación de entrada / salida se debe emplear indefectiblemente el acumulador), por lo que el manejo del repertorio de instrucciones es mucho más sencillo pues no existen excepciones (como por ejemplo la inexistente intrucción MOV AL,byte ptr [DX]) a las reglas generales de direccionamiento o movimiento de datos.

2.3.3 Familias COP de National:

National Semiconductor posee unas familias de Microcontroladores de 4, 8 y 16 bits que denominaron COPS. Una característica interesante de estas familias es la utilización de un bus especial para periféricos compatibles con dicha familia (conversores varios, memorias no volátiles, etc.) llamado bus Microwire®.

2.3.4 Familias Microchip PIC.

La característica principal de esta familia es que el microprocesador de estos microcontroladores son tipo RISC (Reduced Instruction Set Computer) conteniendo algo más de treinta instrucciones simples y rápidas. Esto los hace mucho más rápidos y con un código más compacto (y repertorio de instrucciones más fácil de manejar) que los de tipo CISC (Complex Instruction Set Computer) pertenecientes a otros fabricantes.

Existe compatibilidad de código entre los distintos integrantes de la familia, teniendo como atractivo trascendente su muy bajo costo y disponibilidad de compiladores de libre disponibilidad así como programadores fácilmente armables por el usuario.

3. Primer análisis de la familia MCS®-51.

En este punto haremos una presentación introductoria de la familia y posteriormente haremos un estudio más detallado de alguno de los puntos aquí introducidos.

[4] Perro guardián o Cancerbero. Un monoestable redisparable que si no es pulsado en un tiempo predeterminado, resetea al procesador. Se emplea para evitar que el microprocesador quede en un estado “colgado” permanente.

3.1 Estructura interna del 8051.

Como mencionamos anteriormente, entraremos ahora en los detalles concernientes a las características de la familia MCS®-51 de Intel.

Estos Microcontroladores poseen un ciclo de máquina de seis estados, y estos a su vez de dos períodos de reloj, entonces utilizando un reloj de 12[5] MHz (habitual para las primeras generaciones de microcontroladores) dicho ciclo de máquina es de 1 μSeg. La mayoría de los códigos de operación se ejecutan en uno o dos ciclos de máquina.

La familia 8051 posee cuatro puertas bidireccionales de ocho bits, esto es, cualquier pata de cualquier puerta puede configurarse indistintamente como entrada o como salida. Esto nos daría 32 líneas de Entrada / Salida para utilizar como más nos convenga, pero esto también tiene un precio, como veremos más delante.

Si deseamos utilizar por ejemplo líneas de interrupción externas o el puerto serie, deberemos sacrificar algunas de las 32 líneas. Dependiendo del modelo de MCS®-51 tendremos diferentes configuraciones de memoria y cantidad de temporizadores según la siguiente nomenclatura:

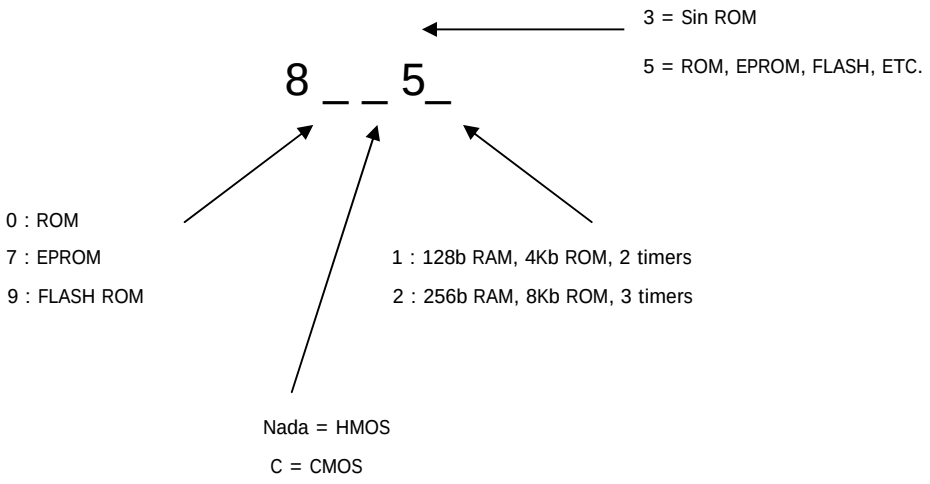


Fig. 3. Nomenclatura de los microcontroladores de la familia 51

Si por ejemplo tenemos entonces un 87C52 disponemos de 8Kb de memoria EPROM, 256 bytes de memoria RAM, 3 temporizadores y todo esto en tecnología CMOS.

3.2 Arquitectura Interna.

En la Fig. 4 vemos la arquitectura interna del 8051. Podemos adelantar:

1. La aparición de un *casi* segundo acumulador llamado B.
2. Que una de las fuentes de entrada a la Unidad Aritmética Lógica es el Acumulador primario o principal (a diferencia del 8088 en el que podían elegirse los registros cualquiera que intervenían en una operación aritmética lógica).
3. La existencia de cuatro puertas de Entrada / Salida designadas P0 a P3[6].

---

[5] En la práctica se emplean cristales de 11,0592 MHz para poder derivar las velocidades más populares de comunicación serie. En efecto 11,0592 MHz/12 provee de una frecuencia de reloj de 921.600 Hz, que empleada por los temporizadores internos y dividida por 96 permite la comunicación a 9600 bits/s.

[6] y cuyas patas se indican con un número después de la puerta, así P0.4 quiere significar la pata 4 de la puerta 0

4. Un registro puntero a memoria de 16 bits denominado DPTR (Data Pointer).
5. Un conjunto de registros de funciones especiales (SFR Special Function Registers) que englobarán todos los registros del microcontrolador.
6. Una comunicación serie asincrónica bidireccional full duplex.
7. Varios temporizadores programables.

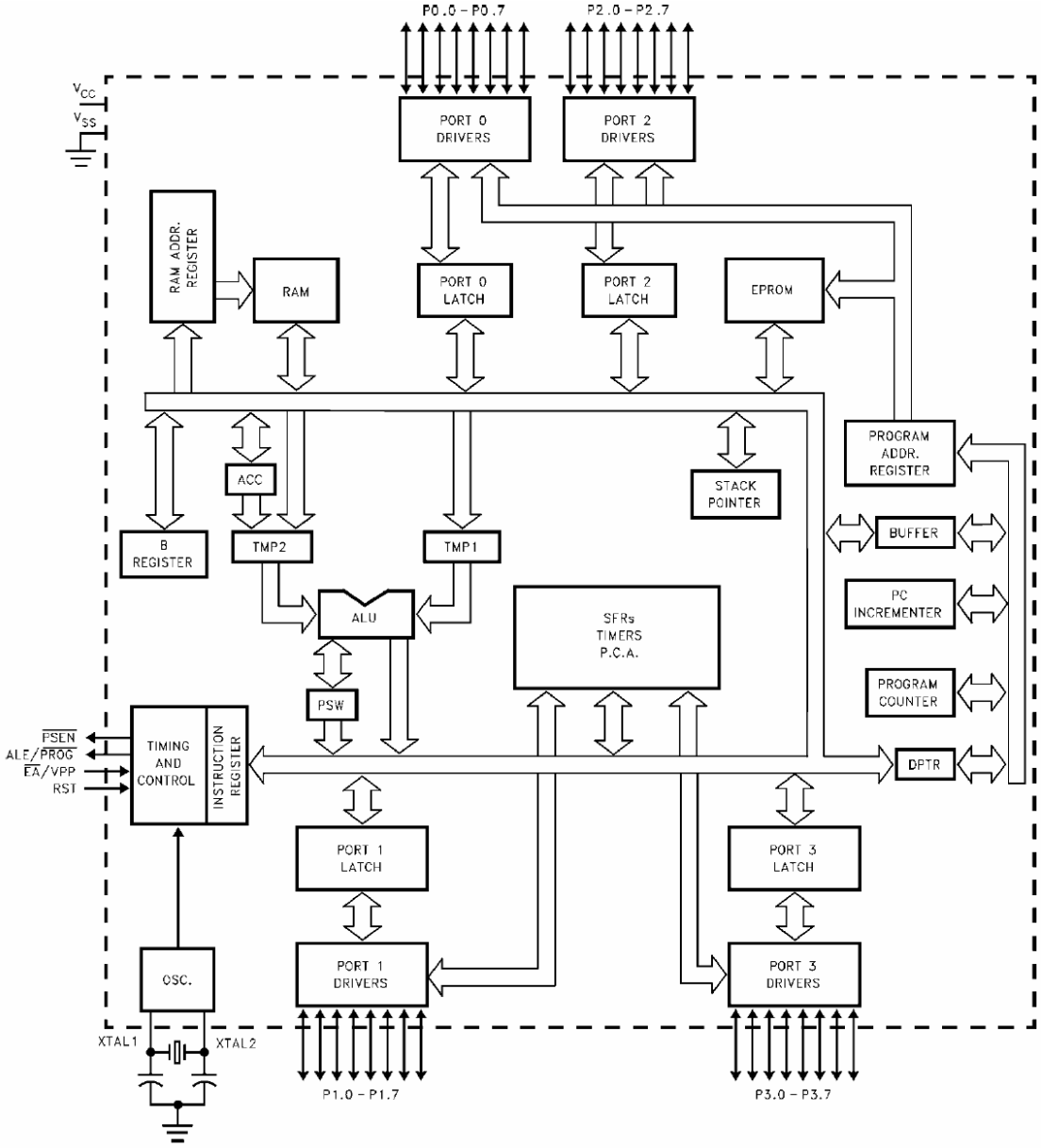


Fig. 4 Estructura interna de un 8X51

3.3 Diagramas de conexionado.

En la Fig. 5 se observa el diagrama de conexionado del 8051 en la tradicional versión DIP (dual in line) y en la versión LCC (Leaded Chip Carrier) que ocupa mucha menos área de circuito impreso al tener por un lado conexiones en los 4 lados del cuadrado y por otro lado la separación entre patas es de media décima de pulgada frente al doble del DIP.

Nos centraremos en la configuración DIP. En ella vemos el significado primario de las patas en la designación más próxima al número de la pata y entre paréntesis el significado ampliado que iremos analizando a lo largo de este capítulo.

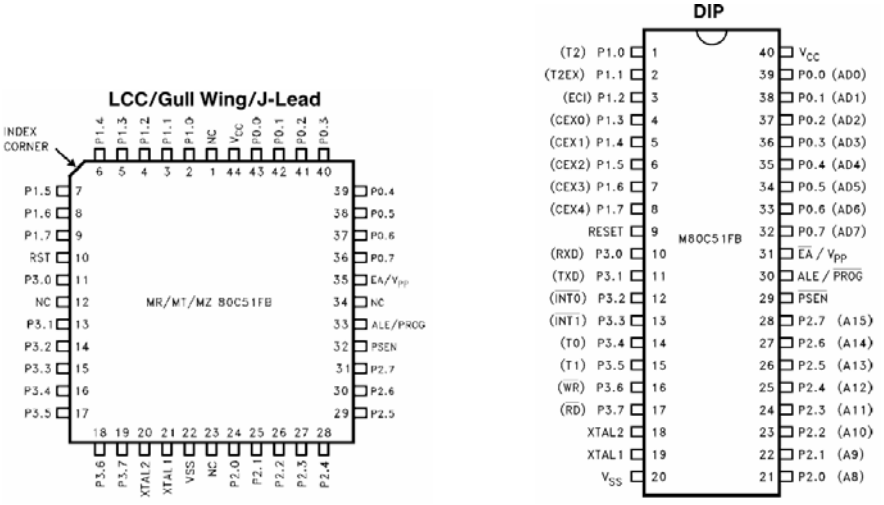


Fig. 5. Patas del Microcontrolador

3.4 Puertas de Entrada / Salida.

Las 32 líneas de Entrada / Salida están organizadas en cuatro puertos de ocho bits, P0, P1, P2 y P3. Cada uno de estos puertos a su vez tiene comportamientos diferentes y pueden cumplir tareas específicas, y aquí en particular haremos una descripción más concisa de estas funciones.

Estos puertos no solo son diferentes en la forma de utilización a nivel funcional sino que también a nivel circuital, ya que la configuración interna de las salidas (Drivers) es diferente entre ellos. Mientras que por ejemplo, la puerta P0 posee una configuración de salida con un “Soft Pull-Up”, esto es una “resistencia” programable por software que hace las veces de Pull-Up (en la práctica se utiliza un mosfet), el resto de las puertas poseen resistores de valor fijo integrados internamente como pull-up.

3.4.1 Puerta 0.

El 8051 permite conectar sobre sus buses externos tanto memorias de datos como de programa o dispositivos de entrada / salida (p.ej. conversores A/D o D/A). A este modo de funcionamiento se lo denomina de “acceso externo” (External Access, o EA).

La puerta 0 se tiene la particularidad de emplearse como bus multiplexado de la parte baja de direcciones y datos (idéntico al del 8085 y 8088), con una señal ALE que indicará el momento en el cual tendremos direcciones a fin de retenerla por medio de un 74LS373.

Como el bus multiplexado deberá pasar por el estado de alta impedancia, la salida es una tri-state estándar (ambos transistores de salida pueden estar cortados) sin ningún tipo de pull-up interno. Por tal motivo, si se desea emplear la puerta 0 como salida para excitar algún dispositivo que

no pueda tener las entradas en alta impedancia (por ejemplo un display, un decodificador, un contador), ***se deberán colocar resistores de pull-up externos.***

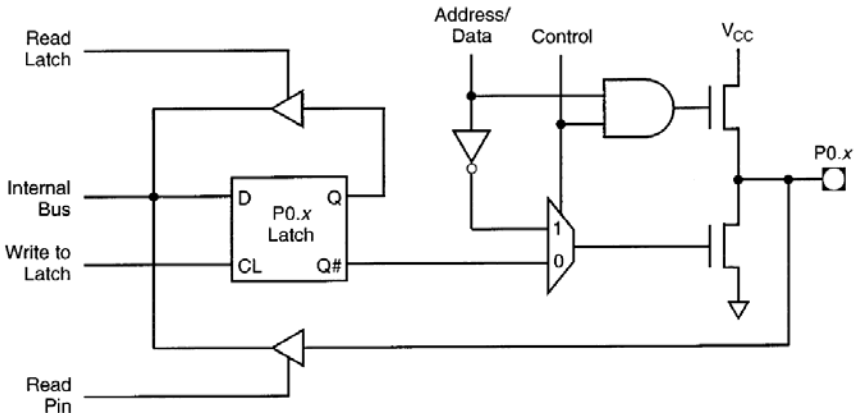


Fig. 6. Estructura de una pata del Port 0

En la Fig. 6 se ve la representación esquemática de una pata de la puerta 0. Allí se pone de manifiesto la dualidad de funciones que puede asumir esa pata. Según lo indique la señal ***Control***, se podrán abrir los dos transistores de salida (alta impedancia) y simultáneamente habilitar la lectura (***Read Pin***), con lo cual el estado de la pata P0.x llegará al bus interno, produciendo una lectura del bus de datos en el caso de operar con acceso externo[7]. En caso de operar como puerta de entrada, la lectura llevará al bus interno del microcontrolador, el estado de la salida del latch.

En la operación de salida, la señal ***Control*** permite que las direcciones o datos (Address / Data) lleguen en forma complementada (inversor de por medio) a los transistores de salida, mientras que en el caso de que se habilite la salida de la puerta 0, almacenada en el latch, sólo se podrá activar el transistor inferior, permitiendo que al conducir el mismo se genere un "0", mientras que si el mismo queda cortado, solo un pull-up externo puede producir un "1".

Basándose en esto último, es que podemos definir una pata como entrada, escribiendo un "1" en el bit correspondiente de dicho puerto.

Adelantándonos en lo referido al repertorio de instrucciones, diremos que si deseamos utilizar como entrada la pata 3 de la puerta P0, ejecutaremos:

```
SET     P0.3
```

Con lo cual quedarán deshabilitados ambos transistores de salida y el estado de la pata será manejado por la fuente exterior.

3.4.2 Puerta 2.

En el próximo tema ***3.5 Mapas de memoria y conexión con el mundo exterior.***, veremos que el 8051 dispone de 64K de capacidad de direccionamiento de memoria externa. Ello requiere 16 líneas de direcciones. Las 8 más bajas aparecerán sobre la Puerta 0 (P0) y las 8 más significativas lo harán sobre la Puerta 2 (P2).

En la Fig. 7 vemos la estructura de una pata de la puerta 2. Es muy similar a la de la puerta 0, pero como no debe tomar el estado de alta impedancia al pasar de direcciones a datos o viceversa, en lugar de un transistor superior, tenemos un resistor que hace las veces de pull-up.

[7] El acceso externo se habilita por medio de la pata /EA para la memoria de programa o por medio de códigos de operación especiales que indican acceso a memorias de datos externas.

Análogamente a lo planteado para la puerta 0[8], para programar una pata como entrada basta con poner un “1” en el bit correspondiente.

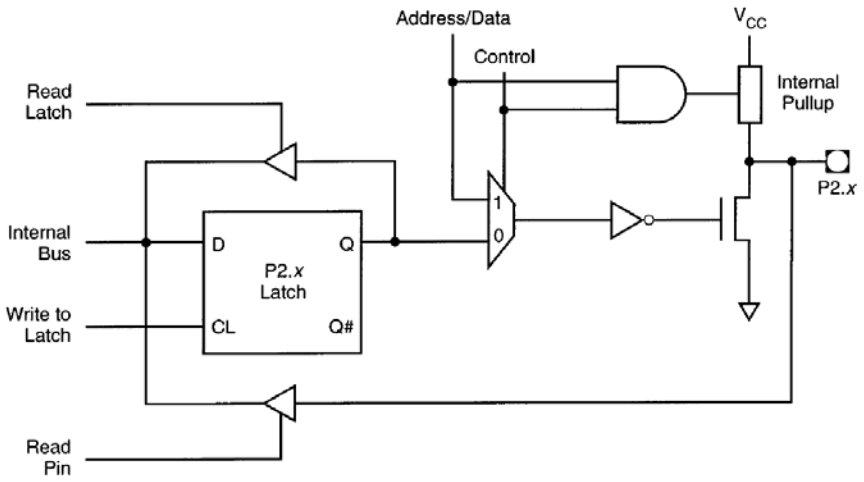


Fig. 7. Estructura de una pata de la Puerta 2

3.4.3 Puertas 1 y 3.

Ambas puertas son idénticas entre si y están representadas por el circuito de la Fig. 8. Vemos que aparece un resistor de pull-up que garantiza un “1” lógico cuando el transistor no conduce.

Las señales del bus de control (/RD, /WR, interrupciones, etc.) se generarán a expensas de líneas de puerta (fundamentalmente la puerta 3 y en algunas versiones se agrega alguna línea de la puerta 1). Por tal motivo es que aparece una señal de comando ***Alternate Output Function*** para las señales de control de salida (por ejemplo /WR, /RD) que suspende la operación de la pata de la puerta y redefine su operación. Análogamente la señal de comando ***Alternate Input Function*** que redefine las patas para señales de control de entrada (Interrupciones).

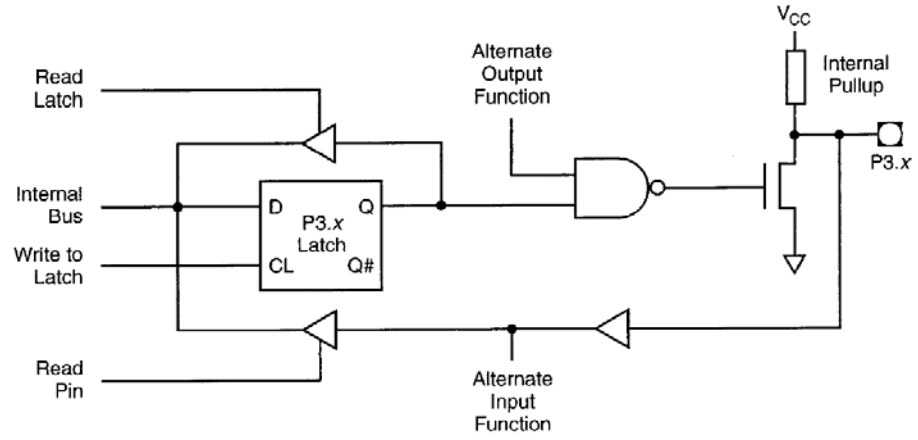


Fig. 8 Estructura de una pata de la puerta 1 ó 3.

