Notas Técnicas de Uso y Aplicación

NT1000

SISTEMA MULTITAREA BASADO EN PRIORIDADES - KERNEL
PARA MICROCONTROLADORES CON CPU08

Preparado por: Rangel Alvarado
Estudiante Graduando de Lic. en Ing. Electromecánica
Universidad Tecnológica de Panamá
Panamá, Panamá

ÍNDICE

Introducción

¿Alguna vez ha pensado lo difícil que es programar sistemas para microcontroladores?. El porqué de esto se debe a que no hay un orden específico para programar un sistema de este tipo.

Prueba de esto está cuando queremos hacer una mejora de software a un sistema, bajo la petición de un cliente o bajo nuestro propio gusto; el final de la historia es un gran dolor de cabeza. Aunque parezca mentira, se puede realizar un sistema el cual sea fácil de modificar, de código reutilizable, liviano, de respuesta rápida y poderoso cuyo nombre de guerra es Kernel, palabra que se asocia mucho con sistema multitarea y que además existe toda una filosofía detrás de este abstracto concepto.

A grandes rasgos, el Kernel, es el corazón o núcleo de un sistema embebido (en este caso, microcontroladores), el cual se encarga de administrar acceso a dispositivos, variables y del tiempo de ejecución de la máquina, comunicación entre tareas, por decir ciertas cualidades. En este se arrrojan módulos de programa y da la flexibilidad de que ningún dispositivo o recurso tenga conflicto con otro.

¿Qué es la multitarea en sí?, bueno, es la apariencia de que varias tareas o módulos o secciones de código de programa (que es todo lo mismo) trabajen al mismo tiempo y que piensan que tienen el CPU para sí mismas; esta apariencia es controlada por el kernel, pues, solo una tarea puede correr a la vez.

En esta guía se describen los conceptos básicos y se da una aplicación de kernel de tiempo real para evaluar su aplicación.

Conceptos Básicos

Tareas y Multitareas (Tasks and Multitasking)

Una tarea no es más que un código de programa (programa para: prender un LED, escanear teclado, escribir en una pantalla). También se le llama a la tarea, *flujo de ejecución de un programa* (THREAD). Es aquella que piensa que tiene el CPU para ella misma. Las siguientes líneas de código, puede ser el ejemplo de una tarea.

```
TASK01                          ;Led on
            bset BIT7,PORTD     ;Prende el LED
            rts                 ;Retorna
```

Listado No. 1: Ejemplo de una tarea sencilla. Prende un LED.

Los diferentes estados de una tarea en un kernel son:

- Durmiente (DORMANT)
- Lista (READY)
- Corriendo (RUNNING)
- En espera de un evento (WAITING FOR AN EVENT) o,
- Interrumpida (INTERRUPTED)

Durmiente: son tareas no creadas o listas para ser creadas. Es una tarea no activada y que se encuentra en ROM, puede o no tener código

```
TASK06                          ;Ejemplo de una tarea durmiente
            rts                 ;retorna
```

Listado No. 2: Ejemplo de una tarea durmiente. En teoría una tarea durmiente reside en nuestra ROM, pero que no se ha hecho disponible por el kernel; la tarea 6 pudiera ser un ejemplo de esta.

Lista: Se le dice lista a la tarea que le sigue en orden de ejecución.

Corriendo: Es aquella tarea que tiene el control actual del CPU.

En espera de un evento: Cuando esta requiere que ocurra un evento. Un ejemplo de esto es cuando una operación de E/S se complete, un recurso sea disponible, ocurra un pulso, etc.

Interrumpida: Cuando ocurre una interrupción el sistema interrumpe la tarea actual.

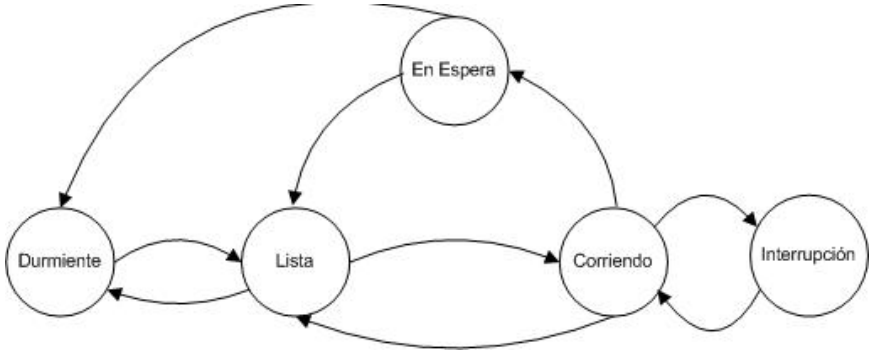

Figura No. 1: Estados de las tareas. Ciclo que realiza en el Kernel

Un sistema multitareas es entonces el proceso de darle un itinerario a las tareas y conmutar el CPU. Una de los aspectos más importantes es proveer que las aplicaciones del programador se realicen de forma inherente a la complejidad del programa.

¿Como se ejecutan las tareas?
Algo importante es saber como se ejecutan las tareas en esta clase de sistemas. Las tareas se ejecutan de modo paralelo, una detrás de otra.

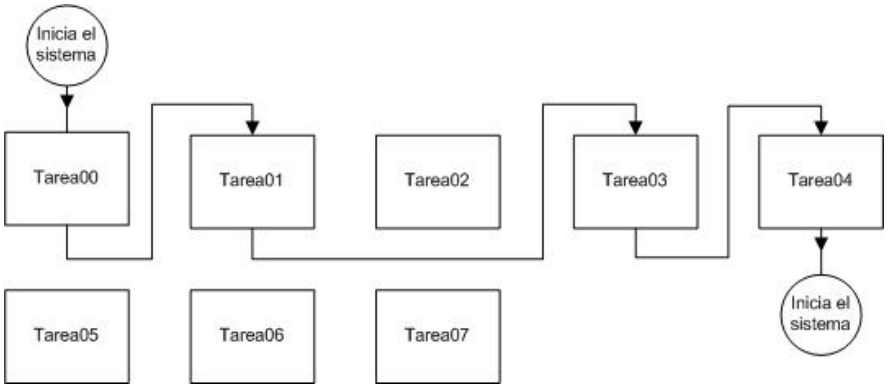

*Figura No. 2: Diagrama de Bloques de las tareas. Las tareas se ejecutan **una detrás de otras sin esperar**, si una tiene que esperar algo carga un temporizador y sigue*. En este ejemplo tenemos las tareas 2, 5, 6 y 7 deshabilitadas o sea que no se ejecutan.

Sección Crítica (Critical Section)
También llamada *región crítica*, es una sección de código, datos compartidos, o recurso compartido, o una función no-reentrante[1] la cual en su ejecución no puede ser interrumpida. Un método de que esto no ocurra es que se deshabiliten las interrupciones, se revise el código y luego habilitar las interrupciones, esto se debe de hacer en un período muy corto de ejecución y no se debe abusar de esta simple

[1] Función que no puede ser utilizada por más de una tarea en su momento, pues puede que la data de salida sea corrupta.

técnica, pues el uso excesivo, puede hacer que el sistema no funcione como uno de tiempo real.

Recursos y Recurso Compartido (Shared Resource)
El recurso es cualquier dispositivo de E/S (entrada/salida) como una pantalla o un teclado, impresora, disponible para alguna tarea del kernel. Entonces, dicho lo anterior, el recurso compartido es el mismo que puede ser utilizado ya por varias tareas. Para que la data no llegue corrupta, cada tarea debe de tener un acceso exclusivo a este para prevenir corrupción de la información. El método de tratar que los recursos compartidos no creen conflicto en un sistema se llama *exclusión mutua* (MUTEX) y se tratará en temas posteriores.

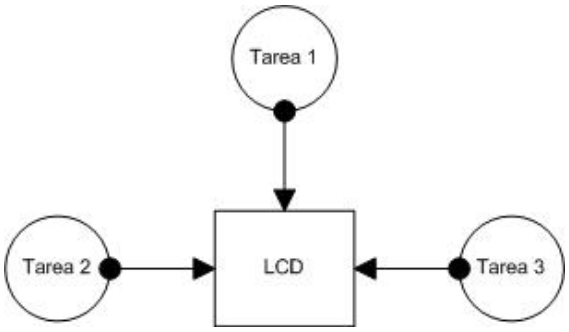

Figura No. 3: Recurso Compartido. Como se muestra en la figura, todas las tareas quieren acceder a un recurso, en este caso, una pantalla de cristal líquido. No todas pueden accederlos a la vez, solo a una tarea se le permite el ingreso de momento, hasta que esta termine de utilizarlo.

Conmutación de Contexto o Cámbio de Tarea (Context Switch/Task Switch)
Cuando un sistema multitarea trata de decidir que tarea sigue en un momento diferente, salva el contexto (registros del CPU) en la pila[2]. Una vez que esto sucede, registros anteriores son recuperados de la pila y se prosigue con la ejecución del código. (A, H:X, CCR, PC, SP).

Subrutinas de Interrupciones (ISR)
Una interrupción es un mecanismo de hardware usado para informar al CPUI que un evento asíncrono ha ocurrido. Cuando ocurre, el CPU salva todo o parte de su contexto y salta a un servicio de interrupción (ISR). A su retorno toma las decisiones adecuadas para seguir con el programa. Existe la posibilidad de ignorar las interrupciones, por medio de la instrucción SEI y habilitarlas nuevamente por medio de la instrucción CLI. Para cualquier sistema de tiempo real, las interrupciones deben ser deshabilitadas en intervalos relativamente cortos para no afectar el rendimiento del sistema.

[2] En las interrupciones, guarda todos los registros, excepto el registro H.

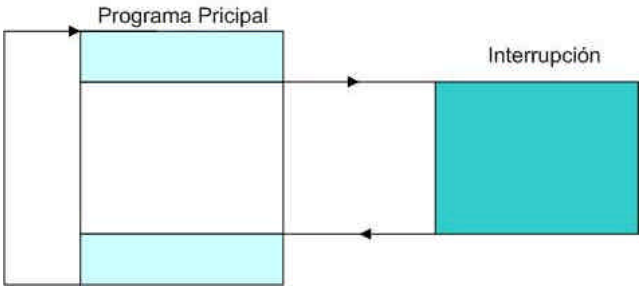

Figura No. 4: Interrupciones. Las interrupciones, por lo común son ocasionadas por el hardware y se dan en cualquier momento, los sistemas que hemos diseñado hasta este momento, se basan en esta técnica.

```
TICKISR
                    ;
                    ;código de interrupción
                    ;
        rti

        org TIM     ;Puntero del Vector del Temporizador
        dw TICKISR  ;Busca la dirección a la que apunta TIM
```

Listado No. 3: Interrupción. En este ejemplo se utiliza la interrupción del TIMER Canal 1, que salta a su subrutina al darse un sobreflujo del contador.

Kernel

El kernel es la parte más importante del sistema multitareas, responsable de la administración de tareas, el tiempo del CPU y comunicación entre tareas. El servicio que proporciona el kernel de mayor importancia es el cambio entre contextos. Cada tarea del Kernel necesitará entonces su propio espacio de stack, esto es RAM, un recurso escaso (para el caso de el microcontrolador JK3 y JL3 es de 128 bytes de RAM).

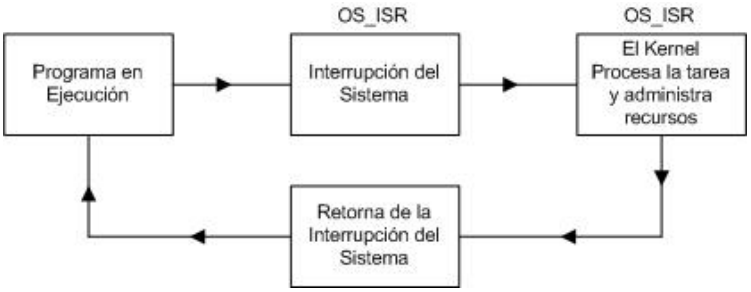

Figura No. 5: Ejecución de un Kernel. El Kernel es el TICKISR en este caso, el cual divide el tiempo de tareas de una forma astuta.

```
TICKISR
                ldx BIT_COUNTER     ;Bit en el que se está parado
S04.NEXTBIT     cmpx #MAX_BITS      ;compara BITCOUNTER con 7
                ble SEC04.COMPARE   ;Si es <= salta a comparar
                clr BIT_COUNTER     ;sino, hay un sobreflujo y borra
                ldx BIT_COUNTER     ;X = BITCOUNTER = 0
                bra SEC04.NEXTBIT   ;Salta a buscar tarea inicial
S04.COMPARE     lda BIT.,x          ;A = Bit de la tabla en la posición X
                and OS_TASK         ;A = A && OS_TASK
                cmp #NO_BIT_TASK    ;¿Tareas?
                beq SEC04.INCBIT    ;No hay, entonces revisa el siguiente
                                    ;bit
                stx OS_BIT          ;X->OS_BIT
                inc BIT_COUNTER     ;incremento BIT_COUNTER
                                    ;EVALUO A QUE TAREA SALTAR
                bclr TOF,TSC        ;Borra la interrupción
                rti                 ;retorno
S04.INCBIT      incx                ;incrementa X
                inc BIT_COUNTER     ;incremento BIT_COUNTER
                bra SEC04.NEXTBIT   ;reviso el siguiente bit
```

Listado No. 4: Kernel. En este ejemplo se utiliza la interrupción del TIMER, el cual se encarga del control de determinar que tarea está lista.