[8] También válido para las puertas 1 y 3.

3.5 Mapas de memoria y conexión con el mundo exterior.

3.5.1 Memoria interna y externa.

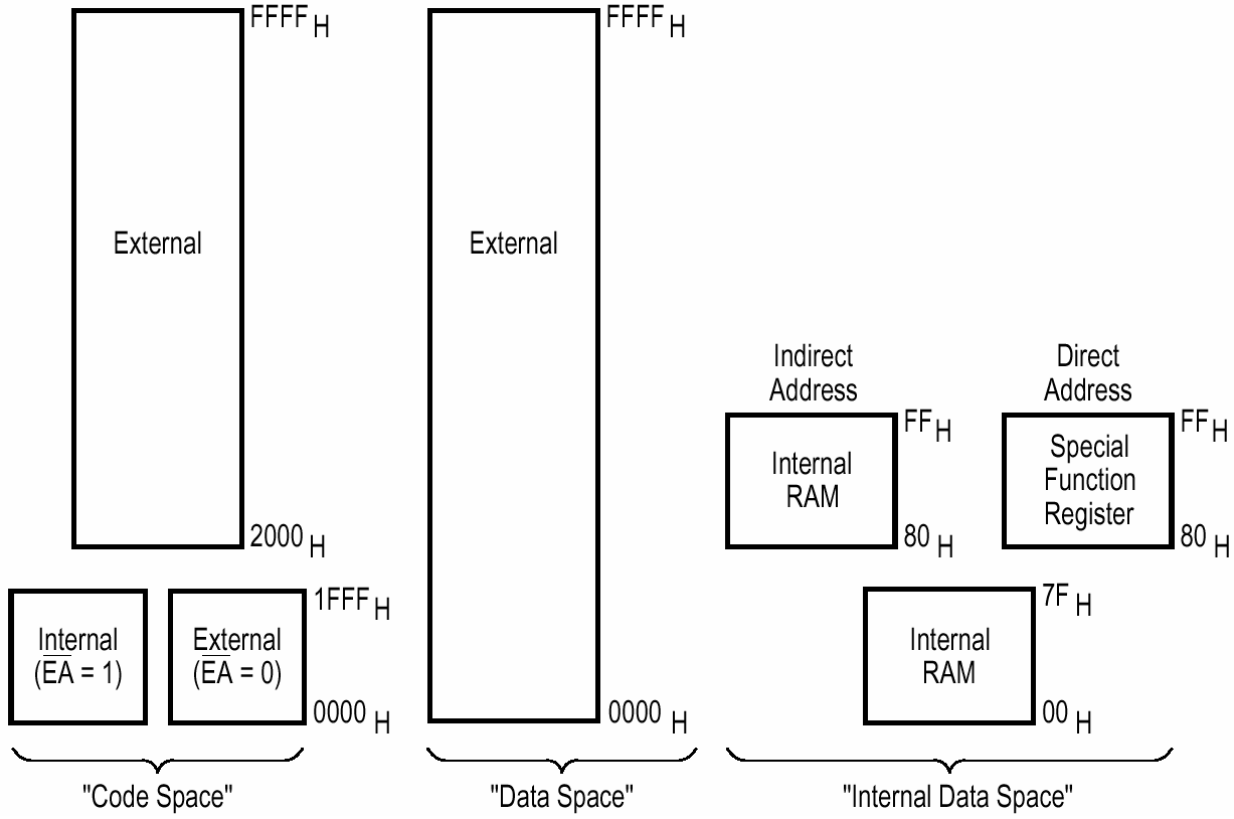


Fig. 9. Mapas De Memoria Interna Y Externa.

La familia 8051 dispone de una pequeña memoria de datos interna de 128 ó 256 bytes según el componente de la familia de que se trate. Existen versiones (Dallas, por ejemplo) que proveen hasta 64 kbytes de memoria de datos incorporada en el circuito del microcontrolador.

La memoria de programa interna puede variar según la versión y el fabricante desde 0 K (versión sin memoria de programa o ROMless) hasta 64 kbytes.

En este momento nos detendremos para analizar más en detalle la organización de la memoria interna del Microcontrolador.

Como puede observarse en la **Fig. 9**, existe para los dispositivos de 256 bytes de memoria interna (8x52 y posteriores), una zona del mapa donde se superponen dos áreas, estas son: el área alta (80_h a FF_h) y el área de SFR ***(Special Function Registers).***

El mecanismo que permite acceder a una zona de memoria o a la otra es utilizando diferentes modos de direccionamiento para referirse a los datos en estas zonas. Más adelante veremos en detalle todo lo que tiene que ver con los modos de direccionamiento para los códigos de operación y el acceso a la memoria de datos interna y externa, por el momento aceptemos que se puede acceder en forma directa a las posiciones de memoria interna con los códigos de operación apropiados, por ejemplo:

```
MOV   P1,A
```

Lleva el contenido del acumulador a la puerta P1, pero la puerta P1 ocupa la posición 90H dentro del mapa de memoria de los registros de funciones especiales. Por ello, la instrucción anterior es equivalente (hasta en el código de operación) a:

```
MOV    90H,A
```

Se observa que se ha empleado direccionamiento directo, es decir que en la instrucción se halla la dirección del operando.

Para acceder a la memoria de datos, se empleará un modo de direccionamiento que denominamos indirecto, donde un registro se utiliza como índice para alcanzar los datos en otra posición de memoria.

Por ejemplo, para escribir el contenido del acumulador en la posición de memoria de datos interna 70H, haremos:

```
MOV    R0,#70H      ; Se inicializa el registro interno R0, en forma inmediata (#) con 70H
MOV    @R0,A        ; Se almacena el Acumulador en la posición de memoria apuntada por R0 (@)
```

Si contamos con una versión tipo 8052 (256 bytes de memoria de datos), podremos escribir la posición de memoria de datos 90H, por medio de :

```
MOV    R0,#90H      ; Se inicializa el registro interno R0, en forma inmediata (#) con 90H
MOV    @R0,A        ; Se almacena el Acumulador en la posición de memoria de DATOS 90H.
```

El modo de direccionamiento ha llevado el contenido del acumulador a la posición de memoria interna de datos 90H. Recalcamos que es sobre la posición de memoria y no sobre la puerta P1 (para escribir en ella, debería emplearse el direccionamiento directo).

Para acceder a la posición de memoria externa de datos, deberemos utilizar un código de operación particular ***(MOVX)*** que hace que el microcontrolador genere la dirección *hacia el exterior* conjuntamente con las señales de control apropiadas para activar la memoria.

3.5.2 Puertas de comunicación con el mundo exterior.

Ahora veremos como se comportan las puertas del Microcontrolador. Como mencionamos más arriba el mecanismo de direccionamiento es utilizando un bus multiplexado en el tiempo, esto es, en un momento aparecen las direcciones sobre el bus utilizando la señal ALE, , y luego de un tiempo determinado, sobre el mismo bus aparecerán los datos asociados a la dirección anterior. En la familia MCS®-51 se reservan los ports P2 y P0 para tal fin, vemos ahora que se produce una merma importante en la cantidad de líneas de Entrada / Salida disponibles para propósitos múltiples al configurar al Microcontrolador en modo de acceso externo.

La señal generada por el Microcontrolador para indicar el comienzo de la generación de una dirección se llama ALE (Address Latch Enable) o habilitación del latch de dirección. Debido a este mecanismo tendremos que colocar un dispositivo para retener la dirección mientras por el mismo bus circulan los datos. Dicho dispositivo se ha transformado en un estándar y es el 74LS373. También existe este mismo pero en una versión que dispone sus entradas y salidas en un formato más conveniente a la hora del diseño de circuito impreso para buses de 8 bits, este es el 74LS573.

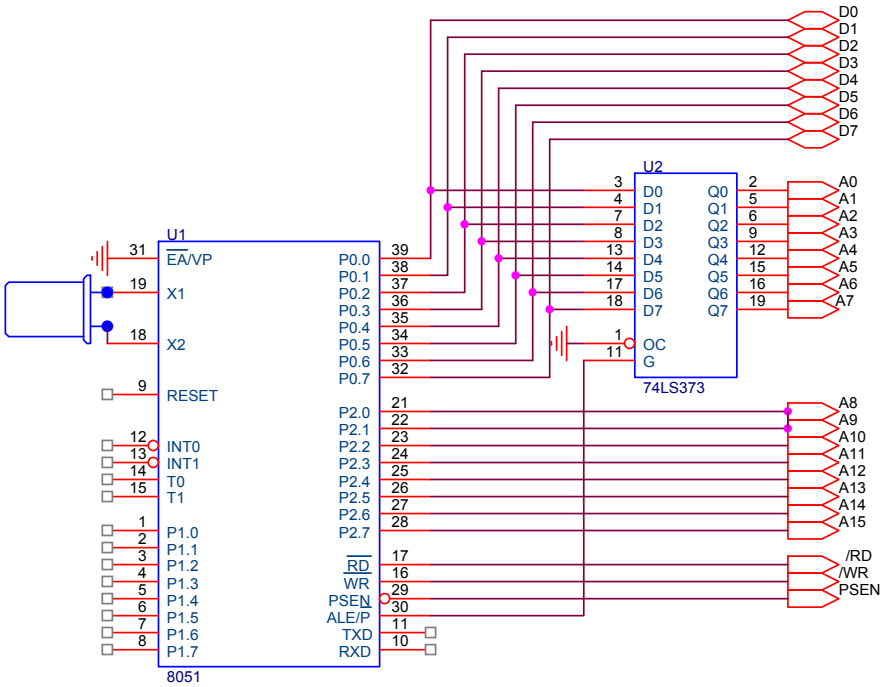


Fig. 10. Diagrama circuital básico de un microcontrolador con expansión de memoria externa.

En la página del Departamento de Electrónica (www.electron.frba.utn.edu.ar) se hallan disponibles ambas hojas de datos y recomendamos consultarlas para interpretar lo mejor posible los conceptos aquí vertidos.

En el circuito de la Fig. 10 vemos el diagrama circuital de un microcontrolador preparado para conectar memoria de programa (obsérvese /EA = “0”) y eventualmente memoria de datos.

Para concluir con nuestro resumen debemos mencionar el comportamiento de tres líneas adicionales:

/RD (Read): Esta línea, activa a nivel bajo, indica primero con su flanco descendente que el microcontrolador esta dispuesto a recibir datos de la memoria de datos[9]. Y segundo, estos datos deben estar presentes en dicho bus al momento del flanco de subida al estado inactivo de la línea /RD, con este flanco el Microcontrolador ingresa los datos.

/WR (Write): La contrapartida de /RD. Esta línea, también activa baja, indica con su flanco de bajada que los datos colocados por el microcontrolador en el bus multiplexado son válidos y que el o los dispositivos que lean este dato disponen del tiempo que dura el pulso de /WR más un pequeño tiempo adicional especificado por cada fabricante para recibir este dato.

/PSEN (Program Store Enable): Esta línea puede utilizarse de dos formas diferentes, como su nombre lo indica hace las veces de /RD pero para la memoria de programa (ROM). Cabe destacar que aunque se utilice acceso interno, esta línea permanece activa. Puede vérsela como una línea más de direcciones entonces podemos tener acceso a 64Kb de datos y 64Kb de programa. Mediante los siguientes mecanismos ejemplificados en la siguiente figura podemos ver como se utiliza esta línea.

[9] Recordemos que el microcontrolador 8051 y sus derivados se basan en una arquitectura Harvard.

Mapas separados

64k código y 64k de datos

Mapas combinados

64k de código y datos (total)

Fig. 11. Mapas de conexión de memoria

3.5.3 Análisis de la función de las patas restantes del microcontrolador.

Ya hemos visto las funciones de la mayoría de las puertas y algunas patas del P3,. Esta puerta, además de proveer las señales de /RD y /WR tiene:

/INT0 e /INT1: Estas son dos entradas para interrupciones externas que pueden utilizarse por nivel o por flanco.

T0 y T1: Estas entradas pueden utilizarse para contar eventos externos con los registros de los temporizadores / contadores internos.

RXD y TXD: Estos dos pines son la entrada y salida respectivamente de la UART interna del Microcontrolador, maneja niveles TTL, por lo que es necesario adicionarle un circuito adaptador de niveles para transformarlo en una comunicación que emplee otra norma, por ejemplo RS232 ó RS485.

Estas ocho últimas patas (/RD, /WR, T1, T0, /INT0, /INT1, RXD Y TXD) conforman el P3. Como vemos, al agregarle funcionalidad al sistema seguimos perdiendo líneas de puertos para propósitos múltiples.

En particular para las versiones de microcontrolador con tres temporizadores, también deben sacrificarse dos líneas de la puerta P1 (que hasta ahora esta totalmente libre), para la entrada T2 que cumple las mismas funciones de T0 y T1, y la entrada T2EX para la captura y recarga del temporizador 2 que luego veremos con más detalle.

XTAL1 y 2: Estas patas sirven para conectar un cristal oscilador externo o en su defecto, una señal externa de reloj en la pata XTAL2 dejando el XTAL1 al potencial más negativo de alimentación del Microcontrolador (Vss o GND). El fabricante recomienda el siguiente esquema de conexión.

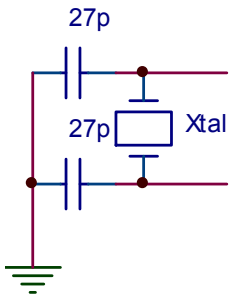


Fig. 12. Conexión del Cristal.

Vcc y Vss: Aquí se conecta la fuente de alimentación del Microcontrolador, los requerimientos de corriente y niveles de tensión dependen del tipo de dispositivo (familia lógica y versión) y cargas conectadas, recomendamos consultar las hojas de datos de estos dispositivos.

RESET: Produce una inicialización de registros internos en el Microcontrolador deteniéndose la ejecución del programa en curso, se carga el contador de programa[10] con el vector de inicio (0000_h en este caso de reset, más adelante veremos los restantes vectores del Microcontrolador). La condición para que se produzca el proceso de reset es que se mantenga a nivel alto la pata correspondiente por lo menos durante 2 ciclos de máquina mientras funcione el oscilador. De esta manera se inicia la acción de un generador de bombeo (pump generator) que internamente rectifica la señal del oscilador y provee la polarización negativa del sustrato necesaria para la operación del dispositivo.

3.6 Registros de funciones especiales (SFRs)

Aprovechando que debemos ver lo que sucede con los registros internos a la hora del reset, haremos una aproximación a la descripción de los registros de funciones especiales (SFR).

Como todo procesador el 8051 posee registros para funciones varias, la particularidad que tiene este Microcontrolador es que estos registros forman parte de la RAM interna accesible por programa. Por ejemplo, los registros acumuladores para operaciones generales poseen una dirección en la zona de SFR.

Esto significa que no solo vamos a tener un código de operación para poner un acumulador en cero, también podemos hacerlo poniendo en cero la posición de memoria correspondiente a dicho acumulador mediante ***el mismo*** código de operación.

Recordando lo que mencionamos al comenzar este apunte sobre la “filosofía del bit”, algunos de los registros en la zona de SFR son direccionables a nivel de bit, al igual que posiciones de memoria baja. En el gráfico tenemos un mapa simplificado y en la tabla un detalle de los registros del área SFR.

En la Fig. 13 presentamos los registros de funciones especiales agrupados por bloques.

[10] Versión para el 8051 del IP del 8088 con la particularidad de que al no haber cola interna, apunta a la próxima instrucción a ejecutar

Block	Symbol	Name	Address	Contents after Reset
CPU	ACC	Accumulator	**$E0_H$** 1)	00_H
	B	B-Register	**$F0_H$** 1)	00_H
	DPH	Data Pointer, High Byte	83_H	00_H
	DPL	Data Pointer, Low Byte	82_H	00_H
	PSW	Program Status Word Register	**$D0_H$** 1)	00_H
	SP	Stack Pointer	81_H	07_H
Interrupt System	IE	Interrupt Enable Register	**$A8_H$** 1)	$0X000000_B$ 3)
	IP	Interrupt Priority Register	**$B8_H$** 1)	$XX000000_B$ 3)
Ports	P0	Port 0	**80_H** 1)	FF_H
	P1	Port 1	**90_H** 1)	FF_H
	P2	Port 2	**$A0_H$** 1)	FF_H
	P3	Port 3	**$B0_H$** 1)	FF_H
Serial Channel	PCON 2)	Power Control Register	87_H	$0XXX0000_B$ 3)
	SBUF	Serial Channel Buffer Register	99_H	XX_H 3)
	SCON	Serial Channel Control Register	**98_H** 1)	00_H
Timer 0 / Timer 1	TCON	Timer 0/1 Control Register	**88_H** 1)	00_H
	TH0	Timer 0, High Byte	$8C_H$	00_H
	TH1	Timer 1, High Byte	$8D_H$	00_H
	TL0	Timer 0, Low Byte	$8A_H$	00_H
	TL1	Timer 1, Low Byte	$8B_H$	00_H
	TMOD	Timer Mode Register	89_H	00_H
Timer 2	T2CON	Timer 2 Control Register	**$C8_H$** 1)	00_H
	T2MOD	Timer 2 Mode Register	$C9_H$	$XXXXXXX0_B$ 3)
	RC2H	Timer 2 Reload/Capture Register, High Byte	CB_H	00_H
	RC2L	Timer 2 Reload/Capture Register, Low Byt	CA_H	00_H
	TH2	Timer 2 High Byte	CD_H	00_H
	TL2	Timer 2 Low Byte	CC_H	00_H
Pow. Sav. Modes	PCON 2)	Power Control Register	87_H	$0XXX0000_B$ 3)

Fig. 13. Registro de funciones especiales agrupados por bloques.

NOTAS:

1) Registros de funciones especiales direccionables a nivel de bit.
2) Este Registro de Funciones Especiales aparece listado en múltiples oportunidades ya que algunos de sus bits pertenecen a otros grupos funcionales.
3) “X” significa que el valor esta indefinido (para el 8051) y la posición reservada.

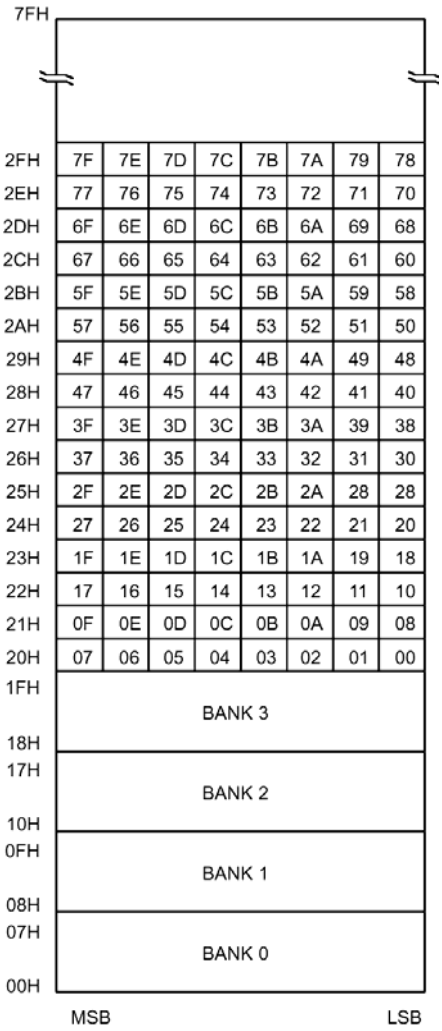


Fig. 14. Memoria interna. Bancos de registros y zona direccionable por bits.

Como se puede apreciar en la Fig. 14, hay una zona de registros que es direccionables a nivel de bit que está comprendida entre los bytes 20_h y $2F_h$ (en total 16 bytes que dan 128 bits), cada uno de los bits de esta zona se numera de 0 a 127 y pueden ser accedidos por sendos código de operación.

Por ejemplo, las dos operaciones siguientes son equivalentes:

```
SETB   66H     ; Pone en “1” el bit 66H computado a partir del bit 0 de la posición 20H.
SETB   2C.6    ; Pone en “1” el bit 6 de la posición de memoria de datos 2CH.
```

Más abajo podemos notar la existencia de cuatro bancos de ocho registros cada uno. Estos registros son seleccionados por un par de bits dentro de un registro llamado PSW (PROGRAM STATUS WORD, que luego veremos más en detalle).

La pila se encuentra dentro de la escasa memoria de datos interna, por lo cual al producirse una interrupción y querer salvar los registros en uso se podría agotar fácilmente la disponibilidad de memoria. Por tal motivo se implementó el cambio de banco de registros activo como forma de salvaguardar los registros sin perder demasiado tiempo. Ello se realiza cambiando un par de bits en el registro PSW que almacena el estado del programa[11].

[11] Inclusive los compiladores proveen una forma muy simple de efectuar el cambio de banco de registros activo por medio de la sentencia “*Using”* así si escribimos en nuestro programa ***Using 2*** el compilador generará las instrucciones necesarias para cambiar los bits del PSW y activar al banco 2.

Luego del Reset se halla activo el banco de registros 0 que abarca los registros R0 a R7 que ocupan las posiciones de memoria de datos 0 a 7.

Supongamos que se produce una interrupción de la fuente de interrupción A (temporizador, comunicación serie, interrupción externa, etc.). En un 8088 intentaríamos salvar en la pila los registros en uso. Aquí pasamos a emplear, por ejemplo, el banco de Registros 2 de forma que al emplear la instrucción:

MOV R0,#32H

No se cargará el registro R0 ubicado en la posición de memoria de datos 0, sino el Registro 0 del banco 2 que se halla en la posición de memoria 10H (ver Fig. 14). De esta morma, hemos "salvado" todo el banco de registros 0 sin emplear la pila.

Resumiendo:, al tener seleccionado un banco, se utilizan con ciertos código de operación solo los registros del banco en cuestión, al cambiar de banco y utilizando los mismos código de operación se opera sobre el banco ahora activo quedando intacto el contenido de los registros del banco seleccionado originalmente[12].

Como dichos registros se numeran de R0 a R7, existen cuatro R0, cuatro R1, etc. y al referirnos a uno de ellos debemos tener en cuenta a que banco pertenece para seleccionarlo si es necesario.

El resto de los registros del área de SFR serán explicados a medida que vayamos avanzando sobre los temas con ellos relacionados a lo largo de este apunte.

4. Operación con memoria externa.

4.1 Memoria de Programa.

Hemos mencionado en varias oportunidades que el microcontrolador puede disponer de hasta 64 K de memoria externa de programa y de 64K de datos.

Si bien las memorias se conectan sobre los mismos buses de direcciones y datos, mencionamos que la pata /EA (External Access) es la que obliga al microcontrolador a ir a buscar el programa a la memoria externa. En tal situación será la señal /PSEN (Program Store Enable) la que produce la lectura. Como es memoria de programa, no habrá necesidad de ninguna señal de escritura.

En la Fig. 15 vemos el diagrama básico de conexionado con memoria de programa externa. La señal ALE sirve para demultiplexar la parte baja del bus de direcciones y la señal /PSEN actúa sobre la habilitación tristate de salida del bus de datos de la memoria.

Para un 8051, la memoria externa de programa se accederá en dos circunstancias:

1. Cuando /EA este conectada a "0".
2. Cuando el contador de programa supere 1FFFH.

[12] Gracias a este rudimentario mecanismo de cambio de contexto (Context Switching), algunas compañías de software han desarrollado núcleos de sistema operativo (kernels) en tiempo real (RT kernels) para Microcontroladores de la familia MCS®-51 (ver citas bibliográficas y WEBográficas al final del apunte) estos nos permiten ejecutar varias tareas utilizando el concepto de ranuras de tiempo (Time Slicing O Time Slots), donde a cada tarea se le asigna un tiempo de procesamiento en general configurable y el kernel se encarga de conmutar entre las tareas activas, concepto similar al de los sistemas operativos como el UNIX y sus derivados o los basados en Windows.

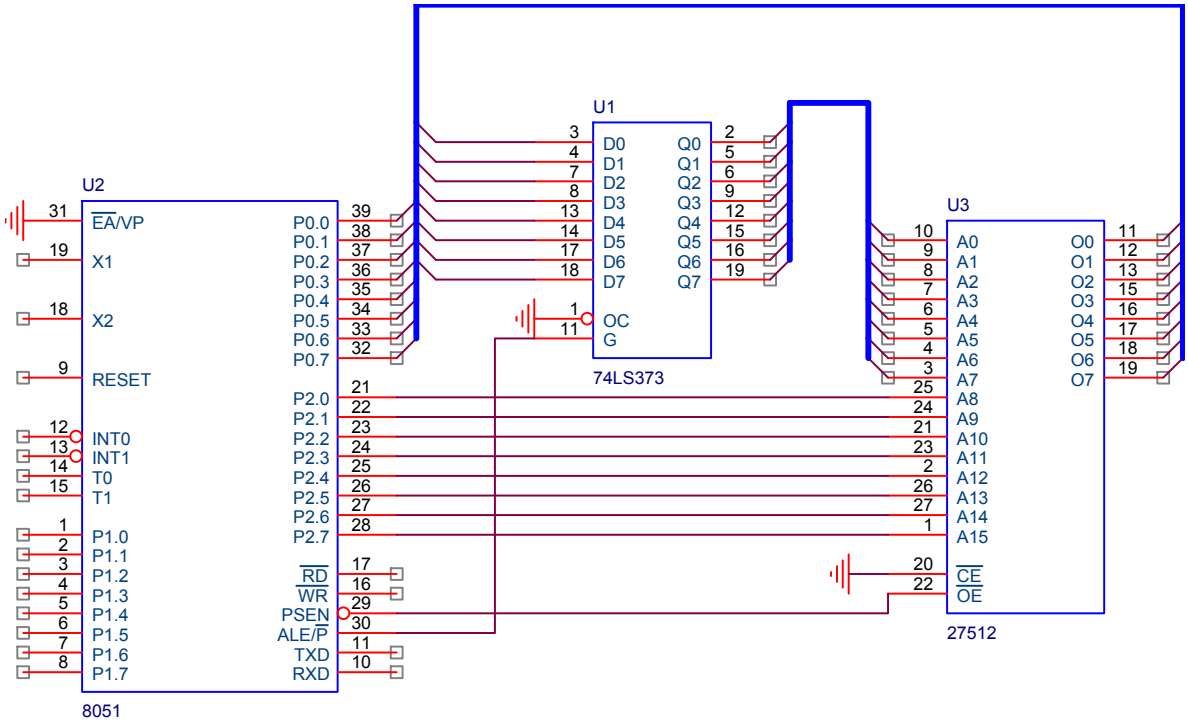


Fig. 15. Circuito del 8051 con memoria de programa externa.

En la Fig. 16 vemos el diagrama temporal de la lectura de memoria de programa. Allí vemos que cada ciclo de máquina consta de 6 estados (S1 a S6) y por la especificación del manual, sabemos que cada estado consta de dos ciclos de cristal, por lo que cada ciclo de máquina demora 12 ciclos de cristal.

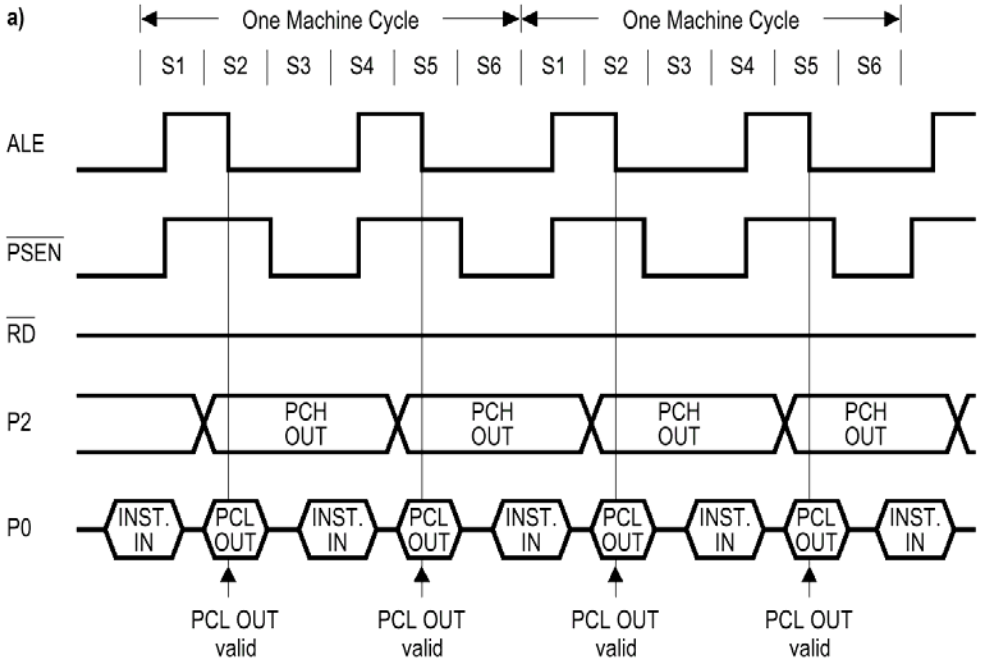


Fig. 16. Captura de código de Operación en memoria externa.

En la **Fig. 16** también vemos que ALE aparece en el primer estado del cada ciclo de máquina y baja en el segundo. En ese momento en P0 se halla presente la parte baja de contador de programa (PCL). Cuando el microcontrolador retira PCL de P0 para a alta impedancia y espera que

la memoria de programa (cuya salida ha sido habilitada por /PSEN) coloque el código de operación u operando sobre el bus de datos. Sobre P2 aparece estable la parte alta de PC.

4.2 Memoria de Datos.

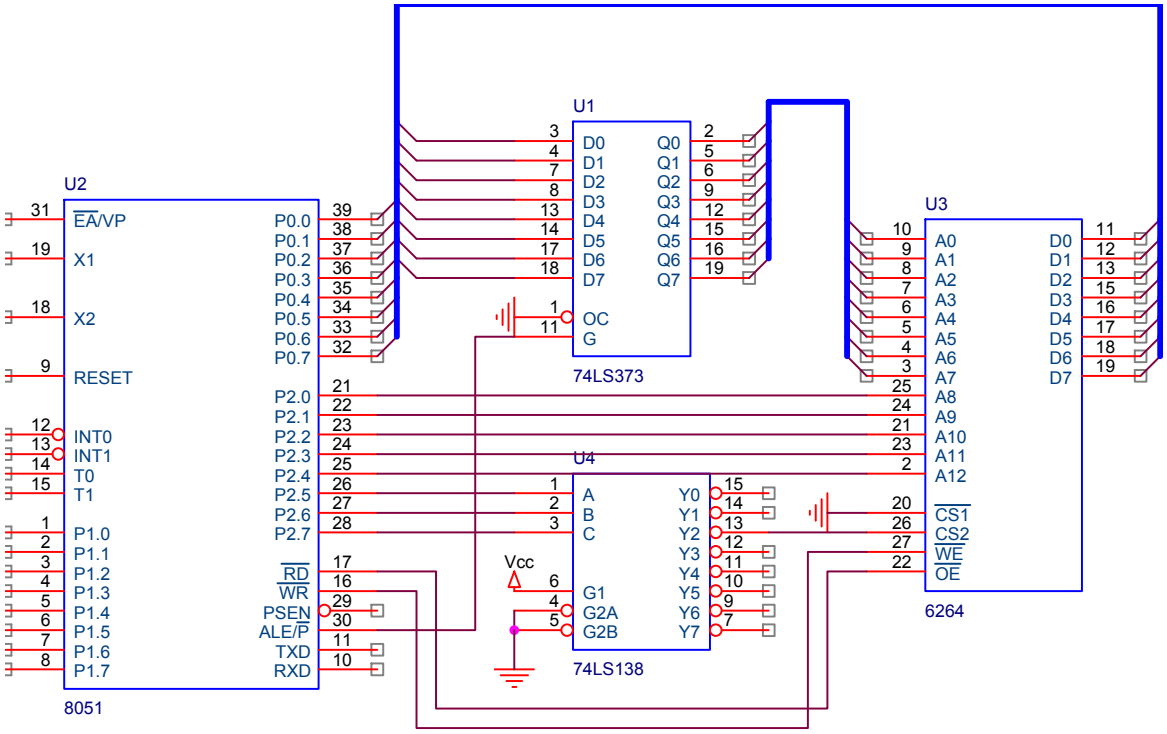


Fig. 17. Microcontrolador 8051 con memoria de datos externa.

Para la operación con una memoria de datos externa, la configuración es muy similar a la vista en la Fig. 15. Solamente que se emplean las señales /RD y /WR para acceder a la memoria de datos.

En la Fig. 18 tenemos el diagrama temporal con memoria de programa y de datos externa.

En el primer ciclo de máquina todo ocurre como en el caso anterior. Para acceder a la memoria de datos externa se deberán emplear las instrucciones adecuadas (MOVX) y se sacará en el conjunto P2 – P0 el puntero a datos (DPTR) o en P0 se sacará el valor del puntero de 8 bits a memoria de datos (R0 o R1 del banco de registros activos) manteniendo P2 como puerta de E/S.

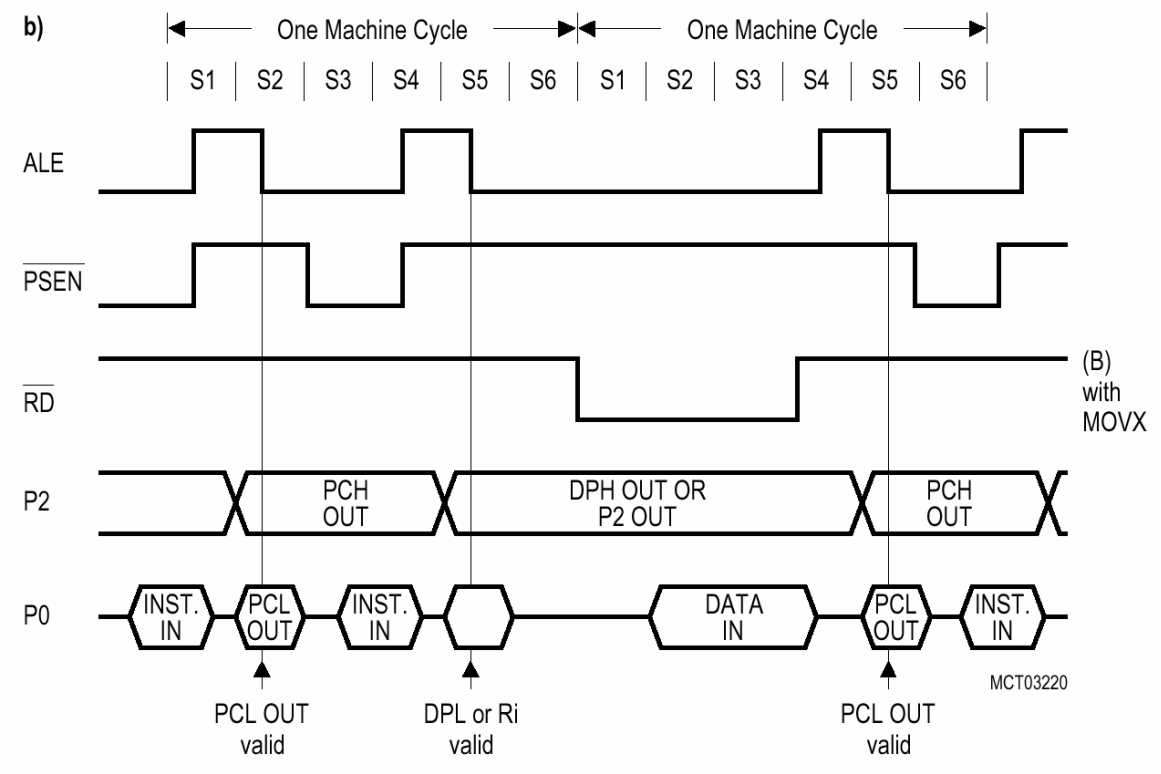


Fig. 18. Diagrama temporal con memoria de programa y lectura de datos externos.

Similarmente se procede para el escritura, empleándose la señal /WR en lugar de /RD y será el microcontrolador el que escribe los datos.

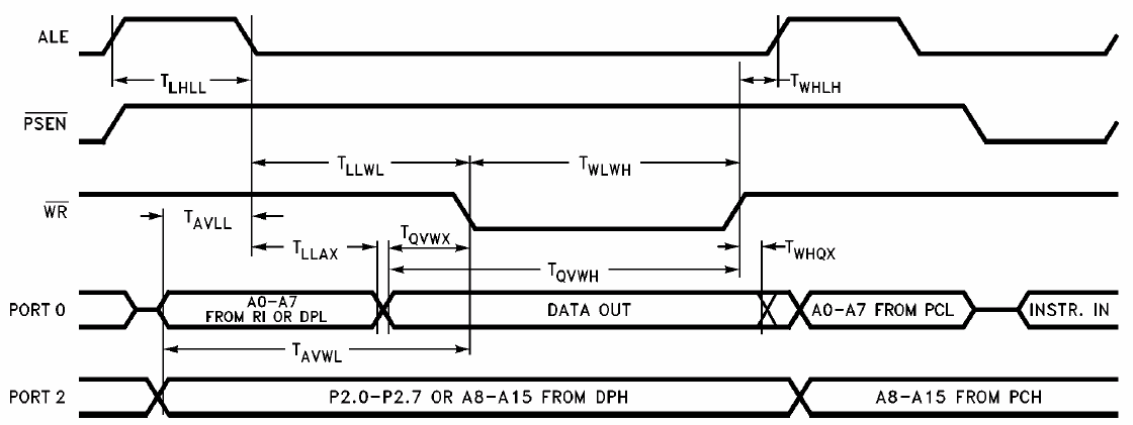


Fig. 19. Diagrama temporal con memoria de programa y escritura de datos externos.

5. Modos de direccionamiento.

Tal como vimos en el caso de los microprocesadores, estos modos de direccionamiento se refieren a la forma en que se le entregan al microprocesador los operandos para ejecutar una instrucción y como su nombre lo indica, se clasifican por la forma en que se accede a los datos a los que hacen referencia y como se ejecutan los código de operación.

De forma análoga al 8088, cada instrucción del microcontrolador consta en general de un código de operación y datos asociados a los que hemos llamado operandos. La CPU del Microcontrolador se encarga de extraer de la memoria de programa estos códigos de operación y sus operandos y los decodifica para luego obrar en consecuencia.

La forma en que se codifican los código de operación depende del modo de direccionamiento o sea de la forma en que se proveen de los operandos al microcontrolador. En esta familia de Microcontroladores los código de operación son de 8 bits, lo que en principio nos proporcionaría 256 código de operación posibles, pero en realidad no están implementados todos, el único no implementado o mejor dicho, no documentado es el que corresponde al byte $A5_h$ (10100101_b). Recomendamos consultar el apéndice de tablas para más detalles.

En la familia MCS®-51 existen seis modos de direccionamiento, estos son:

- ***Implícito***
- ***Inmediato***
- ***Directo***
- ***Indirecto***
- ***Por registro***
- ***Indexado***

Antes de empezar a ver en detalle estos modos aclararemos ciertos tópicos y definiciones concernientes al ensamblador del Microcontrolador y algunos registros.

La CPU de los Microcontrolador de la familia MCS®-51 posee registros de 8 bits, y en particular dos de 16 bits, uno de ellos compuesto por dos registros de 8 bits. Dentro de los registros que maneja esta CPU tenemos dos acumuladores, A (también llamado ACC, nomenclatura que adoptamos de ahora en adelante) y B, estos registros y en particular el ACC, poseen propiedades que los diferencian de otros registros del Microcontrolador. Por ejemplo el ajuste decimal para manejar números en BCD solo puede hacerse sobre el ACC, cuando se debe decidir si un operando es cero, el mismo deberá estar en el acumulador, etc.