Itinerario (Scheduler)

Tambien llamado repartidor de tareas (dispatcher), es la parte del kernel la cual determina sabiamente, que tarea sigue a continuación. La mayoría de los sistemas de tiempo real se basan en prioridades. En un kernel basado en prioridades, el control del CPU siempre sera expuesto a *la tarea de más alta prioridad* lista a correr. Hay dos tipos de kernel basado en prioridades: Cooperativo (non-preemtive) y Preemptivo (preemptive). La palabra preempt, viene de "*tomar el lugar de*".

```
                jsr ADJUST_POINTER  ;Ajusta el Puntero
                jsr TASK_TO_RUN     ;Decide que tarea es prioritaria
```

Listado No. 5: Itinerario. Parte del Kernel que se encarga de suplantar la tarea en curso por una de mayor prioridad.

Instante del reloj (Clock Tick)

Un instante de reloj, es una interrupción especial que ocurre periódicamente. El tiempo entre interrupciones es de aplicación específica y para sistemas embebidos está entre los 10 y 200 ms. La interrupción permite al kernel retardar tareas por una cantidad de ticks y provee un tiempo fuera de la aplicación si no responde. Mientras el sistema sea frecuentemente interrumpido, el sobremanejo es mayor.

El quantum o pedazo de tiempo (time slicing), es la cantidad de tiempo en la que el kernel selecciona otra tarea. Para esta aplicación fue seleccionado como base 20 ms, por medio de la ecuación:

$$TMOD[H:L] = \frac{t \cdot 4.9152 \times 10^6}{2^{(2+PS)}}$$

Ecuación No. 1: Módulo de Contador. Determina el quantum o time slice de la aplicación.

En donde "*t*" es el tiempo en segundos y "*PS*", es el valor del Preescalar de binario a decimal, que es el divisor del temporizador. El resultado de esta operación da un módulo en decimal, que debe ser transformado en hexadecimal.

Para t = 20 ms, PS = 0. TCH1 = 24576_{10} = 6000_{16}.

```
OS_INIT_TICK                ;Subrutina de tick de reloj del sistema
        ldhx #TICK          ;H:X = TICK = $6000
        sthx TMODH          ;H:X -> TMOD[H:L]
        mov #TIM_CONFIG,TSC ;TOIE = TSTOP = 1
        ldhx #NULL          ;H:X = 0, nuevamente inicializa a 0
        cli                 ;Habilita interrupciones
        bclr TSTOP,TSC      ;Habilita el temporizador
        rts                 ;retorna
```

Listado No. 6: Instante del Reloj . Esta parte importante del programa, trata de suministrar al kernel una interrupción periódica para luego en la interrupción administrar que tarea sigue.

Los Sistemas de Tiempo Real

Hay dos tipos de sistemas de tiempo real, los de SOFTWARE y los de HARDWARE. El SOFTWARE, trata de ejecutar el sistema lo más rápido posible, mientras que el HARDWARE, se ejecutan tanto correctamente, como en tiempo. Un equilibrio del híbrido de estas dos tendencias, lo poseen la mayoría de los sistemas operativos de tiempo real.

En concepción general, toda aplicación que se realice, para una aplicación específica y responda al usuario correctamente, es un sistema de tiempo real. Hay tres tipos de sistema de tiempo real distintos:

- Sistemas de Primer Plano/Segundo Plano (Foreground/Background)
- Kernel Cooperativo (Non-Preemptive Kernel)
- Kernel Preemptivo (Preemptive Kernel)

Sistemas de Primer Plano/Segundo Plano (Foreground/Background)
El primero se refiere a sistemas de baja complejidad de código, también se les conoce como super lazos (super – loops). Hasta ahora, lo que hemos estado haciendo, es diseñar sistemas de este tipo (ver figura 1). Este diseño, es, hasta cierto punto, óptimo en ocasiones que se quiere medir los recursos del sistema, pero tiene como desventaja la disyuntiva que todo el código actúa con el resto. Una pequeña modificación de software, puede ser una gran tarea a realizar para la entrega de un proyecto.

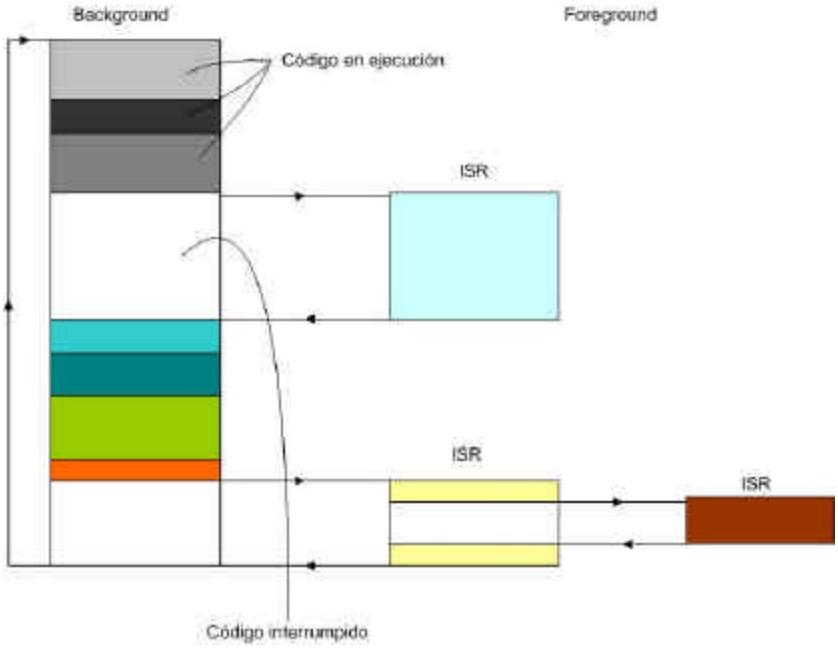

Figura No. 6: Sistema de Primer Plano/Segundo Plano. Al sistema Foreground, o nivel de interrupción se le llama ***nivel de tareas****, pues, las operaciones críticas son efectuadas en esta zona de código.*

La figura 6 consiste en un grupo de aplicaciones basadas en un lazo cerrado (funciones) que realizan una deseada operación (segundo plano). Los eventos asíncronos o interrupciones (primer plano), realizan una determinada toma de desición, pero el nivel de respuestas de tareas, no es ejecutado hasta retornada esta interrupción.

Este tipo de diseño primitivo, no es malo, es más, muchos sistemas embebidos, son diseñados bajo esta democrática filosofía, puede que en algunos casos, este tipo de sistema sea la mejor desición.