Como mencionamos anteriormente existen cuatro bancos de registros llamados R0 a R7, de los cuales el ***R0*** y ***R1*** cumplen también funciones especiales como actuar de registros ***punteros a memoria*** de datos en ciertos modos de direccionamiento.

El 8051 puede acceder a 64 K memoria de datos y otros 64K de memoria de programa, por lo que se necesitará (por lo menos de) un registro de 16 bits para acceder a todo el mapa de memoria.

Uno de los registros de 16 bits mencionado anteriormente se llama DPTR (Data Pointer) el cual esta compuesto por dos registros de 8 bits, el DPL (los ocho bits menos significativos) y el DPH (los ocho bits más significativos). .

El otro registro de 16 bits es ni más ni menos que el contador de programa (Program Counter), el cual es actualizado por la CPU del Microcontrolador con la dirección del próximo código de operación u operando a acceder.

Otro registro muy importante es el PSW (Program Status Word o Registro de estado del programa o Registro de códigos de condición), direccionable a nivel de bit y que contiene ciertos indicadores o “flags” que indican el estado de la CPU, sus bits indican lo siguiente

Registro: PSW - Bit Addressable							
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
C	AC	F0	RS1	RS0	0V	-	P

B7, C: bit de acarreo (Carry), se pone a 1 al producirse acarreo desde el bit 7 del ACC, o sea siendo el ACC.7 = 1 se le suma 1 a este bit. Ejemplo:

ACC = 10011011

\+

10000100

ACC = 00011111 y el carry es = 1 ya que se produce acarreo en el bit 7.

B6, AC: acarreo auxiliar, funciona igual que el anterior solo que al producirse el acarreo del nibble inferior del ACC al superior.

B5, F0: Flag cero, es un bit de propósito general disponible para el usuario, que puede ponerlo a cero, hacerlo valer "1" a voluntad y luego tomar decisiones en base a ello por medio de saltos condicionados a su valor (JF0).

B4 y B3: Con estos bits se selecciona el banco de registros activo (R0 a R7) según la siguiente tabla:

RS 1	RS 2	Banco
0	0	0
0	1	1
1	0	2
1	1	3

B2, OV: Desborde u overflow. Este bit es el resultado de la operación OR-EXCLUSIVO entre el resultado del acarreo de los bits ACC.7 y ACC.6, es decir, si no se produce acarreo en ambos o si se produce en ambos, el OV es cero. Según lo visto en Técnicas Digitales I, este bit previene el excederse de la capacidad del acumulador, es decir que busca advertir al programador que ha sumado dos números positivos y el resultada ha sido (absurdamente) negativo o similarmente que al sumar dos números negativos, el resultado fue positivo.

B1 : Este al igual que F0 es un bit disponible para propósitos generales solo que no tiene un nombre en particular, se hace referencia a el como "PSW.1".

B0, P: Bit de paridad, indica la paridad de la cantidad de bits en uno en el acumulador incluido el bit de paridad, es decir, si la cantidad de bits en uno en el acumulador es impar entonces el bit P = 1, se dice que este bit indica "paridad par".

Veremos a continuación de que se trata cada uno de los modos de direccionamiento.

5.1 Direccionamiento implícito (o inherente):

Este modo es el más sencillo y consta de un solo código de operación, no requiere operandos adicionales ya que el mismo código de operación implica a los datos que modifica (de allí el nombre de implícito). Pero para aclarar este concepto veamos un ejemplo:

Supongamos que queremos poner el contenido del acumulador A en cero, para ello existe un código de operación de modo implícito que realiza esta operación:

CLR A ; limpia el contenido del acumulador

Por ejemplo también tenemos operaciones como dividir el contenido del acumulador A con el del acumulador B (obviamente requiere que previamente se carguen los contenidos de dichos acumuladores con los números a dividir):

DIV AB ; realiza la operación A/B, coloca el cociente en A y el resto en B

Como podemos apreciar, con un simple código de operación el microcontrolador tiene toda la información necesaria para ejecutar la instrucción.

5.2 Direccionamiento inmediato:

Aquí el byte que sigue inmediatamente al código de operación es tratado como una constante, es decir, no se utiliza como una dirección o haciendo referencia a algún otro registro. Podríamos compararlo con una sentencia de asignación en un lenguaje de alto nivel, por ejemplo:

MOV A,#3Eh ;carga el ACC con la constante (número) $3E_h$, no el contenido del registro $3E_h$.

Es de hacer notar la utilización del símbolo "**#**" indicando que lo siguiente es una constante.

5.3 Direccionamiento directo:

En este modo, el código de operación va acompañado de un operando que contiene la dirección de un byte de la RAM interna (de 0 a 127), o la dirección de un registro de la zona de SFR (de 128 a 255). Recordando lo que mencionamos sobre la forma de acceso a la memoria en los dispositivos de 256 bytes de memoria interna, este es el modo que permite el acceso a la zona de SFR.

De esta manera si por ejemplo deseamos “MOVer” el contenido del acumulador A al byte $3F_h$ de la RAM interna, el código de operación es el siguiente:

MOV 3Fh,A ; mueve el contenido de A a la posición $3F_h$ [13]

Es interesante notar que a este mismo modo también pertenecen algunos código de operación que operan sobre bits en lugar de bytes. En este caso el operando que acompaña al código de operación indica la dirección de uno de los bits direccionables (de la zona 20_h a $2F_h$ o de los registros con la propiedad direccionables a nivel de bit), por ejemplo:

SETB 0Fh ; pone en “1” el bit 7 de la posición de RAM interna 21_h

Otra notación posible para el mismo código de operación en el ensamblador de este microcontrolador es:

SETB 21h.7 ; o sea el bit 7 de a posición de memoria de datos 21_h

De la misma forma se puede operar con los acumuladores y registros direccionables por bits, por ejemplo:

SETB ACC.2 ; pone en “1” el tercer bit del acumulador A.

[13] Nótese que el ensamblador del microcontrolador requiere la especificación de la base de numeración (h en este caso) para el operando[13]. Se puede especificar en binario (00111111b) o en decimal (63d).

5.4 Direccionamiento indirecto:

Aquí el escenario es algo más complejo. Podríamos clasificar el direccionamiento indirecto en dos sub modos, indirección a través de registros de 8 bits e indirección a través de registros de 16 bits.

Básicamente, ya sea mediante indirección de 8 o 16 bits, la idea es hacer referencia a un dato apuntando a la posición de memoria que lo contiene con el contenido de un tercer registro (de 8 o 16 bits). Con el siguiente ejemplo trataremos de ponerlo un poco más claro:

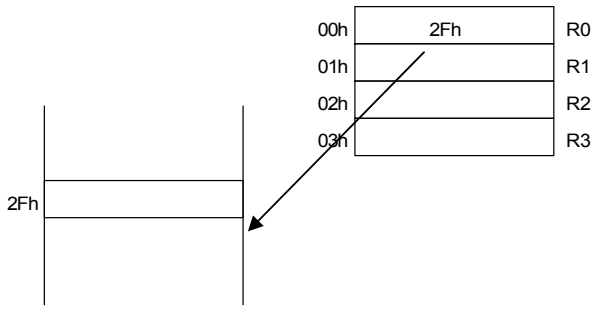


Fig. 20. Ejemplo de direccionamiento indirecto.

Aquí, utilizando el recientemente introducido concepto del direccionamiento inmediato, cargaremos el registro R0 con el valor $2F_h$:

MOV R0,#2Fh ; Se carga R0 con $2F_h$

Ahora, utilizando el direccionamiento indirecto, movemos el contenido de la posición apuntada por R0 ($2F_h$) al ACC:

MOV A,@R0 ; pone el contenido de $2F_h$ en ACC.

El símbolo "@" (at..., que en inglés significa “en el sitio...” o “en el lugar...”) nos indica que el dato es "apuntado" por el registro R0 y no "contenido" en él.

5.5 Direccionamiento por registro:

Este modo es particular de estos microcontroladores, ya que se aplica en código de Operación que utilizan el número de registro R0 a R7 del banco actualmente seleccionado o activo. Los código de Operación que pertenecen a este modo llevan en su byte el número de registro seleccionado, por ejemplo:

DJNZ R5,LOOP ; decrementa R5 y salta a "LOOP" si R5 no llega a ser cero en el decremento.

Esta es una instrucción muy poderosa que nos permite armar el equivalente de lazos al estilo "FOR..NEXT", como analizaremos posteriormente.

Lo que nos interesa ahora es notar la codificación de esta instrucción. Analicemos los dos bytes que componen esta instrucción:

1 1 0 1 1 x x x **a a a a a a a a**

Los primeros cinco bits del primer byte (1 1 0 1 1) identifican al código de operación del "DJNZ Rx,Dirección", los bits marcados como xxx codifican el número de registro utilizado en esta instrucción y el segundo byte indican una dirección relativa (en complemento a 2) a la de la instrucción en curso.

5.6 Direccionamiento indexado:

La indexación a la que se refiere este modo solo es posible sobre la memoria de programa (ROM/EPROM/FLASH). Y aquí cabe la pregunta ¿para qué puede servirnos apuntar con un índice sobre la memoria de programa?. Pues bien, es bastante útil para armar tablas estáticas en esta memoria, es decir, no utilizamos la memoria de programa solo para guardar código de Operación o programa, la utilizamos por ejemplo para guardar una tabla que nos permita al leer la tensión de salida de una termocupla, hacer la conversión de mV a ºK (que sería impracticable de hacer por medio de la expresión matemática que las vincula) o realizar una tabla con direcciones de salto a subrutinas varias en base a un dato que actúa como índice de la tabla.

Para el manejo de estas tablas son necesarios dos elementos, una dirección base de la tabla y tamaño definido de la tabla.

El primer elemento es necesario como punto de referencia y será apuntado por un registro ad hoc, si pretendemos que nuestra tabla se ubique en cualquier punto de nuestra memoria de programa y si recordamos que dicha memoria puede ser de hasta 64 Kb concluimos entonces que es necesaria la utilización de un registro de 16 bits para ubicar (apuntar) el comienzo de nuestra tabla. De aquí se desprende que los únicos registros disponibles para tal fin son el Puntero a datos (DPTR) y el Contador de Programa (PC).

El segundo elemento para manejar la tabla es un índice para recorrerla desde la dirección base (apuntada por alguno de los registros de 16 bits) hasta su tamaño máximo, aquí la posibilidad es un límite de 256 bytes, de esta forma utilizando algún registro de 8 bits (el ACC por ejemplo) nos desplazamos a lo largo de la tabla.

Para el manejo de estas tablas entonces, existen dos sentencias del ensamblador que nos permiten utilizar el ACC y el DPTR o el PC.

6. Anatomía de los códigos de operación.

En este capítulo trataremos en detalle los código de Operación de la familia MCS®-51, y convendremos algunos detalles de notación para interpretar las tablas y ejemplos de codificación.

Las instrucciones del 8x51 pueden codificarse en instrucciones de hasta tres bytes. El primer byte siempre es el código de operación, los siguientes pueden existir o no dependiendo precisamente del modo de direccionamiento que involucre el código de operación de que se trate.

También de esto dependerá el significado de los bytes y hasta los bits que acompañan y conforman el nemónico en cuestión. Luego veremos los diagramas temporales ***(tempogramas)*** que describen el comportamiento de las señales que componen los buses de control, datos y direcciones cuando la CPU del microcontrolador efectúa en la memoria de programas la búsqueda de código de Operación para ejecutar ***(Op Code Fetch)*** y el acceso a memoria externa de datos.

6.1 Manejo de memoria.

Analizaremos las instrucciones del ensamblador según los tipos de las mismas, es decir, aritméticas, de movimiento de datos, lógicas, etc.

Utilizaremos las siguientes convenciones para especificar la sintaxis de las instrucciones del ensamblador:

- Todo lo que aparezca entre paréntesis significa “el contenido de...”.

  El símbolo “←” indica movimiento de datos hacia donde apunta la flecha.

- Cualquier registro o posición de memoria seguida de un punto y un número de 0 a 7 hace referencia a un bit de dicho registro o posición de memoria (el número indica el bit en cuestión).

Comenzamos entonces con las instrucciones para el movimiento (MOV) de datos entre registros. El propósito de estas instrucciones es permitir la asignación de valores a registros o modificar el contenido de los mismos intercambiando si es necesario con el contenido de otros registros. Esto se puede llevar a cabo con instrucciones que utilizan direccionamiento directo, indirecto, por registro, inmediato e indexado. Así, tenemos entonces el ...

6.1.1 Grupo MOV:

Este grupo de instrucciones consta de dos operandos, uno de ellos es el origen y el otro el destino, el concepto del MOV es que el contenido del lugar especificado por el operando origen pasa (se copia) al lugar especificado por el operando destino. Decimos “el lugar especificado por el operando...” ya que no siempre el operando es el lugar donde se encuentra el dato como veremos pronto.

Entonces, la sintaxis de la instrucción MOV es:

MOV x, y ; o sea $x \leftarrow (y)$, "x" se copia con el contenido de "y"

Donde "x" es el operando destino e "y" el operando fuente.

Ahora veremos como se componen los operandos que representan a "x" e "y" en la instrucción MOV ya que las variantes son muchas.

Un registro cualquiera (cualquier byte de la RAM interna de 00_h a $7F_h$ y cualquier registro del área SFR) puede hacer las veces de operando destino, y como operando fuente una constante.

Pongamos unos ejemplos para aclarar este concepto:

MOV A, #03h ; $A \leftarrow 03_h$

Luego de esta operación el contenido de ACC pasa a ser "3".

En la siguiente, supongamos que el banco de registros internos (R0 a R7) seleccionado es el primero (posiciones de memoria de 00_h a 07_h).

MOV R1,#02h ; $R1 \leftarrow 02_h$

Notemos aquí que esta última instrucción es equivalente a:

MOV 1, #02h ; $1 \leftarrow 02_h$

Con el número "1" estamos indicando el contenido de la 2ª posición de memoria de la RAM interna que corresponde al registro R1 del primer banco.

Distinto es el caso si el banco de registros seleccionado es el 2º (posiciones 08_h a $0F_h$) , entonces el equivalente a R1 es la posición de memoria 09_h

MOV 9,#02h ; $R1 \leftarrow 02_h$ perteneciendo R1 al 2º banco de registros.

De esta forma vemos que es posible mover inmediatamente (direccionamiento inmediato) un valor constante a cualquier registro de memoria. De la misma forma, es decir, a través de una dirección de memoria, podemos acceder al ACC, al acumulador B, a los puertos de 8 bits P0 a P3, etc., la instrucción:

MOV P1, #00000001$_b$; $P1 \leftarrow 01_h$ pone en "1" el bit 0 del port 1 y los demás en cero.

Y ...

MOV 90h,#00000001$_b$; hace exactamente lo mismo que la anterior.

También podemos utilizar como operando fuente a otro registro, es decir, el contenido de un registro puede copiarse en otro utilizando la instrucción MOV, veamos:

MOV R1, #07h ; R1 ← 07_h , supongamos del banco 0.

MOV A, R1 ; A ← (R1), ACC se carga con el contenido de R1, es decir 07_h

De la misma forma:

MOV 1, #07h ;al igual que antes carga R1 del banco 0 con 07_h

MOV 0E0h, 1;$E0_h$ es la posición del SFR correspondiente al acumulador.

Si bien el resultado es el mismo, no lo son los código de Operación de **"MOV A, R1" y "MOV 0E0h, 1"**. Recomendamos consultar los apéndices con las tablas de los código de Operación.

Continuando con las variantes del MOV, ahora veremos como acceder en forma indirecta mediante un registro.

La idea es utilizar a un registro en particular (R0 o R1 únicamente pueden ser estos registros) para apuntar a una posición de memoria, la cual puede hacer las veces de fuente o destino de los datos. El inconveniente (o ventaja, como veremos luego) es que previamente se debe cargar al registro que servirá de apuntador (R0 o R1), luego se puede hacer el MOV en cuestión, veamos:

MOV 3Fh, #08h ; $3F_h$ ← 8, en la posición de memoria $3F_h$ queda almacenado en Nº 8

MOV R0, #3Fh ; R0 ← $3F_h$, hacemos que R0 "apunte" a $3F_h$

MOV B, @R0; B ←((R0)), en el acumulador B se copia el contenido de lo

; apuntado por R0.

Luego de esta secuencia de instrucciones el contenido del acumulador B pasa a ser 8 (no $3F_h$), ya que el modo de direccionamiento utilizado es el indirecto (recordar el significado de "@"...).

Como dijimos anteriormente, el registro apuntador puede utilizarse como fuente o destino, de esta forma es válido escribir (suponiendo que no se modificó el contenido de R0):

MOV @R0, #0E1h ; (R0) ← $E1_h$, coloca $E1_h$ en la posición apuntada por R0.

Este mismo modo puede utilizarse para acceder a memoria externa de datos, esto se hace utilizando **MOVX** (por MOVe eXternal) en lugar de MOV. Sin embargo el único registro que puede operar como fuente o destino directo de acceso a memoria externa es el ACC. Como registros de indirección se pueden utilizar R0, R1 y el DPTR (compuesto como ya vimos por DPL y DPH). Esto permite acceder a dispositivos conectados al bus multiplexado del microcontrolador. La sintaxis de estas instrucciones es la siguiente:

MOVX A ,@Ri ; A ← ((Ri)) en memoria externa.

MOVX @Ri, A ; (Ri) ← (A).

MOVX A,@DPTR ; A ← ((DPTR)).

MOVX @DPTR,A ; (DPTR) ← (A).

Ri significa que puede ser R0 o R1 (i=0 o i=1). Estas instrucciones producen un comportamiento diferente al de otras sobre los buses del microcontrolador.

6.1.2 Instrucciones de intercambio.

Se trata de instrucciones que permiten intercambiar los contenidos de los registros que se especifican como operandos, la única salvedad es que uno de los registros siempre debe ser el ACC. Las instrucciones son (donde Rn es uno de los registros R0 a R7 del banco activo y Ri puede se R0 o R1):

XCH A, Rn ; (A) ← (Rn) y (Rn) ← (A), ACC y Rn intercambian sus respectivos
; contenidos. (Rn) ↔ (A).

XCH A, byte ; ídem entre ACC y una posición de memoria de la RAM interna.

XCH A, @Ri ; A ((Ri)) y (Ri) (A), el lugar apuntado por Ri intercambia contenido
; con ACC.

XCHD A, @Ri ; ACC y el registro apuntado por Ri intercambian nibbles menos significativos.

Veamos un ejemplo para clarificar:

MOV **A, #12h** ; ACC contendrá 12_h

MOV **R0, #3Fh** ; R0 apunta a la RAM $3F_h$

MOV **@R0, #34h** ; guardamos 34_h en la posición de RAM interna $3F_h$

XCHD **A, @R0**;luego de esto ACC = 14_h y el contenido de la RAM $3F_h$ = 32_h

Esta operación está pensada principalmente para la manipulación de dígitos en BCD.

6.1.3 Tablas de Búsqueda.

Retomemos el ejemplo de Direccionamiento indexado:. En la página 27 hemos vista que deseamos medir temperatura por medio de una termocupla cuya tensiones amplificada y pasada por un conversor analógico a digital de 8 bits de salida. La misma es alineal y se emplea una tabla de conversión.

Para la conversión se emplearán dos instrucciones (según el caso):

6.1.3.1 MOVC A,@A + DPTR

En la cual DPTR apunta al inicio de la tabla de conversión. En esta tabla de conversión el programador ha colocado el valor de temperatura (en un byte) correspondiente a cada byte de salida del conversor. Esta tabla se podrá realizar en forma empírica o empleando la completa relación entre tensión y temperatura que rige a la termocupla.

En el Acumulador se halla el valor medido a la salida del conversor. La instrucción MOVC genera un código de operación que produce la lectura de la memoria de código y se llevará al acumulador el contenido de la posición de memoria de programa cuyo contenido es apuntado por el origen de la tabla de conversión y el desplazamiento esta dado por el valor medido.

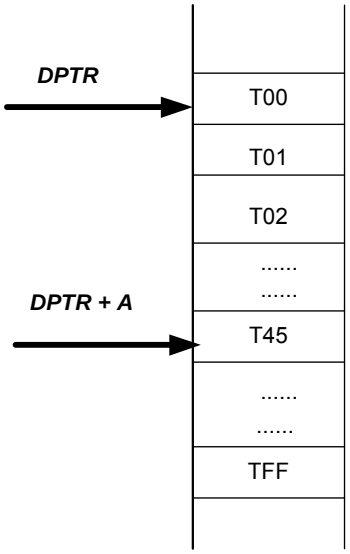


Fig. 21. Uso del DPTR para acceder a una tabla de búsqueda.

Así, supongamos que el valor obtenido a la salida del conversor ha sido 45H. Ese valor se hallará en el acumulador y se inicializará DPTR al comienzo de la tabla. Al ejecutar **MOVC A,@A + DPTR** se trasladará al acumulador el valor T45 que corresponde a la temperatura estimada para esa tensión de salida de la termocupla.

6.1.3.2 MOVC A,@A+PC

Esta instrucción opera en forma similar, salvo que el contador de programa es empleado como puntero y la tabla es accedida normalmente a través de una subrutina.

Análogamente al caso anterior, el acumulador contiene la salida del conversor se encuentra en el Acumulador. Al llamar a la subrutina, en la misma se accederá a la tabla de conversión.

```
        MOV   A,DATO_CONVERSOR      ; Se lee el resultado del la conversión A/D
        CALL  CONVERSION_A_TEMP
        ........
CONVERSION_A_TEMP:
        INC   A                     ; Para ajustar por el byte que  corresponde a RET
        MOVC  A,@A+PC               ; Se trae el valor de la tabla
        RET
TABLA:  DB    T00,T01,T02,.....,T45,.....TFF
```

6.1.4 Pila

Hasta aquí hemos visto el manejo de datos en memoria con el grupo MOV de instrucciones, sin embargo, no es el único tipo de instrucciones para tal fin.

En este grupo están, las instrucciones para el intercambio entre registros y el manejo de la pila (Stack).

Ya que la pila es un concepto muy importante, repasaremos su funcionamiento para el caso de los microcontroladores de la familia MCS®-51.

Para entender el funcionamiento del STACK pensemos que se trata de un tubo cilíndrico para guardar monedas, cada moneda representa el contenido de un registro que desea guardarse en él.

Al comenzar, con el tubo (STACK) vacío, si guardamos una moneda, el lugar disponible habrá disminuido obviamente, al colocar otra moneda, la única forma de recuperar la primera es quitando la que introdujimos en segundo lugar. Este primitivo método de almacenamiento hace que tengamos inmediatamente disponible el último dato ingresado, el concepto de este mecanismo se conoce como método de pila LIFO (Last In, First Out), el primero en ingresar a la pila será el último en salir.

Dicha pila es una zona de la memoria RAM interna y apuntada por un registro especial reservado exclusivamente para esta tarea (llamado Stack Pointer o SP), luego de producirse un reset del microcontrolador, por el motivo que fuese, el registro SP "amanece" con el número 07_h, es decir, apunta a la 8ª posición de RAM interna, que coincide con el registro R7 del banco 0 de registros.

Respecto de la pila, debemos destacar:

1. El puntero a la pila se ***Incrementa*** al guardar datos en la pila (a diferencia del 8088 en que se decrementaba).

2. Antes de guardar un dato en la pila, primero se incrementa SP y luego se guarda. Por ello es que si bien después del Reset, SP apunta a la posición de memoria 7, al guardar el primer dato, apunta a la posición 8 es decir R0 del banco de registros 1.

3. Es recomendable inicializar el Puntero a la Pila a una zona de memoria de datos que no interactúe con registros, por ejemplo a la zona de memoria de datos alta (por medio de MOV SP,#70H) .

4. La pila emplea la memoria interna que podrá tener 128 ó 256 bytes como máximo. Por ello es que se debe ser muy cuidadoso en el uso de la pila a fin de no sobrescribir variables.

El comportamiento del SP depende de varios factores, estos son exactamente:

1. **Modificación directa del registro SP**. El registro SP puede cargarse directamente con una instrucción MOV, de esta forma podemos hacer que apunte a la zona de memoria que queremos utilizar como STACK

2. **Colocar o sacar un dato de la pila**. Es la utilización de las instrucciones PUSH (empujar) para guardar y POP (hacer saltar), dos instrucciones, muy útiles por cierto, para colocar y sacar de la memoria de la pila el contenido de otras posiciones de memoria (registros).

   Es de hacer notar que se puede hacer PUSH o POP con cualquier posición de memoria en forma directa, es más, se puede hacer un PUSH del mismo SP.

   Al efectuar un PUSH, se incrementa el contenido del SP y al hacer un POP se decrementa. Por ejemplo:

   ```
   PUSH DPL        ; se incrementa SP y luego vuelca el contenido de DPL a la pila.

   PUSH DPH        ; ídem anterior con DPH.
   ```

   Esto es útil cuando se desea utilizar el DPTR y no perder el contenido actual, entonces lo guardamos momentáneamente en el STACK y luego lo recuperamos con:

   ```
   POP DPH         ; coloca en DPH lo que se encuentra en el tope del STACK,
                   ; decrementa SP.
   POP DPL         ; ídem anterior(recordar que se decrementó), vuelve a
   ```

; decrementar SP.

Nótese que para retirar datos del STACK hay que hacerlo en el orden inverso a como fueron ingresados. Es particularmente crítico el manejo del STACK y una de las causas más frecuentes de errores de funcionamiento del software, se debe tener cuidado extremo al colocar y retirar datos del STACK.

3. **Llamados a subrutina**. Una llamada a subrutina se produce al utilizar la instrucción apropiada (CALL). Al hacer esto, automáticamente se guardan en las posiciones siguientes a la apuntada por SP los dos bytes de la dirección que corresponde a la próxima instrucción que se debe ejecutar al retornar de la subrutina llamada. Si lo pensamos un poco esto es lógico, ya que es una forma cómoda de recordar el lugar donde continuará el programa al finalizar la ejecución de la subrutina. Ahora, una vez retornado el control al programa en el punto indicado por la dirección guardada en la pila, al no ser necesaria ya esta dirección, el registro SP vuelve a apuntar al lugar donde se encontraba previo al llamado a subrutina.

   Al colocar datos en la pila, el SP se incrementa apuntando al último dato ingresado, y al sacar datos de la misma, el SP se decrementa apuntando un byte antes del primero disponible en el STACK.

4. **Interrupciones:** En la atención de interrupciones (mecanismo que veremos en detalle más adelante), la situación es similar a la de una llamada a subrutina, solo que en general, sucede asincrónicamente (en cualquier momento de la ejecución del programa). Es como un CALL asincrónico a una subrutina. Nada impide utilizar la pila en una interrupción, solo que al llegar al momento del retorno, el puntero SP se debe encontrar en el sitio adecuado, es decir que se deberán hacer tantos POPs como PUSHes.

6.2 Instrucciones Aritméticas.

Esta familia de MPU permite realizar directamente las cuatro operaciones aritméticas fundamentales, suma, resta multiplicación y división, además permite el ajuste decimal para convertir números en hexadecimal a BCD.

Las operaciones de suma y resta permiten utilizar los modos de direccionamiento por registro, directo, indirecto e inmediato para uno de los operandos, el otro operando es siempre el ACC y además este último es quién recibe el resultado e la operación en todos los casos.

En particular las operaciones de adición pueden hacerse incluyendo el bit de acarreo (suma con acarreo) o no, mientras que en la sustracción siempre[14] se utiliza el acarreo de resta (borrow, que no es otra cosa que el complemento del acarreo), de esta forma tenemos cuatro modos de direccionamiento, dos operaciones y una de ellas con y sin acarreo. Total: doce variantes.

6.2.1 Suma sin CARRY:

La sintaxis de las variantes de esta instrucción son:

```
ADD A, Rn       ; suma el contenido del ACC con el contenido del registro Rn.
ADD A, @Ri      ;ídem con el contenido de lo apuntado por Ri.
ADD A, byte     ;ídem anterior con el contenido de la posición de memoria "byte".
ADD A, #dato    ;ídem anterior con el "dato" inmediato.
```

[14] Esto quiere decir que NO existe la resta sin borrow.

Cualquiera de estas instrucciones, si se produce un desborde de suma (o sea, la suma del valor contenido en el acumulador más el operando a sumar supera el valor 255) hace que se ponga a "1" el flag de CARRY .

6.2.2 Suma con acarreo:

La sintaxis de las variantes de esta instrucción son:

ADDC A, Rn ; suma el CARRY, el contenido del ACC y el contenido del registro Rn.

ADDC A, @Ri ;ídem con el contenido de lo apuntado por Ri.

ADDC A, byte ;ídem anterior con el contenido de la posición de memoria "byte".

ADDC A, #dato ;ídem anterior con el "dato" inmediato.

Cualquiera de estas instrucciones afectan el contenido del acarreo.

6.2.3 Sustracción:

La sintaxis de las variantes de esta instrucción es:

SUBB A, Rn ; resta contenido del registro Rn al ACC con BORROW.

SUBB A, @Ri ; ídem con el contenido de lo apuntado por Ri.

SUBB A, byte ; ídem anterior con el contenido de la posición de memoria "byte".

SUBB A, #dato ; ídem anterior con el "dato" inmediato.

6.2.4 Producto y cociente.

Las operaciones de multiplicación y división solo son con números enteros y se utilizan única y exclusivamente los dos acumuladores A y B.

MUL AB ; realiza el producto entre el número contenido en A por el contenido en B,
; el resultado de la operación se guarda en B (byte más significativo) y en
; A (byte menos significativo).

DIV AB ; divide A por B,el resultado cociente queda en A y el resto de la división en B

6.2.5 Incremento y decremento.

Con respecto a las operaciones de incremento y decremento, pueden aplicarse a registros, al ACC, a variables en memoria mediante indirección con R0 y R1 e inclusive al DPTR. Este último se puede incrementar desde 0000_h hasta $FFFF_h$ pero la operación de decremento no puede aplicarse directamente al DPTR[15].

Parecería que el incremento consiste en sumar 1 a un registro o variable y el decremento en restar 1, pero como diferencia trascendental entre INC y DEC con las operaciones de Suma y Resta es que ni el Incremento ni el Decremento afectan ningún flag.

Entonces tenemos:

[15] Justamente una de las ventajas que presentan algunos de los “8051 de segunda generación” es la posibilidad de decrementar el DPTR

```
INC     A       ;incrementa  el contenido del ACC.
INC     Rn      ;ídem registro Rn del banco activo.
INC     byte    ;ídem de la posición de memoria "byte".
INC     @Ri     ;ídem con la posición de memoria apuntada por Ri.
INC     DPTR    ;Incrementa los 16 bits del DPTR.
```

Y las de decremento:

```
DECA            ;decrementa  el contenido del ACC.
DEC     Rn      ;ídem registro Rn del banco activo.
DEC     byte    ;ídem de la posición de memoria "byte".
DEC     @Ri     ;ídem con la posición de memoria apuntada por Ri.
```

Nótese que no existe decremento para el DPTR[16].

Todas las instrucciones Aritméticas se ejecutan en un ciclo de máquina, salvo:

- **INC DPTR (2 Ciclos de máquina)**
- **MUL AB (4 Ciclos de máquina)**

6.2.6 Ajuste Decimal.

Lo último que queda por estudiar del grupo de instrucciones aritméticas es el ajuste decimal del contenido del ACC para operar con números en BCD. Esto es, el número contenido en el acumulador es llevado a dos dígitos en BCD luego de sumar (ADD o ADDC) dos números que si lo son. Veamos un ejemplo:

Supongamos que tenemos dos magnitudes decimales 12 y 39 y las deseamos sumar.

```
MOV     A,#12h ;preparo el primer operando para sumar
ADD     A,#39h ;y realizo la suma de dos números en binario.
```

Recordemos que la Unidad Aritmética – Lógica sólo sabe operar con números binarios. Hasta aquí el resultado de la suma es $4B_h$ pero para que sea correcto en BCD debería ser 51_h, esta contingencia se resuelve con:

```
DA      A       ;realiza el ajuste de los dígitos en el ACC.
```

Así, el resultado en ACC es efectivamente 51_h.

6.3 Operaciones con bits.

Esta familia de microcontroladores cuenta con un poderoso conjunto de instrucciones para manipular bits. Por razones de método veremos parte de ellas ahora y el resto en la sección siguiente, entonces, quedará claro el por qué.

---

[16] Como ejemplo, presentamos un pequeño programa que realiza el decremento del DPTR

```
        DEC     DPL             ; Se decrementa la parte menos significativa del DPTR
        MOV     R7,DPL          ; Se copia a R7 utilizado como registro auxiliar
        CJNE    R7,#0FFH,NO_DEC ; Si pasó de 00 a FF se debe decrementar la parte alta. Si no continua.
        DEC     DPH             ; Se decrementa.
NO_DEC: .....
```

Para comenzar, las operaciones principales con bits constan de asignarles un estado, "0" o "1", de esta forma simplemente con dos instrucciones podemos cambiar (fijar) el estado de cualquier bit que permita direccionamiento (recomendamos consultar las tablas del capítulo 2).

Estas instrucciones son:

```
SETB    bit     ;pone en "1" el bit indicado.

CLR     bit     ;pone en "0" el bit indicado.
```

Debe notarse que, estas instrucciones si bien son simples, constan de un código de operación y un operando (un byte adicional) para indicar el bit en cuestión.

Intentaremos aclarar un poco más estos conceptos con los siguientes ejemplos:

```
SETB    ACC.1   ;pone en "1" el 2º bit del ACC.

CLR     P1.3    ;pone en "0" el 4º bit del puerto P1.
```

Este mismo grupo de instrucciones existe para un bit privilegiado, este es el bit de acarreo que cumplirá las veces de acumulador de un bit.

```
SETB    C       ; C = 1.

CLR     C       ; C = 0.
```

Estas dos instrucciones son de un solo byte. Podemos verificar con las siguientes instrucciones los privilegios de C y que hace las veces de acumulador pero en operaciones a nivel de bit:

```
MOV     C, bit  ;copia al C el estado del bit indicado.

MOV     bit, C  ;copia el estado del C al bit indicado.
```

Vemos que se comporta como un equivalente a los MOV A,... y viceversa para las operaciones con registros.

6.4 Operaciones Lógicas.

Dentro de este grupo podremos operar con registros de 8 bits o directamente con los mismos bits que son direccionables dentro del microcontrolador.

Las operaciones que se pueden realizar son las fundamentales dentro del álgebra de Boole. Tendremos entonces la posibilidad de utilizar la adición (OR), multiplicación (AND), negación o complemento (NOT), OR exclusivo (XOR). A diferencia de las operaciones aritméticas no se utiliza únicamente el acumulador, esto nos permite realizar una operación y obtener el resultado directamente en una variable que no sea el acumulador. Las operaciones posibles con esta comodidad son AND, OR y XOR:

```
ANL     A, Rn

ANL     A, byte

ANL     A, @Ri

ANL     A, #dato

ANL     byte, A

ANL     byte, #dato
```

En las cuatro primeras podemos ver que el resultado de la operación queda en el acumulador y en las dos últimas el resultado queda en el byte indicado.

De la misma forma se utilizan:

```
ORL   A, Rn
ORL   A, byte
ORL   A, @Ri
ORL   A, #dato
ORL   byte, A
ORL   byte, #dato

XRL   A, Rn
XRL   A, byte
XRL   A, @Ri
XRL   A, #dato
XRL   byte, A
XRL   byte, #dato
```

Que efectúan las operaciones de OR y XOR respectivamente.

La operación de complemento se aplica exclusivamente al ACC:

```
CPL   A          ;complementa el contenido del ACC.
```

A nivel de bit (variable booleana) podemos realizar también operaciones por el estilo, tenemos:

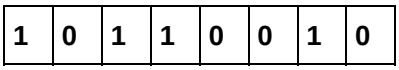
```
ANL   C, bit     ;AND entre C y bit especificado, resultado en C.
ANL   C, /bit    ;ídem anterior pero con el complemento del bit.

ORL   C, bit     ;OR entre C y bit especificado, resultado en C.
ORL   C, /bit    ;ídem anterior pero con el complemento del bit.

CPL   C          ;complementa el C.
CPL   bit        ;complementa el bit especificado.
```

Podemos notar aquí lo conveniente de la notación "/bit", indica que al realizarse la operación, se utiliza el complemento del bit especificado, ahorrándonos un paso intermedio.

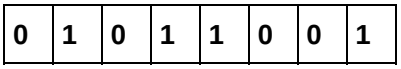
6.5 Rotaciones y desplazamientos

Por último trataremos las rotaciones de bits sobre el acumulador y el acarreo. Efectivamente, el contenido del acumulador se puede rotar (no desplazar, ya veremos la diferencia) a izquierda o derecha, pasando por el flag de acarreo o no. Esto último significa que, el byte que se encuentra dentro del Acumulador puede "rotarse", esto es, los bits que lo componen se copian al bit siguiente a derecha o izquierda según sea el caso y utilizando al C como un bit más. Ahora, la diferencia entre una rotación y un desplazamiento es que en la rotación los bits de los extremos se introducen nuevamente en el ACC por el extremo contrario, ejemplo:

Contenido del ACC antes de una rotación a la derecha

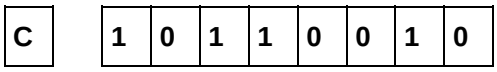


Contenido del Acumulador después de la rotación a la derecha

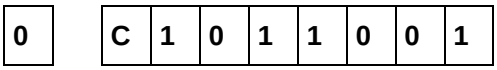


Como podemos observar, al rotar, el bit menos significativo pasa a ocupar el lugar del más significativo. En un desplazamiento, el bit menos significativo " cae al vacío ", se pierde, y el más significativo se completa con "0". Sin embargo, en estos MCU no existe la operación de desplazamiento, pero ésta puede hacerse utilizando el bit C, veamos.

Contenido del Acumulador y acarreo antes de una rotación a la derecha utilizando el C



Contenido del ACC y carry después de la rotación a la derecha



Ahora bien, si antes de realizar la rotación forzamos el valor de C = 0, conseguimos el efecto de un desplazamiento:

```
CLR     C       ;pone el C = 0
RRC     A       ;rota el contenido del ACC a la derecha a través del carry.
```

En definitiva el conjunto de instrucciones para las rotaciones es:

```
RR      A       ;rota a la derecha los bits del ACC sin pasar por C.
RL      A       ;ídem anterior pero a la izquierda
RRC     A       ;rota el contenido del ACC incluido el C a la derecha.
RLC     A       ;ídem anterior a izquierda.
```

6.6 Saltos condicionales e incondicionales.

Ahora veremos las instrucciones que producen desvíos en la secuencia normal del programa, esto es, cambiar el curso de ejecución del programa por algún motivo (condicional o incondicionalmente). Podemos hacer una clasificación de los tipos de bifurcaciones.

Básicamente tenemos: los ***saltos o desvíos incondicionales***, en los que se fuerza a un salto incondicionalmente, porque sí, y ***los desvíos condicionales***, en los que dada una cierta condición, el programa tomará uno u otro camino en caso de cumplirse o no ésta condición. A su vez tenemos dentro de los incondicionales el salto a subrutinas, la diferencia es que en la subrutina se "recuerda" el punto de retorno (para esto se usa la pila). Veamos entonces como primer punto los llamados a subrutinas.

6.6.1 Llamados a subrutina.

Una subrutina es un fragmento de programa dedicado a una función particular, puede suceder que necesite ser llamada desde varios puntos del programa principal en reiteradas oportunidades, de esta forma, dependiendo del punto del programa desde donde fue llamada, la misma subrutina deberá retornar a lugares diferentes. El mecanismo para el retorno automático es, al momento del llamado, guardar en algún lugar seguro (stack) la dirección de retorno. En particular para estos microcontroladores se puede llamar a subrutinas que se encuentren más cerca (en los alrededores de los 2 Kb de programa) o más lejos (saltos dentro del espacio de memoria de programa de 64 Kb).