Kernel Cooperativo (Non-Preemptive)

Este tipo de kernel requiere que cada tarea realice algo explícito antes de darle el control al CPU. Para mantener la idea de que las acciones ocurren concurrentemente, este proceso debe repetirse frecuentemente. Cada tarea coopera con otra para compartir el CPU. La interrupción maneja que tarea sigue a continuación, pero al retornar de la interrupción esta siempre retorna a la tarea interrumpida. Luego, la nueva tarea ocupa el lugar de la anterior solo cuando la tarea en curso deja el control del CPU.

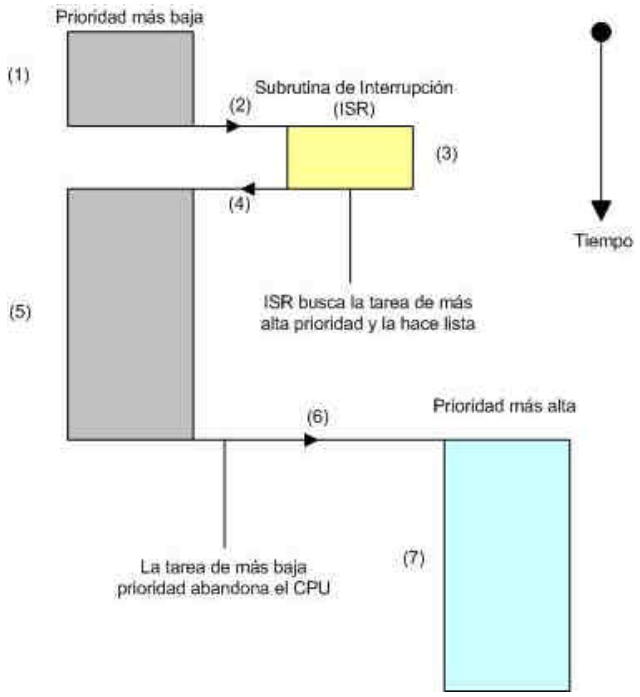

Figura No. 7: Sistema Multitarea Cooperativo. La tarea menos prioritaria le cede el control a la nueva tarea solo al culminar su ejecución.

Una analogía de un kernel cooperativo es una conversación entre dos personas,, cuando una habla (gris), la otra escucha (celeste). La falla de este sistema radica en que si llegase una persona que nunca parara de hablar y nunca cede el turno, pues colgaría el sistema. Un ejemplo de sistema cooperativo es el *MS Windows 3.11*.

La figura 7 funciona como sigue:

(1) La tarea menos prioritaria, o sea, la de momento, está en ejecucióna, hasta que és interrumpida por el sistema operativo.
(2) El sistema maneja el evento ISR
(3) Determina la tarea más prioritaria y la hace lista (READY TO RUN)
(4) La rutina de interrupción finaliza y el CPU retorna a la tarea interrumpida
(5) La tarea que fue interrumpida sigue ejecutándose hasta que finaliza, y
(6) Llama un servicio provisto por el kernel que permitirá abandonar el CPU hacia otra tarea.
(7) La nueva tarea más prioritaria toma control del CPU y se sigue ejecutandose hasta que es interrumpida por el sistema operativo.

Una de las ventajas de este tipo de kernel es que sus *interrupciones latentes*[3] son cortas, que no se tiene la necesidad de salvar los datos compartidos, y que se pueden usar sin problemas funciones sin necesidad de que la data se pierda, debido a que el CPU solo abandona la tarea cuando esta se completa.

[3] Tiempo que las interrupciones permaneces inhabilitadas y se ejecuta la primera instrucción de la ISR.

Una desventaja clara es que el nivel de respuesta del kernel a la siguiente tarea es bajo, debido a que la tarea lista tiene que esperar a que se le ceda el control.

```
ADJUST_POINTER                  ;Ajusta el puntero de la tarea
            lda OS_BIT          ;A = valor de prioridad
            ldx #TWO            ;Carga con 2
            mul                 ;Multiplica 2*A
            sta OS_TASK_PTR     ;Lo almacena en el puntero de direcciones
            rts                 ;retorna
```

Listado No. 7: Determinación del puntero a la tarea . Parte de la interrupción (TICKISR) del kernel que ajusta el índice a valores de dos en dos.

```
TASK_TO_RUN
            ldx OS_TASK_PTR     ;Carga variable OS_TASK en X
            lda TASK_LIST,x     ;Carga parte alta de la palabra (WORD = 2
                                ;bytes) de dirección de la tarea prioritaria,
                                ;0084
            sta OS_PTASK        ;Lo almacena en la parte alta de la memoria
                                ;ram
            lda TASK_LIST+1,x   ;Carga parte baja de la palabra (WORD = 2
                                ;bytes) de dirección de la tarea prioritaria
            sta OS_PTASK+1      ;Lo almacena en la parte baja de la memoria
                                ;ram y lo almacena en 0085
            rts                 ;retorna
```

Listado No. 8: Determinación de la tarea más prioritaria . Parte del kernel que busca la tarea más prioritaria y almacena en las direcciones 0084 y 0085. OS_TASK_PTR es el índice que busca en la lista.

```
S01.LOOP                        ;AQUÍ SE EJECUTARIA EL SCHEDULER
            jsr OS_TASK_RUN     ;Ejecuta la tarea
            bra S01.LOOP        ;Salta a hacer el lazo infinito
```

Listado No. 9: Tarea a correr. Es un lazo principal que solo ejecuta la tarea que se encuentra en RAM, salta a la subrutina de la dirección 0083.

Kernel de Toma de Lugar o Preemptivo (Preemptive)

Un kernel de este tipo es requerido cuando la respuesta del sistema es importante. La mayoría de los kernels comerciales son preemptivos. La característica principal de este sistemas es que la tarea más prioritaria toma el control del sistema inmediatamente. Cuando la tarea es lista para correr, la tarea actual es suspendida y la de más alta prioridad le cede el control inmediatamente al CPU.

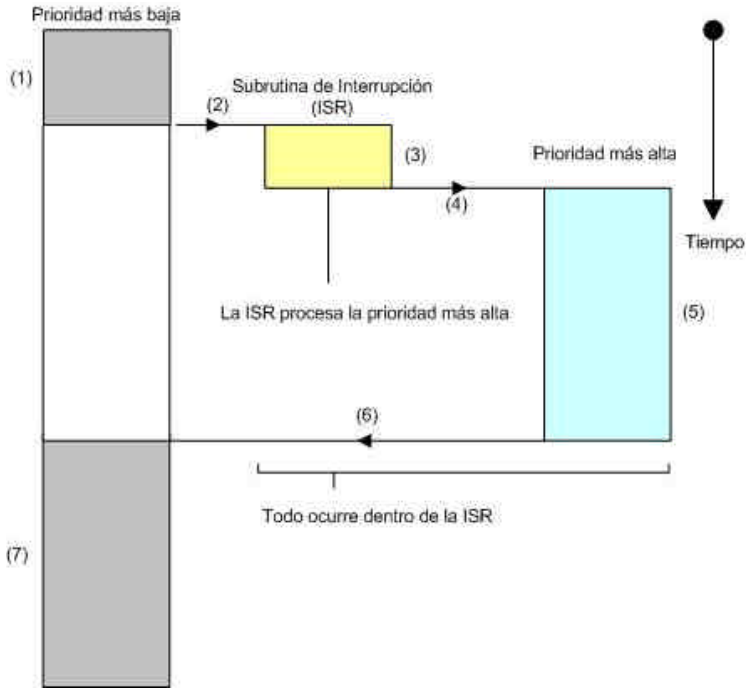

Figura No. 8: Sistema Multitarea Preemptivo. La tarea menos prioritaria le cede el control a la nueva tarea inmediatamente esta se hace disponible por el kernel.

Una analogía de kernel preemptivo está en la conversación de dos personas y un moderador. Cuando uno de los dos personajes habla más de la cuenta, el moderador le dice al personaje hablantín que termine inmediatamente y le cede el turno a la otra persona.

La figura 8 funciona como sigue:

(1) La tarea menos prioritaria, o sea, la de momento, está en ejecucióna, hasta que és interrumpida por el sistema operativo.
(2) El sistema maneja el evento ISR
(3) Determina la tarea más prioritaria y la hace lista (READY TO RUN)
(4) El kernel proporciona un servicio, el cual le cede el control del CPU a la tarea más prioritaria
(5) La tarea de mayor prioridad se ejecuta y finaliza
(6) El servicio de interrupción culmina y se retorna a la tarea de menor prioridad.
(7) La tarea menos prioritaria vuelve a tomar el control del sistema.

En el kernel preemtivo no se deben de usar funciones de tipo no-reentrantes, pues hay posibilidad de que la data se pierda. La forma más inteligente de acceder a recursos o secciones de código compartidas es por medio del uso de exclusiones mutuas como semáforos.

Algunos ejemplos de sistemas operativos de este tipo son UNIX, LINUX, uC/OS – II, etc.

Exclusión Mutua

La manera más sencilla de comunicar recursos compartidos, estructuras de datos, tareas y demás recursos es por medio de exclusiones mutuas o MUTEX. Los diferentes métodos de exlusión mutua son:

- Deshabilitación de interrupciones
- Probar e Imponer (TAS)
- Deshabilitar el Itinerario (DISABLE SCHEDULER)
- Semáforos

Deshabilitación de interrupciones

La manera más sencilla y rápida para ganar acceso exclusivo a un recurso compartido es de deshabilitar las interrupciones, y luego habilitarlas. Esta característica es inherente del ensamblador de HC08 por medio de las instrucciones SEI y CLI.

```
        sei     ;Deshabilita interrupciones
                ;Accesar recurso; leer/escribir variables
        cli     ;Habilita interrupciones
```

Listado No. 10: Exclusión Mutua, deshabilitación y habilitación de interrupciones.

Debe de ser lo más cuidadoso posible de no deshabilitar el recurso por muy largo tiempo, porque este puede afectar la respuesta del sistema, especialmente si es un sistema preemptivo.

Probar e Imponer (TEST AND SET o TAS)

Es un método muy bueno si no se usa un kernel. Las funciones están de acuerdo en accesar un recurso si la variable es 0. Para prevenir que otro recurso que quiera accesar la variable no la accese, la variable se pone en 1.

```
                lda TAS             ;variable de Probar e Imponer
                sei                 ;Deshabilita interrupciones
                djnza NoDisponible  ;Si la variable TAS es 1 salta a NoDisponible
                inc TAS             ;la variable se vuelve a 1
                cli                 ;Habilita interrupciones
                                    ;Accede al recurso
                sei                 ;Deshabilita interrupciones
                clr TAS             ;la variable se vuelve a 0
NoDisponible    cli                 ;Habilita interrupciones
                                    ;No hay acceso al recurso
```

Listado No. 11: Exclusión Mutua, Probar e Imponer. Ejemplo de cómo se puede implementar, hasta el momento esta no se ha implementado en el programa.

Deshabilitar el Itinerario (SCHEDULER)

Si la tarea no está compartiendo variables o estructuras de datos, se puede habilitar o deshabilitar el itinerario, que permite que una o más tareas compartan datos sin la posibilidad de que la data procesada sea corrupta. Cabe aclarar que mientras el itinerario es bloqueado las interrupciones se pueden acceder, lo que no puede hacer, es ejecutar la tarea más prioritaria.

TICKISR

```
                lda #SCHEDULE_LOCK      ;Carga variable de bloqueo del
                                        ;kernel
                bit OS_FSCR             ;A = SCHEDULE_LOCK & A
                bne S04.LOCKED          ;A = 0?, si no es igual salta a salir del
                                        ;KERNEL
                                        ;……..
                                        ;……..
                                        ;……..
S04.LOCKED      pulh                    ;retorna H de la Pila
                bclr TOF,TSC            ;Borra la interrupción
                rti                     ;retorno

TASK05
                inc OS_FSCR             ;Inhabilita el kernel
                                        ;código de la aplicación
                dec OS_FSCR             ;devuelve el control al kernel
                rts
```

Listado No. 12: Exclusión Mutua, Bloquear el Kernel. La variable OS_FSCR, Bit 0, se encarga del control de que el kernel ejecute la tarea más prioritaria. Durante el tiempo que esté OS_FSCR, Bit 0 en 1, no podrá ejecutarse la tarea siguiente.

Semáforos

El semáforo fue inventado por un alemán, llamado Dijsktra a mediados de los 60s. Un semáforo es entonces un mecanismo de control para el kernel de accesar todo tipo de recursos. Exiten dos tipos de semáforos, los *binarios* y los de *cuenta*. Los binarios se utilizan cuando se quiere acceder a una variable de momento (1 = accesible, 0 = no accesible), mientras los de cuenta, se utilizan por lo general para el buffer anillo[4] (255 = buffer lleno, menor de eso hay espacio para meter datos; si el dispositivo es de 8-bits)

Hay tres estados en los que se puede encontrar un semáforo. Inicializado (INITIALIZE), pues deben de proporcionarse los estados iniciales del semáforo. En espera (WAITING o PENDING); si el semaforo es mayor a 0, se accede al recurso y luego se decrementa, en señal de que está en uso. SIGNAL, es la parte contraria de waiting, que dispone del uso del semáforo y lo bloquea hasta que se disponga del uso total de esta.

[4] Tabla de data que contiene información, como por ejemplo la data de una impresora de computadora que se encuentra en cola.