Las instrucciones de llamado a subrutina son las siguientes:

ACALL **direc**. **11 bit** ;salta a subrutina dentro de límite de los 2 Kb al PC actual.

LCALL **direc. 16 bit** ;ídem dentro del espacio de los 64 Kb.

A la hora de retornar de estos llamados, la instrucción común es:

RET ;retorna de llamado a subrutina.

El comportamiento del microcontrolador es particular en estos llamados y retornos y se resume en los siguientes pasos:

Para la instrucción **ACALL**:

a) Se incrementa en dos bytes el contador de programa (ya que un ACALL ocupa 2 bytes).

b) Se incrementa el SP.

c) Se guarda en el stack el byte bajo del contador de programa.

d) Se incrementa el SP.

e) Se guarda en el stack el byte alto del contador de programa.

f) Los 11 bits menos significativos del contador de programa se cargan con la dirección de la subrutina a la que se está llamando.

Para la instrucción **LCALL**:

a) Se incrementa en tres bytes el contador de programa (ya que un LCALL ocupa 3 bytes).

b) Se incrementa el SP.

c) Se guarda en el stack el byte bajo del contador de programa.

d) Se incrementa el SP.

e) Se guarda en el stack el byte alto del contador de programa.

f) El contador de programa se carga con la dirección de la subrutina a la que se está llamando (16 bits).

Para la instrucción **RET:**

a) La parte alta del contador de programa se carga con el primer byte disponible del stack.

b) Se decrementa el SP.

c) La parte baja del contador de programa se carga con el primer byte disponible del stack (recordar que luego de retirar uno en el punto a), el SP se decrementó en el punto b) apuntando a un nuevo byte).

d) Se decrementa el SP.

6.6.2 Retorno de interrupción.

Existe otra instrucción que retorna automáticamente y se utiliza en las llamadas a rutinas que atienden interrupciones, tema que trataremos en un capitulo aparte, la instrucción es:

```
RETI            ;retorna de atención a una interrupción.
```

6.6.3 Saltos incondicionales.

Ahora veamos los saltos incondicionales, la diferencia con los CALL es que no se guarda en ningún lugar la dirección de retorno. Las instrucciones de salto se llaman JMP (jump, salto) y vienen en tres sabores:

```
LJMP    dirección 16 bits    ;salta a cualquier posición de memoria de programa.
AJMP    dirección 11 bits    ;al igual que el ACALL salto dentro de los 2 Kb.
SJMP    relativo 8 bits      ;salta dentro de la cercanía de los +127 /- 128 bytes.
                             ; (un byte en complemento a dos)
```

Existe un tipo especial de JMP que permite saltar a ejecutar código relativo a un desplazamiento indicado en el ACC:

```
JMP @A+DPTR     ;salta a la posición apuntada por DPTR más el desplazamiento indicado en
                ;el Acumulador.
```

Mecanismo útil para generar una opción del tipo ***DO CASE*** de funciones de un menú principal, por ejemplo:

```
MOV     A, P1       ; ingresamos un dato por el port 1.
ANL     A, #0Fh     ; filtramos los 4 bits menos significativos
CLR     C           ; con estas dos instrucciones se desplaza a izquierda un bit el contenido
RL      C           ; del ACC, efecto: multiplica por dos el contenido del ACC.
```

```
    MOV     DPTR, #TBLA_DE_SALTO    ;cargo dirección base de una tabla de salto.
    JMP     @A+DPTR                 ;salto a una entrada de la tabla de salto.
.....
  TBLA_DE_SALTO:
    AJMP FUNCION_1          ;salta a ejecutar una de las n funciones de la tabla.
    AJMP FUNCION_2
    AJMP FUNCION_3
.....
    AJMP FUNCION_n          ;para el caso presentado n puede ser hasta 16.
```

6.6.4 Saltos condicionales:

Este grupo de instrucciones evalúan una condición que podrá cumplirse o no, en cualquier caso el programa tomará alguno de dos caminos posibles, el salto en caso de cumplirse la condición, o la continuación de la secuencia normal del programa en caso de no cumplirse la condición. Las condiciones evaluables son las siguientes:

```
JZ relativo 8 bits    ; salta al igual que un SJMP si el contenido del ACC = 0. Los 8 bits son
                      ; interpretados en complemento a 2
JNZ relativo 8 bits   ;ídem si el contenido del ACC es distinto de 0.
JC   rel 8 bits       ;ídem si el bit de C = 1.
JNC rel 8 bits        ;ídem si C = 0.
JB bit, rel 8 bits    ;ídem si el bit indicado es = 1.
JNB bit, rel 8 bits   ;ídem si el bit indicado es = 0. P. Ej  JB  P1.4, COND_1
JBC bit, rel 8 bits   ;ídem si el bit indicado es = 1 y luego pone dicho bit = 0.
```

6.6.5 Comparaciones

Además de estas instrucciones de saltos condicionales, también existe un conjunto muy interesante para realizar comparaciones y decrementos automáticos, estos grupos son los CJNE (Compare and Jump if Not Equal, comparar y saltar si no es igual) y DJNZ (Decrement and Jump if Not Zero, decrementar y saltar si no es cero). La pregunta que surge inmediatamente es qué cosa no es igual a qué otra (en el CJNE) y el decremento de qué registro no es igual a cero (en el DJNZ). Pues bien, en los CJNE se puede comparar:

- Al Acumulador contra cualquier otro registro.
- Al Acumulador contra un dato inmediato.
- A cualquier registro Rn del banco activo contra un dato inmediato.
- A un registro apuntado por Ri del banco activo contra un dato inmediato.

En caso de desigualdad en la comparación, la instrucción salta a la posición relativa a un desplazamiento de 8 bits al igual que un JC o JNB. Adicionalmente, en caso de producirse el salto por la desigualdad, el bit de C se ve modificado indicándonos si el primer operando (por ejemplo el ACC) es mayor (C=0) o menor (C=1) que el segundo.

Así la sintaxis para estas instrucciones es:

CJNE A, byte, rel 8 bits ;compara y salta si no son iguales A y la posición de memoria de
;datos "byte".

CJNE A, #dato, rel 8 bits ;ídem con el contenido del ACC y el "dato" inmediato.

CJNE Rn, #dato, rel 8 bits ;ídem entre Rn y el "dato" inmediato.

CJNE @Ri, #dato, rel 8bits ;ídem entre el contenido de lo apuntado por Ri y el "dato" inmediato.

El caso de los DJNZ es más simple, nos permite decrementar automáticamente el contenido de un registro específico y en caso de no llega a cero saltar a la posición relativa de 8 bits indicada (similar al **LOOP** del 8088) Los código de Operación existentes a tal fin son:

DJNZ Rn, rel 8 bits ;decrementa en una unidad el contenido de Rn, si no llega a cero,
;salta al lugar indicado por el desplazamiento relativo de 8 bits.

DJNZ byte, rel 8 bits ;ídem con la posición de memoria RAM "byte".

6.7 Misceláneos.

Si bien el fabricante y algunos autores colocan dentro de otros grupos a los código de Operación que veremos a continuación, nosotros preferimos agruparlos dentro de misceláneos ya que rigurosamente hablando no forman parte de ninguno de los grupos descritos hasta el momento, son tres instrucciones y su sintaxis y función son las siguientes:

CLR A ; ACC = 0.

SWAP A ; intercambia los nibbles dentro del ACC.

NOP ; No Operation (operando nulo), como su nombre lo indica hace nada.

7. Aplicaciones

Sin entrar en complicadas codificaciones expondremos unos ejemplos de aplicación pero que trataremos de analizar a fondo para fijar lo mejor posible los conceptos hasta el momento estudiados.

Presentaremos problemas prácticos y detallaremos su resolución a nivel software y hardware según sea necesario. A su vez, los ejemplos se irán enriqueciendo uno tras otro. Intentaremos ir explicando el proceso de resolución, lo que propondremos es uno de los muchos caminos posibles para la resolución del problema, queda para los lectores el tratar de optimizar o encontrar caminos o soluciones alternativas a los ejemplos planteados.

Como primer problema para familiarizarnos con los códigos de operación de estos microcontroladores y la interacción con dispositivos de accionamiento externo planteamos el siguiente:

7.1 Ejemplo de aplicación 1:

Se cuenta con un dispositivo capaz de medir temperaturas superiores a los 80 ºC de un horno de secado de pinturas para un proceso industrial. La salida de este dispositivo es "0" para temperaturas inferiores a 80 ºC y "1" para temperaturas mayores o iguales. El control se realizará con un 8751 con un cristal de 12 MHz.

Se desea accionar unos contactores (relays accionados por tensión alterna para altas potencias) que controlarán el accionamiento de los resistores de calentamiento del horno (varios kilovatios) y ventiladores de circulación de aire caliente. Debe tenerse en cuenta que se trata de un recinto de dimensiones importantes y posee una gran inercia térmica.

El ciclo de funcionamiento básico es el siguiente:

1 *Al dar arranque al sistema (se supone temperatura ambiente), el transductor de temperatura indicará que ésta se encuentra por debajo de los 80 ºC (indicará "0"), entonces se comienza con el calentamiento del horno.*

2 *Al llegar a los 80 ºC se deberán encender los ventiladores de circulación de aire y se mantendrán conectados los resistores de calentamiento durante aproximadamente 1 minuto (no es necesaria una exactitud extrema).*

3 *Transcurrido el minuto, se apagará el horno y se mantendrán encendidos los ventiladores hasta que la temperatura baje de los 80 ºC, con esto finaliza el ciclo de funcionamiento, para comenzar un nuevo ciclo se debe resetear el sistema.*

Antes de encarar la resolución debe tenerse en cuenta que este no pretende ser un ejemplo de la vida real ni mucho menos, si bien veremos como resolverlo teniendo en cuenta detalles como si lo fuera.

Visto y considerando que se presentan varios inconvenientes dividiremos el problema para su resolución en dos partes:

1. Electrónica digital.
2. Electrónica de aislación y accionamiento.

Es muy importante esta distinción ya que en los ambientes industriales se debe tener mucho cuidado con los accionamientos de potencia y las aislaciones ópticas o galvánicas para evitar interferencias en la electrónica digital de control.

En lo que respecta al diseño digital, por el momento contaremos con que disponemos de la señal de temperatura en el bit 0 del puerto 1 (P1.0), el accionamiento del horno en el bit P1.1 y el de los ventiladores en el P1.2.

Al producirse un reset de encendido, el contador de programa se cargará con la posición de memoria 0000_h a esta posición la llamamos vector de reset o vector de arranque (llamamos genéricamente "vector" a un valor con el que se cargará el contador de programa ante un evento determinado que puede ser como en este caso un reset o también una interrupción, etc.), de esta forma en esa posición de memoria comenzará nuestro programa.

Pero, no podemos extendernos demasiado a partir de esa posición precisamente, porque inmediatamente aparece en 0003_h otro vector destinado a atender una interrupción, en nuestro caso como se trata de un ejemplo sencillo no nos importaría, ya que no manejaremos interrupciones por ahora, de todas formas es bueno acostumbrarse a no utilizar espacio de memoria reservado para aplicaciones específicas. Lo que corresponde en este caso es colocar una instrucción de salto al comienzo de nuestro programa principal para no utilizar el espacio de las primeras posiciones del mapa de memoria de programa que está reservado para los vectores de interrupciones varias (ver apéndice).

1. Luego de comenzado el programa se deberán tomar ciertos recaudos:
2. Colocar el SP (puntero a la pila) a un valor conveniente de ser necesario.
3. Configurar los puertos del microcontrolador como entradas o salidas según enunciado.
4. Inicializar variables que necesiten un valor determinado antes del desarrollo del programa principal.

Por consiguiente para nuestro ejemplo, como no utilizaremos los bancos de registros 1, 2 y 3, solamente el 0, el SP puede quedar en su posición por defecto.

```
;*************************************************************************
;*   PROGRAMA :      EJEMP1.ASM
;*   COMPILADOR : FRANKLIN A51
;*   PLATAFORMA : MCU 8751
;*    PROPOSITO : CONTROL DE HORNO ELECTRICO CON VENTILACION, SE UTILIZAN
;*                        BITS DEL PUERTO P1 COMO ENTRADAS Y SALIDAS
;*************************************************************************
zonda_t     EQU   P1.0          ;aquí definimos nombres amigables para los bits
calienta    EQU   P1.1          ;de P1
ventila EQU   P1.2

            ORG   0000H         ;indicamos al compilador que el inicio del
                                ;ensamblado es a partir de la posición 0000h

Reset:                  ;etiqueta que indica simplemente el vector de reset
            LJMP  Inicio  ;salto al comienzo de nuestro programa

            ORG   0100H         ;punto de entrada a nuestro programa principal,
                                ;ahora el ensamblado es a partir de 0100h
Inicio:
            SETB  zonda_t       ;pone a 1 el bit P1.0, lo configura como
;entrada
            CLR   calienta
            CLR   ventila ;apaga las señales de calentamiento y ventilador
```

En segundo término, para configurar un puerto como entrada basta con poner el bit correspondiente en "1", de esta forma nuestro programa lucirá más o menos así:

Hasta aquí la inicialización de nuestro problema, ahora queda ver como es el comportamiento del lazo principal de control.

Lo primero según nuestro enunciado, es leer el estado de la temperatura, si el transductor de temperatura indica por debajo de los 80 grados, encenderemos el calentador. Luego, una vez encendido, al llegar a los 80 grados, debemos encender los ventiladores y mantener conectado el calentador un minuto más. Y aquí aparece un detalle interesante, ¿cómo medimos un minuto? Bueno, existen varias formas de hacerlo, más o menos complicadas o exactas, para nuestro caso, al no necesitarse una exactitud extrema, podemos emplear un método sencillo y eficaz para esta aplicación, se trata de que el microcontrolador no haga nada durante la cantidad suficiente de ciclos de máquina hasta completar un minuto. Ya que conocemos la frecuencia de reloj, y la cantidad de ciclos del mismo que tarda un ciclo de máquina, podemos determinar la cantidad de ciclos de máquina para 1 minuto.

De esta forma tenemos que un ciclo de máquina consume 12 períodos de reloj, si este es de 12 MHz, el ciclo de máquina tiene entonces una duración de 1 □Seg.

Por consiguiente, basta con escribir un fragmento de código que ejecute o "consuma" 60 millones de ciclos de máquina y allí tenemos nuestro minuto.

Volvemos a remarcar aquí que este es un método bastante rudimentario como tal, pero el objetivo es notar lo fácil que es implementar lazos repetitivos con unos pocos códigos de operación.

La implementación de la porción del programa que realiza la espera de 1 minuto la haremos como subrutina para ejemplificar el uso del CALL, mientras desarrollamos el programa principal en el momento que sea necesario haremos la llamada pertinente a la subrutina para el retardo y terminaremos de codificar el programa principal, luego codificaremos la subrutina.

Continuando entonces con el desarrollo, el código para verificar el estado del horno puede ser:

```
;*********************************************************************

Main:                                   ;Comienzo del lazo principal

        JB zonda_t, Main ;mientras no baje de 80 grados, se queda en el
                         ;lazo con MAIN.
           SETB  calienta               ;enciende el calentador
Calentando:                             ;se queda aquí hasta alcanzar los 80 grados
           JNB   zonda_t, Calentando

           SETB  ventila        ;enciende los ventiladores
           CALL  un_minuto              ;llama a la rutina que espera un minuto
           CLR   calienta               ;apaga el calentador
Loop:
           JB    zonda_t, Loop ;mientras no baje de 80 grados, looping....
           CLR   ventila        ;apaga los ventiladores
           SJMP  $[17]                  ;fin del programa, para salir de este estado
                                        ;es necesario un reset.

;*********************************************************************
```

Como puede apreciarse el código principal del programa es bastante compacto, veamos entonces para finalizar, la rutina de retardo:

[17] En este ejemplo aparecerá el símbolo $. Éste es el llamado contador de ubicación o location counter. El mismo equivale a decir “aca” y debe interpretarse como:

JMP $ *Equivale a* ACA: JMP ACA.

```
;*************************************************************************
un_minuto:  ;<--- Rutina
; PROPOSITO : Genera un retardo de aproximadamente un minuto
; REGISTROS : utiliza R4, 5, 6 y 7
; REQUIERE  : nada
; DEVUELVE  : nada
; OBS.      :
;*************************************************************************

                PUSH  R4
                PUSH  R5
                PUSH  R6            ;salvamos en la pila el contenido de los registros
                PUSH  R7            ;a utilizar

                MOV   R4,#60        ;ponemos valores de cuenta en los registros
Loop4:                      ;de manera de generar lazos que totalizan
                MOV   R5,#10        ;un retardo de aproximadamente 1 minuto
Loop3:
                MOV   R6,#200
Loop2:
                MOV   R7,#250
Loop1:
                DJNZ  R7, Loop1     ;repite 250 veces, total 500 microsegundos
                DJNZ  R6, Loop2     ;repite 200 veces el lazo de 500 microseg.=100 mseg
                DJNZ  R5, Loop3     ;repite 10 veces lazo de 100 mseg = 1 segundo
                DJNZ  R4, Loop4     ;repite 60 veces 1 segundo = 1 minuto

                POP   R7            ;restauramos el estado de los registros utilizados
                POP   R6
                POP   R5

                RET                 ;retorna al punto donde fue llamada la rutina
;*************************************************************************
```

En particular para saber el tiempo que demora la ejecución de un código de operación cualquiera, recomendamos consultar la tabla correspondiente en el apéndice. De todas formas, los códigos de operación consumen 1, 2 o 3 ciclos de máquina. Con un reloj de 12 MHz cada ciclo de máquina es de 1 □seg.

Ahora notaremos algunos detalles atenientes a la forma de escribir el programa.

1. *Es de muy buena costumbre comentar lo más posible las líneas de código que se van escribiendo, en casi la totalidad de los compiladores esto está permitido anteponiendo al comentario un punto y coma **";"** como puede notarse en el programa de ejemplo.*

2. *Por lo general se suelen escribir por separado (antes o después) las subrutinas del programa principal por una cuestión de claridad e interpretación, al igual que en el reciente ejemplo de aplicación. Es recomendable (aunque no excluyente) escribirlas <u>antes</u> así el compilador al barrer secuencialmente el archivo fuente, al momento de ser invocadas, ya analizó las subrutinas. Simplemente es más rápida la compilación.*

3. *También es de notarse la utilización de "etiquetas" para indicar puntos a donde el programa debe salta. Dependiendo del compilador que se utilice para generar el programa ejecutable del micrcontrolador, la sintaxis varí. En nuestro ejemplo las etiquetas van terminadas por los dos puntos **":"**.*

4. *Otro detalle interesante es la utilización de la directiva "EQU", que nos permite asignarle nombres personalizados a variables, posiciones de memoria en general y bits en particular*

que deseemos nombrar con algún nombre en especial. Esto facilita la interpretación y el posterior mantenimiento del programa.

5. *En la práctica se buscará algún método más efectivo para computar tiempos, pues es un desperdicio total del poderío del microcontrolador emplearlo exclusivamente para hacer una demora ejecutando lazos.*

Por último tenemos la directiva "ORG" para indicar el nuevo origen de ensamblado del programa, al encontrar esto, el compilador comienza a ensamblar código a partir de la posición especificada. De esta forma es posible ubicar porciones de código o tablas de datos en la zona del mapa de memoria de programa que se quiera.

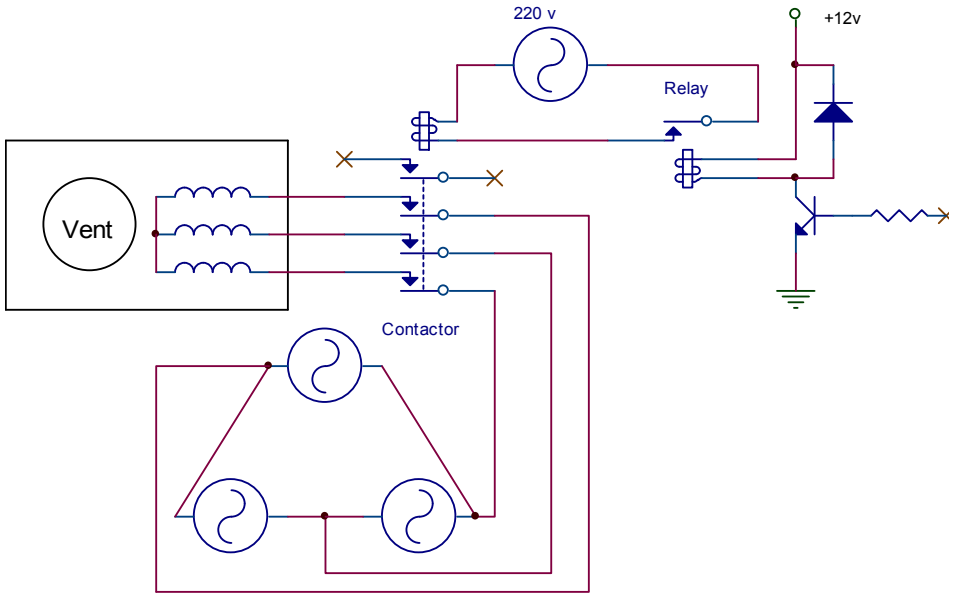
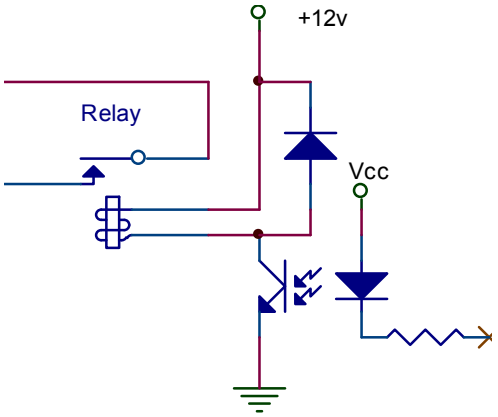


Fig. 22. Esquema de conexión de una salida del microcontrolador.

A continuación, queda definir cómo será la conexión de los dispositivos que debe controlar el MPU. Como mencionamos anteriormente, para los accionamientos de potencia se utilizan contactores, a su vez los contactores deben ser accionados con relays y estos con transistores.

Analicemos un poco el mecanismo de accionamiento. El puerto P1.2 al ponerse a "1" polariza la base del transistor y este se satura llevando de su colector cerca del potencial de tierra, esto hace circular corriente por la bobina del relevador y produce su accionamiento, al cerrarse el contacto del mismo circula corriente por el bobinado del contactor y este cierra el circuito para la alimentación de los ventiladores.

Para mejorar aún más la separación de los circuitos de potencia y digitales se puede recurrir a la aislación óptica:



Debe tenerse en cuenta para este caso en particular que el accionamiento se realiza con un "0" y no con un "1", para que sea como el circuito anterior se debe colocar un optoaislador PNP.

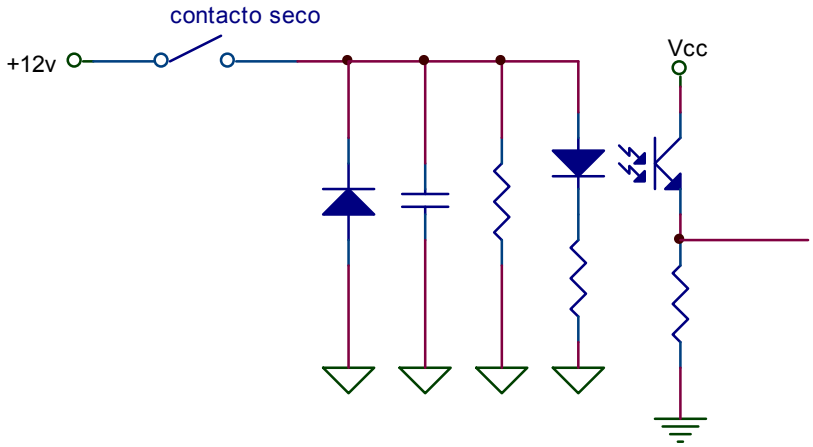
Existen hace ya algún tiempo dispositivos de tecnología T-MOS, capaces de manejar tensiones del orden de los kilovolt y corrientes de cientos de amperios, que pueden ser accionados por señales provenientes de un MPU directamente o por fibra óptica, pero generalmente, por su elevado costo, su uso es restringido a aplicaciones muy particulares.

En principio puede parecer exagerado extremar las precauciones, pero nunca están de más a la hora de diseñar un sistema sólido y confiable ante interferencias que pueden ingresar por las fuentes de alimentación de los relevadores y contactores.

En particular para el accionamiento de los relevadores existe una familia de dispositivos especialmente diseñados "ad hoc" que incorpora el diodo de protección de sobretensiones en la bobina de accionamiento, estos son los ULN2000 a 2004.

Para finalizar, veremos la forma de realizar una entrada desde un dispositivo externo.

Para el caso del sensor de temperatura supongamos que el mismo entrega a su salida "un contacto seco", esto es el equivalente a un interruptor, supongamos que por debajo de 80 grados esta abierto y por arriba de 80 está cerrado, de esta manera lo conectaríamos así:



Podemos ver como al cerrarse el contacto el diodo del optoaislador produce el corte del transistor PNP el MCU "ve" un "0" en el pin del port P1.0. El resto de los componentes hacen la protección contra sobretensiones y permiten un "CLAMPING" de la malla de entrada a un valor limitado de tensión.

Hasta aquí la resolución del problema planteado, con él hemos aprendido:

- ***Escritura de un programa fuente.***
- ***Procedimiento general de inicialización del programa principal.***
- ***Instrucciones de llamada a subrutinas.***
- ***Instrucciones de iteración.***
- ***Lectura y escritura de bits en puertos de I/O.***
- ***Toma de decisiones en base a los estados obtenidos.***
- ***Conexión de dispositivos de accionamiento externo y su aislación.***

7.2 Ejemplo de aplicación 2

Como segundo ejemplo veremos ahora un problema para ejemplificar la conexión de dispositivos al bus multiplexado del MPU.

Se desea implementar un sistema que mediante una tabla de conversión de 256 valores transforma un dato de 8 bits en otro, también de 8 bits. Se dispone entonces de 8 líneas de entrada y 8 de salida, las mismas deben implementarse sobre un bus bidireccional sobre el puerto P1.

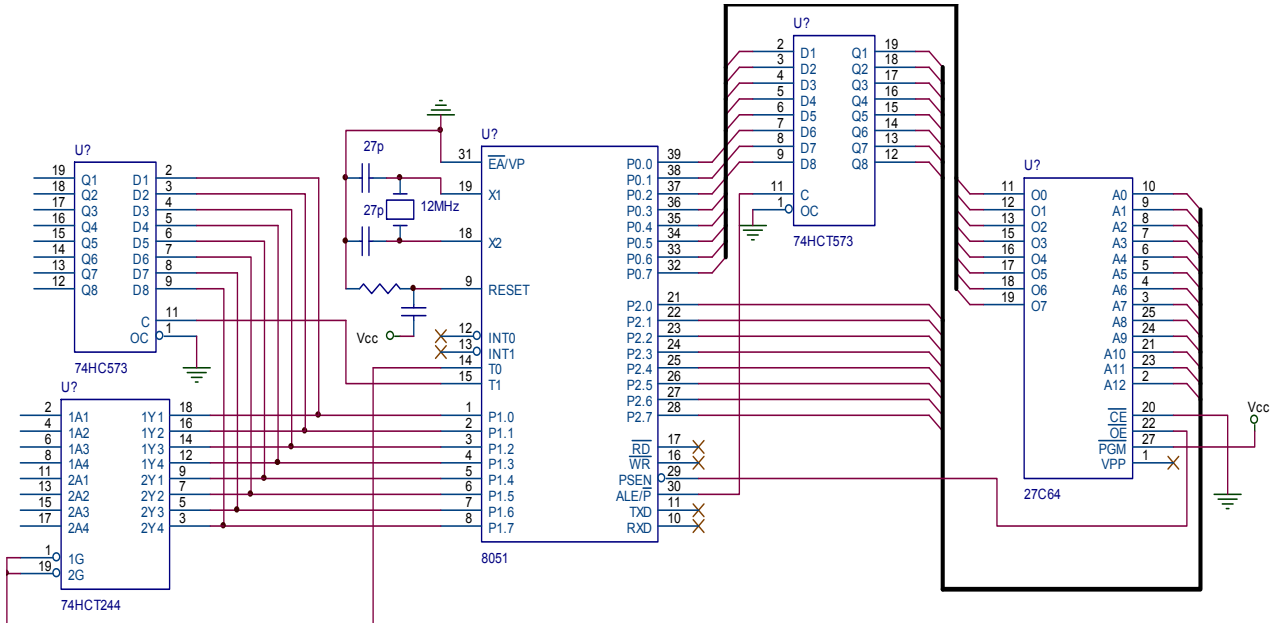
El programa debe leer permanentemente el puerto P1, realizar la conversión del dato mediante la tabla mencionada y volcar el dato convertido sobre el mismo puerto. Se cuenta para el desarrollo con un 80c31 y una memoria EPROM 27c64.

En primer término y a diferencia del problema anterior debemos hacer uso de las capacidades del bus multiplexado para acceso a memoria de programas externa.

A modo de ejercicio implementaremos un bis bidireccional de 8 bits para leer y escribir sendas líneas de I/O utilizando el P1; como líneas de control para este bus haremos uso de líneas adicionales que tomaremos del P3.

La resolución propuesta para el hardware del ejemplo puede ser la siguiente:

Como podemos ver, se habilita el acceso externo (EA#) y el STROBE de lectura de la memoria de programa es PSEN#, contamos también con la línea ALE para retener la parte baja del bus de direcciones sobre el bus multiplexado del MPU.



Con respecto al bus bidireccional sobre el puerto P1, notamos la existencia de dos dispositivos, un LATCH de 8 bits 74hc573 para la salida, y un óctuple buffer de 3 estados para la entrada. La idea es, mientras está habilitado uno de ellos, el otro permanece inactivo. Desde el punto de vista del 573 de salida, cuando retiene un dato, el buffer de entrada debe estar en tercer estado (alta impedancia). Por otro lado, cuando se está leyendo el buffer, no se debe producir el "latcheo" sobre el 573, sino se almacenará el dato entrante en forma directa.

Para llevar a cabo la señalización correspondiente utilizaremos dos líneas del P3, una para controlar la retención de datos en el 573 y otra para poner la salida del buffer de entrada en alta impedancia a voluntad.

El Programa:

La inicialización del programa es similar a la del ejemplo de aplicación 1 que viéramos con anterioridad, tenemos entonces:

```
;************************************************************************
;*
;*        PROGRAMA : EJEMP2.ASM
;*   COMPILADOR : FRANKLIN A51
;*   PLATAFORMA : MCU 80c31
;*    PROPOSITO : CONVERSION DE DATOS DE 8 BITS POR PUERTO BIDIRECCIONAL
;*
;************************************************************************


in              EQU   P3.4    ;aquí definimos nombres amigables para los bits de

out             EQU   P3.5    ;de P1


                ORG   0000H           ;indicamos al compilador que el inicio del

                                      ;ensamblado es a partir de la posición 0000h


Reset:          ;etiqueta que indica simplemente el vector de reset

                LJMP  Inicio  ;salto al comienzo de nuestro programa
```

```
ORG 0100H   ;punto de entrada a nuestro programa ;principal,
```

```
                        ;ahora el ensamblado es a partir de 0100h
Inicio:
        CLR   in        ;deshabilita 244
        CLR   out       ;deshabilita 573
```

Nuestro programa principal (LOOP principal) simplemente debe:

- ***Deshabilitar el 573.***
- ***Habilitar el 244.***
- ***Leer el dato entrante.***
- ***Deshabilitar el 244.***
- ***Llamar a una rutina que devuelva el valor convertido.***
- ***Poner el dato en P1.***
- ***Accionar el "latcheo" en el 573.***
- ***Deshabilitar el 573.***
- ***Volver a comenzar.***

Entonces:

```
Loop:
        SETB  in        ;permitimos la entrada del dato
        MOV   A,P1      ;lo guardamos en ACC
        CLR   in        ;deshabilitamos entrada
        CALL  Convert   ;llamamos a rutina de conversion con el dato en ACC
        MOV   P1,A      ;lo ponemos sobre el P1
        SETB  out       ;
        CLR   out       ;producimos el latcheo
        SJMP  Loop      ;volvemos a comenzar
;*************************************************************************
```

La mayor dificultad , por llamarla de alguna manera y ya veremos por qué, reside en el manejo de una tabla. Notemos antes un detalle importante, el dato a convertir se encuentra en el ACC antes de llamar a la rutina de conversión, cosa que aprovecharemos para utilizar el OPCODE “MOVC A,@A+DPTR”:

```
;*************************************************************************
Convert:    ;<--- Rutina
; PROPOSITO : convierte dato de tabla
; REGISTROS : ACC
; REQUIERE  : dato a convertir en ACC
; DEVUELVE  : dato convertido en ACC
; OBS.      :
;*************************************************************************

                MOV   DPTR,#Tabla ;carga la direccion de la tabla de datos
MOVC A,@A+DPTR  ;levanta dato de tabla
                RET               ;retorna al punto donde fue llamada la rutina
;*************************************************************************
Tabla:
                DB      dato0,dato1,dato2, ... ,
                DB      ....
                .
                .
                .
                DB      dato253,dato254,dato255
;*************************************************************************
```

Como podemos apreciar el DPTR se carga con la posición de comienzo de la tabla y el ACC proporciona el offset para acceder al dato a convertir, luego de la rutina puede verse la forma de implementar la tabla, simplemente se definen bytes (DB) uno a continuación de otro. Esta es una forma muy rápida de implementar (con algunas restricciones) el cálculo de alguna función trigonométrica por ejemplo, cosa que hecha con los algoritmos tradicionales consumiría tiempo de procesamiento.

8. Temporizadores y contadores, programación y configuración.

Los monoestables y astables basados en la carga y descarga de capacitores a través de resistores (circuitos R-C de carga y descarga) son simples de configurar e implementar pero tienen como inconveniente la incerteza de la tolerancia de los componentes y el cambio de valores producidos por la temperatura y envejecimiento. Además para altas constantes de tiempo se produce una incerteza adicional producida por la intersección casi asintótica de la tensión del capacitor con el nivel de umbral que producirá una transición de la salida.

En Técnicas Digitales I hemos visto que un conjunto de flip-flops conectados en cascada pueden emplearse para contar tiempo utilizando una base de tiempo estable (oscilador a cristal por ej.) y luego mediante subdivisiones de frecuencia, obtener los tiempos deseados.

En los microcontroladores de la familia MCS®-51 existen temporizadores programables que nos permitirán contar eventos externos o tiempo. Dependiendo del microcontrolador en cuestión (8051, 8052, etc.) dispondremos de 2 ó 3 temporizadores con características disímiles.

Cada uno de los dos temporizadores comunes a todos los microcontroladores operan y cada uno se programa mediante cuatro registros especiales[18]. Estos registros forman parte del área de registros de funciones especiales y solo uno de ellos es direccionable a nivel de bit.

A continuación veremos un diagrama que muestra la interrelación entre las patas de conexión del puerto P3 con propósito de contar eventos externos, los registros dedicados de los temporizadores y circuitería especial a los efectos del funcionamiento de los temporizadores.

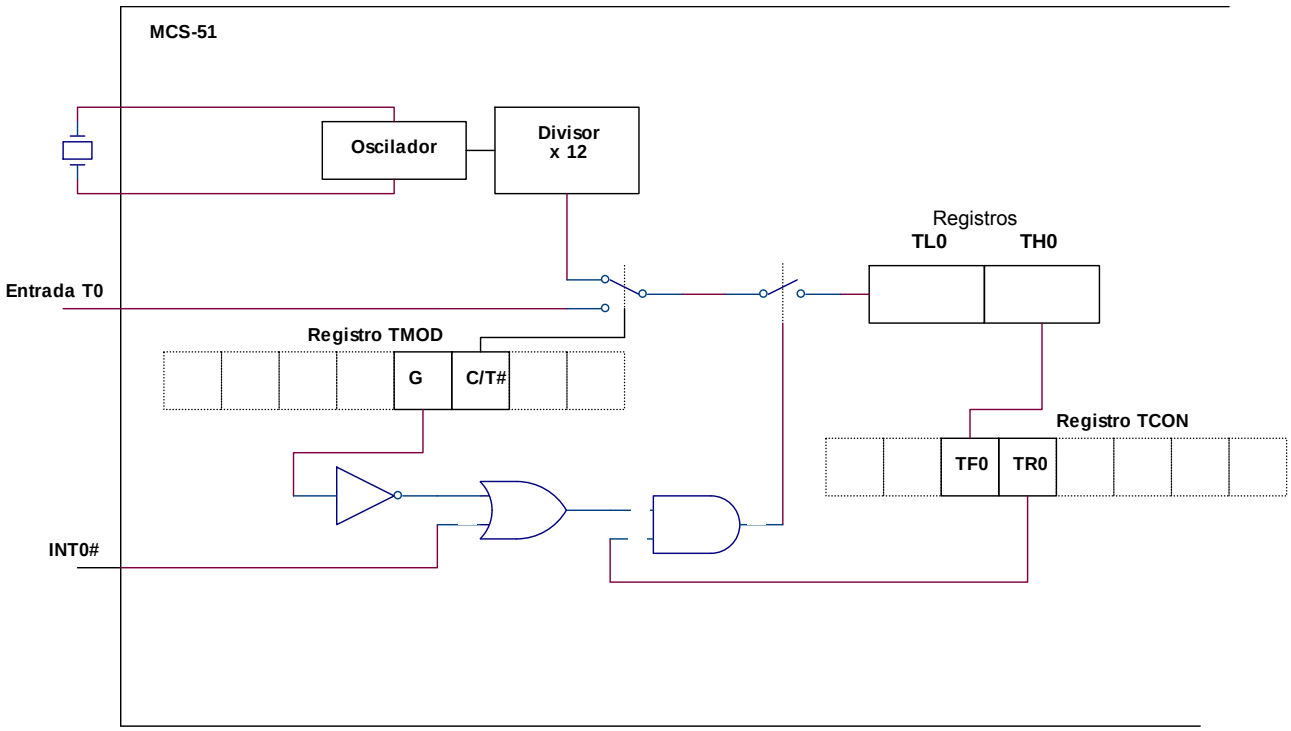


Fig. 23. Diagrama esquemático de los temporizadores comunes a todos los microcontroladores de la familia

[18] El tercer temporizador, exclusivo del 8052 y sus derivados tiene características particulares que estudiaremos en el punto 5.c

8.1 Contadores y Temporizadores.

Existen cuatro registros: TMOD, TCON y un registro de conteo de 16 bits (compuesto por los registros de 8 bits TL0 y TH0). El hardware asociado consta de:

- *unas compuertas*
- *un oscilador controlado por el cristal externo.*
- *un divisor de frecuencia por 12.*
- *2 multiplexores*

Todo ello controlado por el hardware y por el estado de los registros mencionados.

Como veremos próximamente, los registros TL0 y TH0 pueden programarse para comportarse de diferentes modos. En principio lo que debemos tener muy presente es que al recibir transiciones de señal por la línea de entrada a TL0 representada en la Fig. 23, se incrementa el valor de la cuenta de dichos registros (luego veremos las variantes). Al producirse un desborde de la cuenta de estos registros, se ve afectado el bit TF0 del registro TCON indicando el fin de la cuenta y que puede servir tanto para manejar el dispositivo por programa (lectura de registro de control TCON) o bien por interrupción.

En realidad, los registros TCON y TMOD poseen un conjunto idéntico de bits TF, TR, G y C/T# para cada uno de los temporizadores. Nuestro ejemplo es con el temporizador 0 simplemente porque se debía elegir uno de los dos ya que ambos temporizadores se comportan exactamente igual. Los registros de conteo del temporizador 1 se denominarán TL1 y TH1.