Diagrama Esquemático

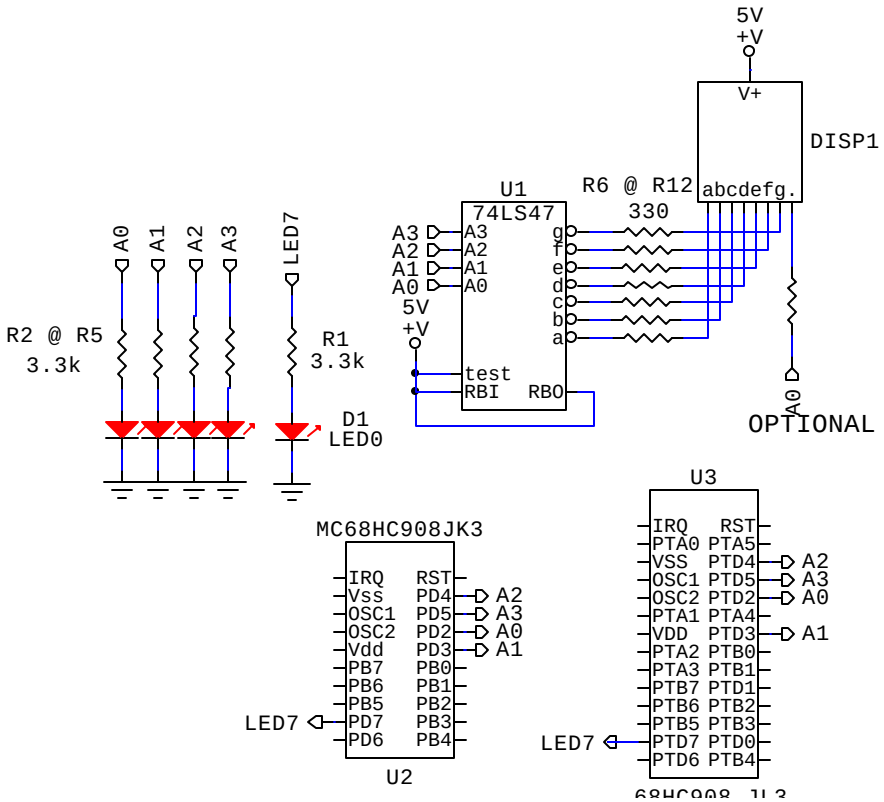

Figura No. 10: Diagrama Esquemático. El diagrama esquemático solo se le aplica a un microcontrolador, no a ambos a la vez. Este esquema prueba que el sistema multitarea realiza y ejecuta los códigos independientes paralelamente.

A continuación, se muestra el programa que controla el esquemático anterior. El sistema prende por intervalos de 1 segundo el LED PTD7 y por intervalos de 3 segundos el 7 segmentos contando hasta 15 (BCD). El kernel es del tipo cooperativo controlado por la interrupción por Temporizador.

Diagrama de Flujo

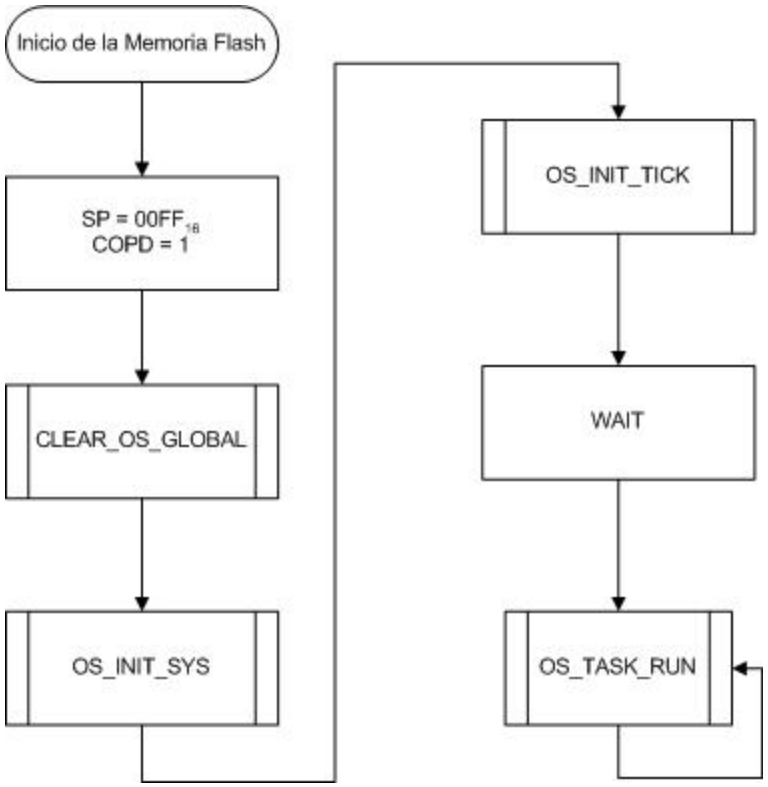

Figura No. 11: Diagrama de Flujo, Sección 1. La sección consiste en un lazo cerrado, donde cada 20 ms, OS_TASK_RUN ejecuta una tarea diferente.

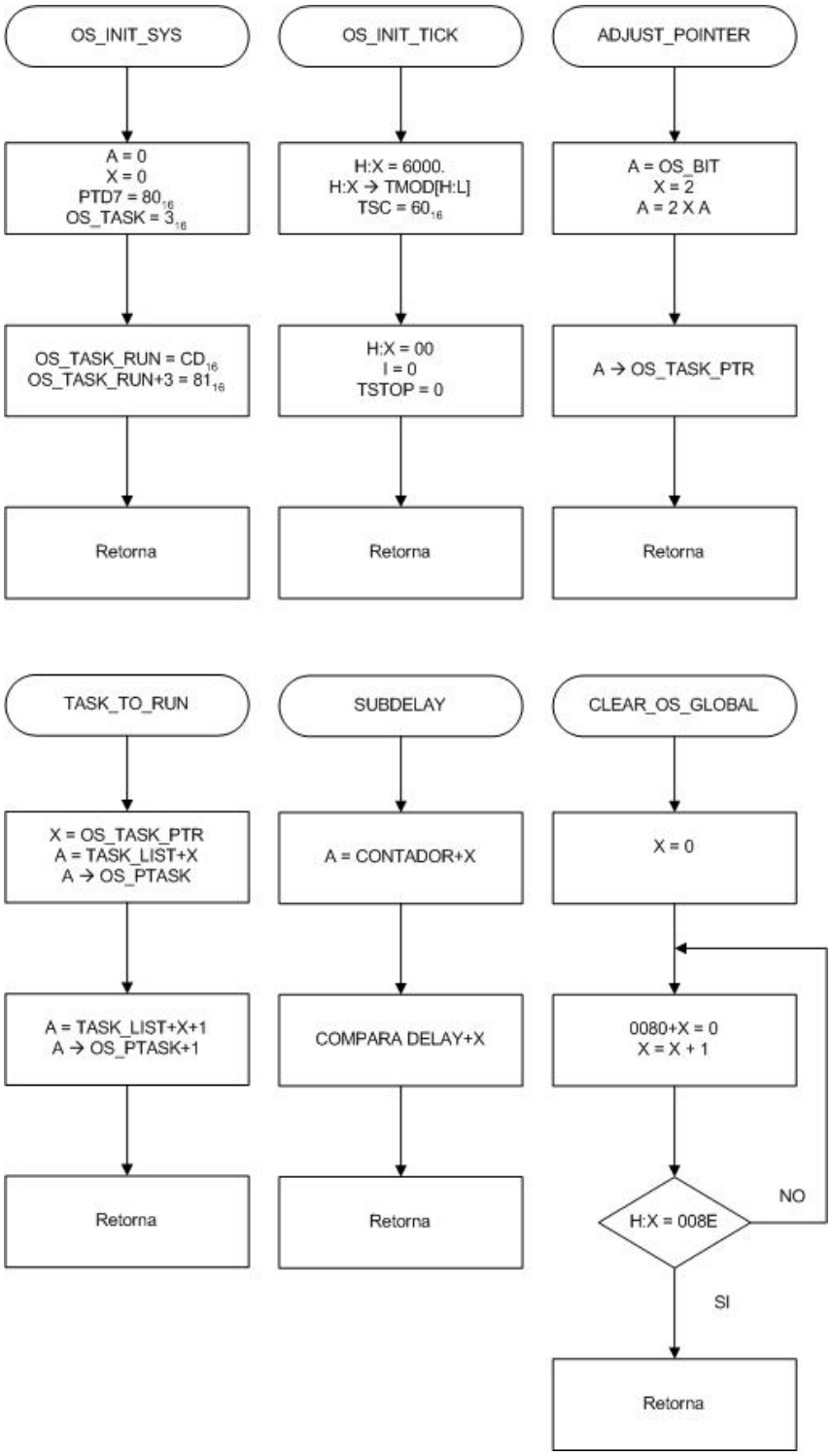

Figura No. 12: Diagrama de Flujo, Sección 2. Subrutinas de la sección 2 utilizadas en las diferentes secciones del kernel cooperativo.

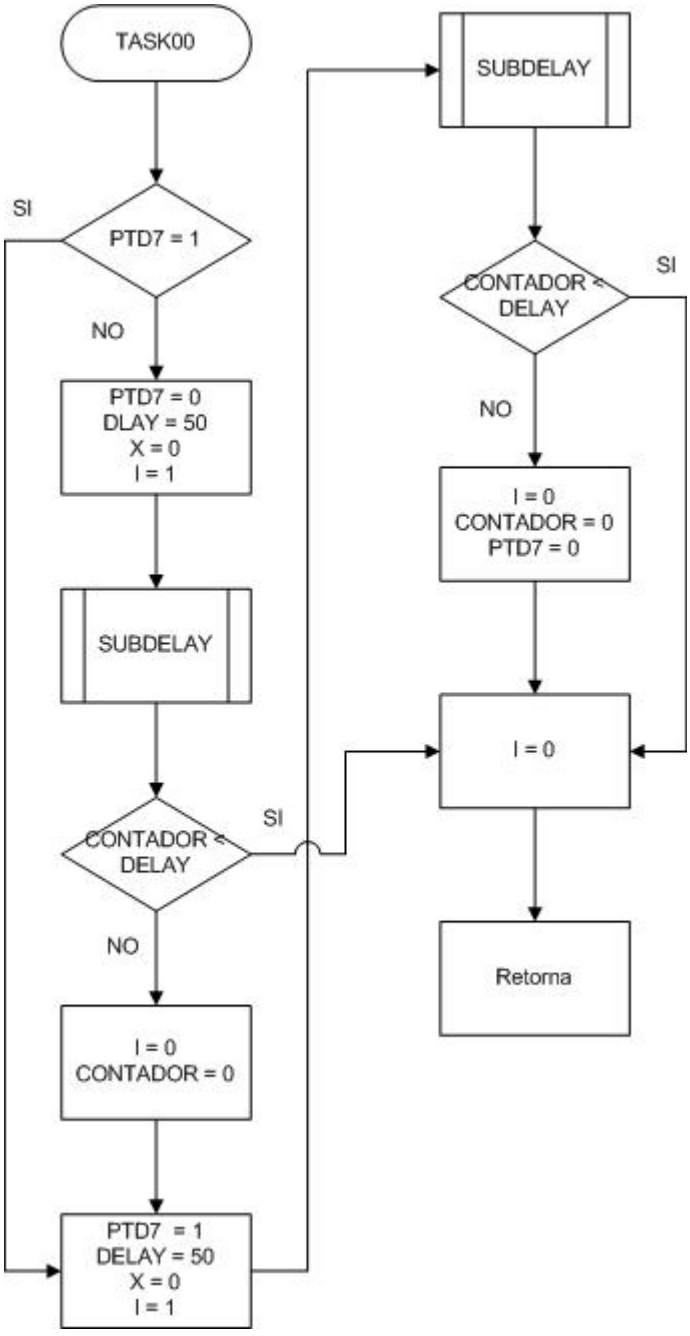

Figura No. 13: Diagrama de Flujo, Sección 3, Tarea00. Tarea o segmento de código que prende un LED y apaga el mismo LED por intervalos de 1 segundo.