Analicemos el funcionamiento del temporizador/contador. En la Fig. 23 vemos que existen dos fuentes para producir el conteo del temporizador. Mediante el bit C/T# del registro TMOD (Fig. 24) seleccionamos que la cuenta se produzca de la división por doce de la frecuencia del cristal (temporiza con la referencia del cristal), o bien por pulsos que ingresan por el pin T0 (P3.4 cuenta eventos externos).

Sin embargo no es suficiente con que se den las condiciones nombradas. También deben darse las condiciones para que la compuerta AND active el multiplexor la llave que conecta con los registros de cuenta. Es indispensable que el bit TR (TIMER RUN) se encuentre en "1", luego basta con que o bien esté en "1" la entrada INT0# o esté en cero el bit G (GATE).

De todo esto se desprende que si por ejemplo debemos contar con una resolución del orden de los milisegundos, colocando un cristal de 12MHz y programando al bit C/T# = 0, tendremos una resolución de cuenta del orden del µseg.

Si en cambio la fuente de reloj (base de tiempo) debe ser externa, utilizamos la entrada T0[19]. Por supuesto que existen muchas variantes que estudiaremos con más detalle en los ejemplos de aplicación del capítulo correspondiente.

Veamos ahora los bits restantes de los registros TMOD y TCON.

Registro:TMOD No es Bit Addressable							
B7	B6	B5	B4	B3	B2	B1	B0
G1	C/T1	M1.1	M0.1	G0	C/T0	M1.0	M0.0

Fig. 24. Registo TMOD.

[19] Debe destacarse que en caso de no utilizarse la opción de contar eventos externos, las patas T0 y T1 de la puerta 3 quedan disponibles para su uso normal.

Como podemos ver, el registro se divide en dos, una mitad controla al temporizador 0 y la otra al temporizador 1. Resta ver el significado de los bits M (modo), las combinaciones de estos bits significan (para cualquiera de los temporizadores):

- Modo 0 (0,0): En este modo los registros TH y TL se juntan formando un registro de 13 bits[20], o sea que no se utilizan para la cuenta 3 bits del registro TL0 o 1 (los más significativos), su estado es indeterminado y no deben utilizarse en este modo. Una vez terminada la cuenta, deben recargarse los registros para poder comenzar una nueva cuenta.

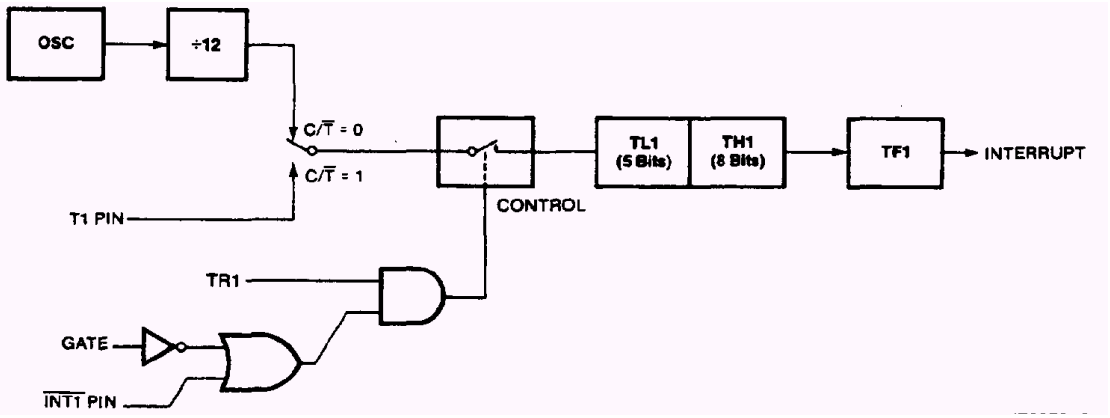


Fig. 25. Bosquejo del Temporizador 1 operando en modo 0.

- Modo 1 (0,1): Idem anterior pero utilizando los 16 bits del par TH-TL.

- Modo 2 (1,0): En este modo se utilizan para la cuenta los 8 bits del registro TL. El registro TH se utiliza para mantener en el un valor de *recarga* para TL, es decir, una vez agotada la cuenta de TL, éste se recarga con el contenido de TH y así sucesivamente. Es de notarse que al acabarse la cuenta de TL, se activa el bit TF correspondiente y se lo "resetea" leyéndolo. Cuando estudiemos la UART interna del microcontrolador veremos una de las utilidades de este modo.

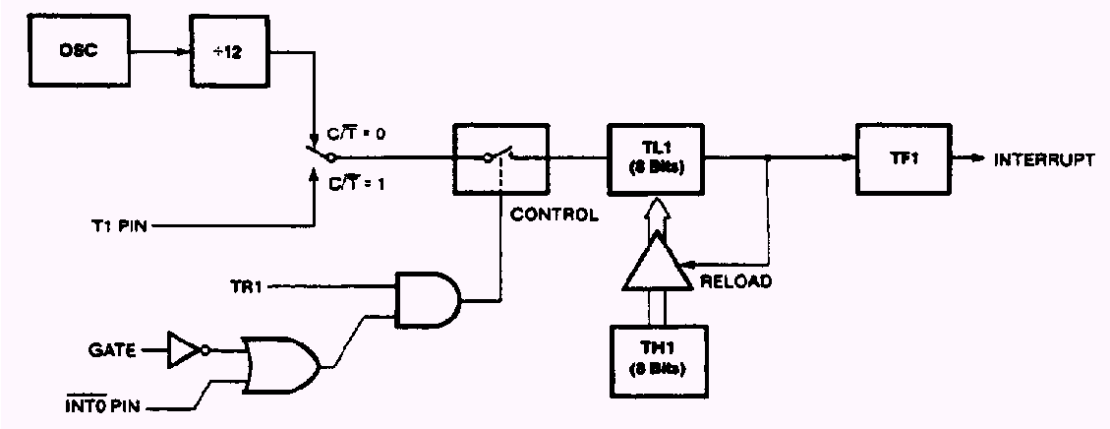


Fig. 26. Bosquejo del temporizador 1 operando en modo 2 (autorecarga)

[20] Este modo de 13 bits se planteó por compatibilidad con el 8048, predecesor en la familia de microcontroladores de Intel.

- Modo 3 (1,1): Este es un modo mixto, aquí tenemos que TL0 y TH0 se comportan en forma independiente (como si se trataran de dos temporizadores separados). TL0 funciona como en los modos 0 y 1 pero contando 8 bits y actúa sobre el indicador TF0. TH0 lo hace controlado por los bits de control del temporizador 1 y por ende actúa sobre TF1 y solo como temporizador. El temporizador 1 queda bloqueado como tal y solo se lo puede utilizar como contador pero sin interactuar con ninguno de los dos flags TF0 ó TF1.

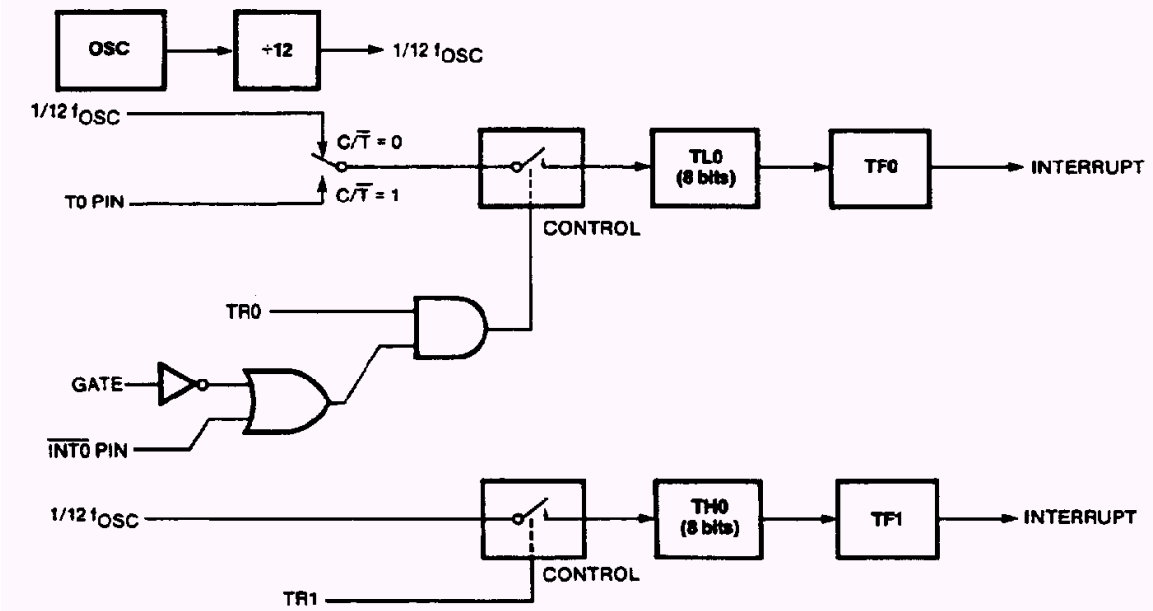


Fig. 27. Bosquejo del Temporizador 0 trabajando en modo 3.

Registro:TCON Bit Addressable							
B7	B6	B5	B4	B3	B2	B1	B0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. 28. Registro TCON

Simplemente mostramos la ubicación de los bits 4 a 7 que son los que por ahora nos interesan, los restantes los veremos al estudiar las interrupciones.

8.2 Capturando Eventos.

Como mencionamos anteriormente existe un tercer temporizador con capacidades especiales que lo diferencian de sus hermanos T0 y T1. Si bien este tercer temporizador no está disponible en todos los miembros de la familia MCS®-51 es importante estudiarlo.

El temporizador 2 posee las siguientes características:

- Contador/temporizador con y sin autorrecarga de 16 bits.
- Cuenta ascendente o descendente.
- Captura de cuenta. Suele ser frecuente querer determinar el momento en el que ocurre un evento. Un método sería estar leyendo permanentemente un bit de una puerta de entrada y en el momento que se activa la señal, leer un temporizador. Lamentablemente ese método emplea mucho tiempo de CPU. El temporizador 2 del 8052 incorpora un nuevo modo de trabajo que al producirse ese hecho externo se retiene el estado de ese

momento del temporizador 2 en un registro particular de captura. El temporizador sigue su cuenta pero el valor capturado quedó almacenado en el registro.

- Generador de frecuencias 50% de ciclo de actividad desde 61 Hz a 4 MHz.
- Generador de la velocidad de la comunicación (Baud Rate) de recepción o transmisión para UART interna.

Para llevar a cabo lo arriba enunciado, dispone de cinco registros y dos patas del puerto P1. Los registros involucrados comprenden:

- Un registro de control (T2CON).
- Dos registros para realizar la cuenta (TL2 y TH2).
- Dos registros para "capturar" instantáneamente la cuenta de los arriba mencionados o para almacenar el valor de recarga (RCAP2L y RCAP2H).

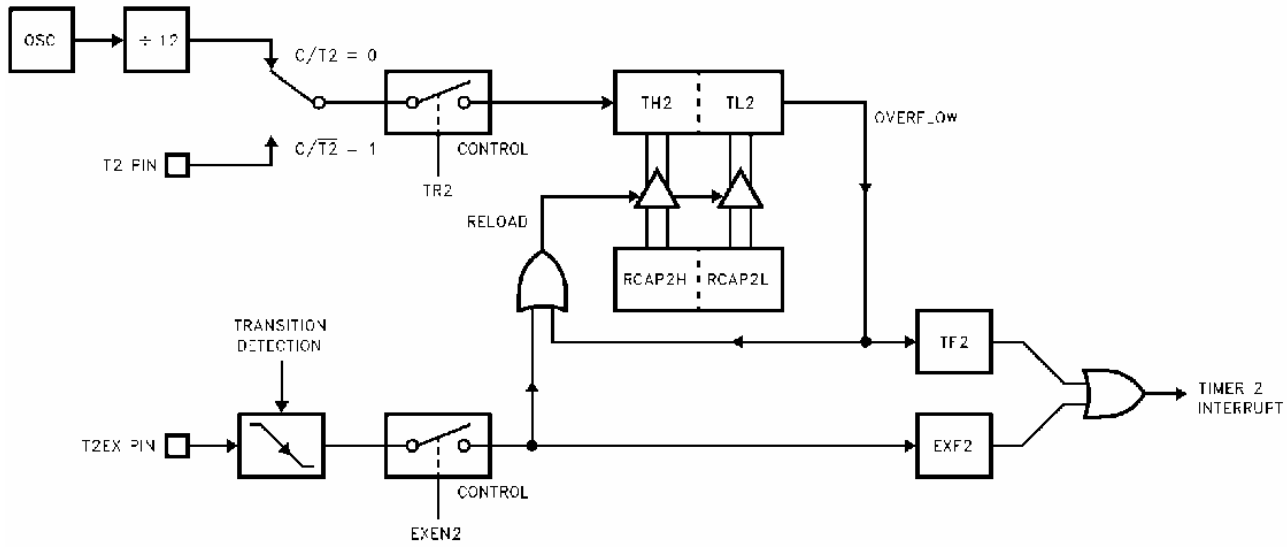


Fig. 29. Temporizador 2 en el modo captura.

Los pines externos que forman parte del P1 son:

T2 (P1.0) : Entrada externa de cuenta.

T2EX (P1.1) : Pulso de captura o recarga.

Veamos la estructura del registro T2CON que controla su funcionamiento:

Registro:T2CON - Bit Addressable							
B7	B6	B5	B4	B3	B2	B1	B0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

- TF2 (Timer 2 overflow flag): Puesta a "1" por desborde del temporizador 2, debe resetearse por software. No se pone a "1" cuando RCLK o TCLK son "1".
- EXF2 (Timer 2 external flag): Se pone en "1" cuando se produce una captura o recarga por una transición a estado bajo en T2EX y el bit EXEN2 está en "1". Cuando la

interrupción del temporizador 2 está habilitada EXF2 = 1 causará el saltó al vector de interrupción de temporizador 2. Debe limpiarse este flag por software y no causará interrupción en el modo de cuenta ascendente descendente (DCEN = 1).

- RCLK (Receive clock flag): Al estar en "1" fuerza la utilización de los pulsos de desborde del temporizador 2 como reloj de recepción del puerto serie en los modos 1 y 3 (se verá más adelante).

- TCLK (Transmit clock flag): Idem anterior pero con la transmisión del puerto serie.

- EXEN2 (Timer 2 external enable flag): Si está en 1, permite la captura o recarga del temporizador 2 en las transiciones a nivel bajo de T2EX si es que el temporizador 2 no está siendo utilizado como reloj del puerto serie. Puesto a "0" se ignoran los pulsos y transiciones de T2EX.

- TR2 (Start/stop control for Timer 2): En "1" arranca el temporizador 2.

- C/T2 (Timer or counter select): Igual a "0" funciona como contador interno divisor por 12 de la frecuencia de cristal. Igual a "1" es contador de eventos externos (por flancos descendentes).

- CP/RL2 (Capture/Reload flag): Cuando está en "1", se producen capturas del valor de cuenta sobre los registros de captura con los flancos descendentes en T2Exsi EXEN2 = 1. Cuando está en "0" se producen autorrecargas con el desborde del temporizador 2 o transiciones negativas en T2EX cuando EXEN2 = 1. Si RCLK o TCLK son "1" este bit es ignorado y el temporizador se autorrecarga solo con los desbordes.

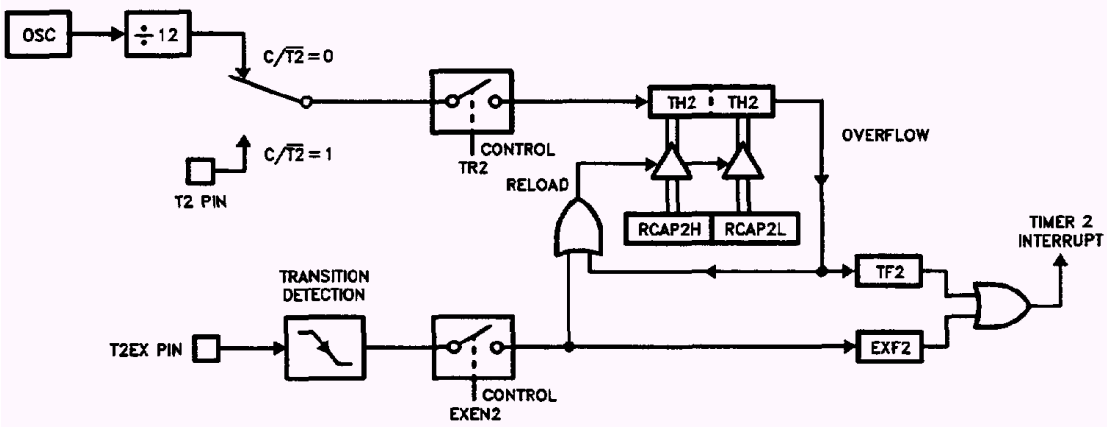


Fig. 30. Timer 2 en modo autorecarga con DCEN = 0.

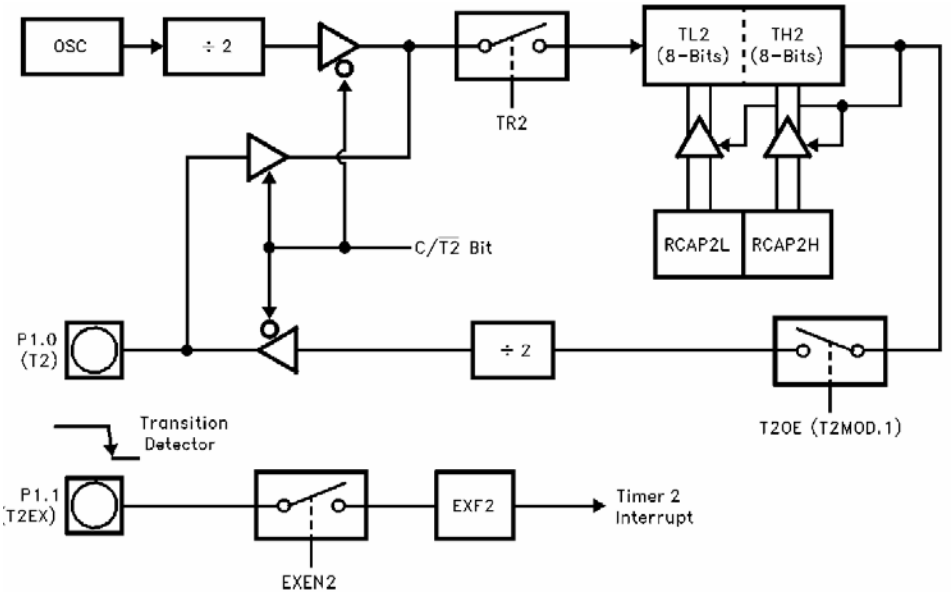
La programación del bit DCEN se hace sobre el registro T2MOD que describimos a continuación:

Registro:T2MOD No Bit Addressable							
B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	-	T2OE	DCEN

- : Reservados para usos futuros.

DCEN : Habilita la cuenta ascendente/descendente del temporizador 2.

Timer 2 en modo autorecarga con DCEN = 1.



Modo generador de frecuencia.

Existe un modo en el que se puede programar una señal rectangular de salida con un ciclo de actividad del 50% por el pin P1.0. Este pin , en lugar de ser una I/O común, además puede comportarse de forma de proporcionar una frecuencia de salida programable desde 61 Hz hasta 4 MHz (con cristales externos de 16 MHz). Para configurarlo en este modo se deben cumplir las siguientes condiciones en estos bits: C/T2=0, T2OE=1, TR2=1.

La frecuencia queda determinada por la siguiente ecuación:

Fout = Frecuencia de cristal osc. / 4.(65535 – RCAP2H,RCAP2L)

En este modo no se generan interrupciones por desbordes del temporizador 2. Es posible también utilizar al temporizador 2 como generador de reloj para el puerto serie y frecuencia externa, pero no pueden tener frecuencias independientes ya que ambos dependen del contenido de RCAP2 y el cristal externo.