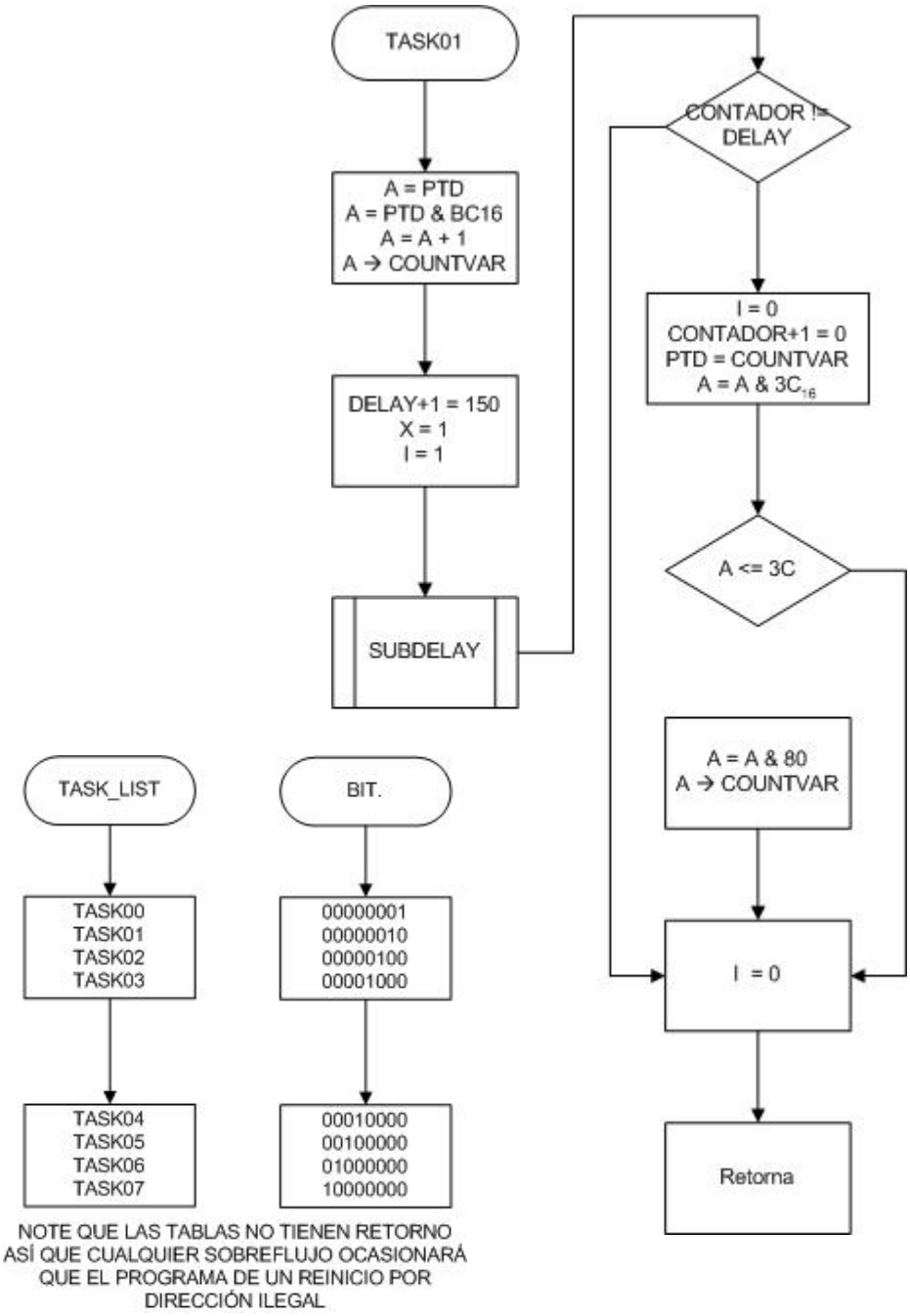

Figura No. 14: Diagrama de Flujo, Sección 3, Tarea01 (superior) y listas (inferior izquierda). La tarea 01 cuenta hasta 15 y las listas son secciones que se reservan en ROM para uso del Kernel.

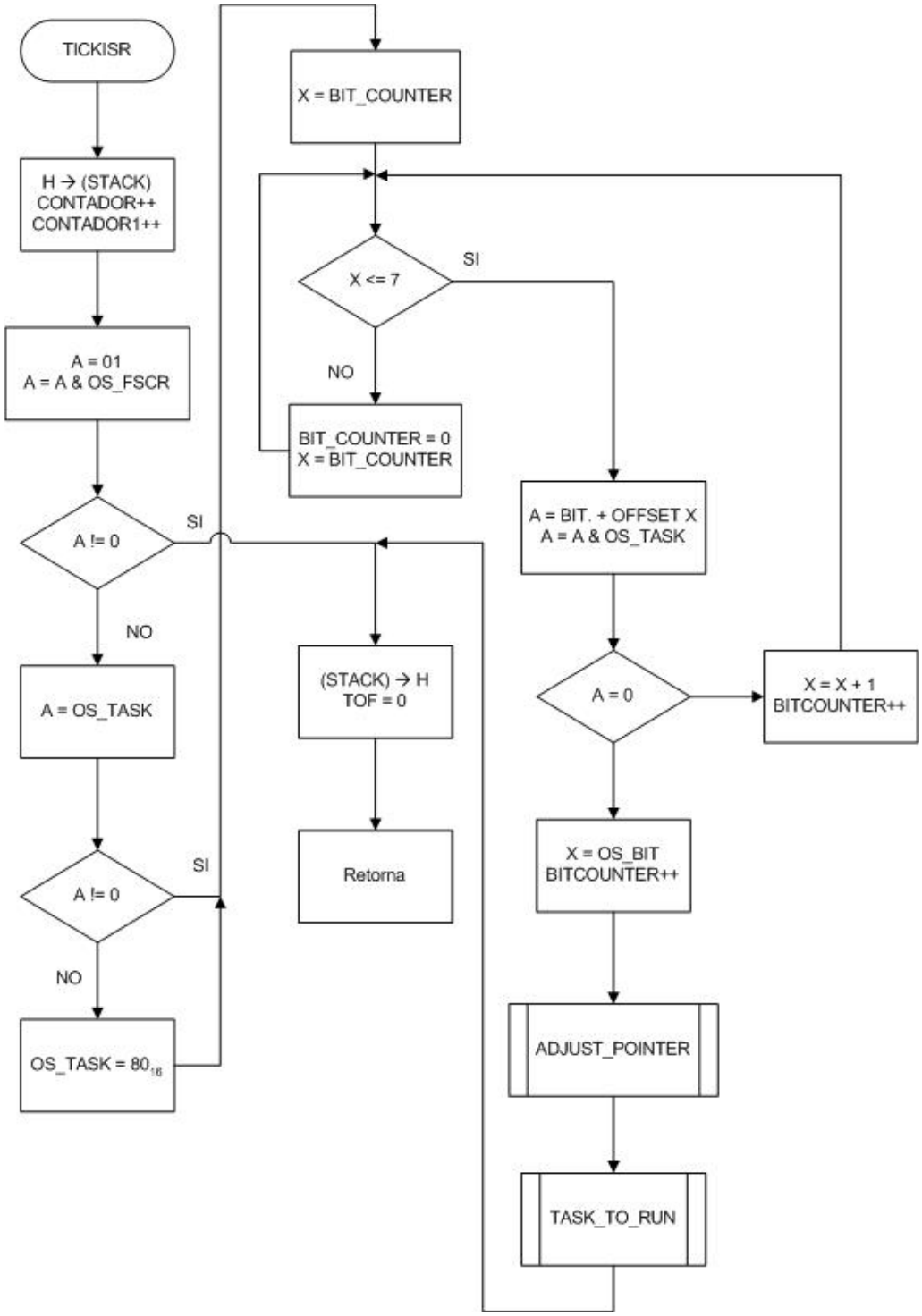

Figura No. 15: Diagrama de Flujo, Sección 4, Interrupcion del Sistema. La interrupción del sistema es periódica de 20 ms y decide que tarea ejecutar (corazón del kernel).

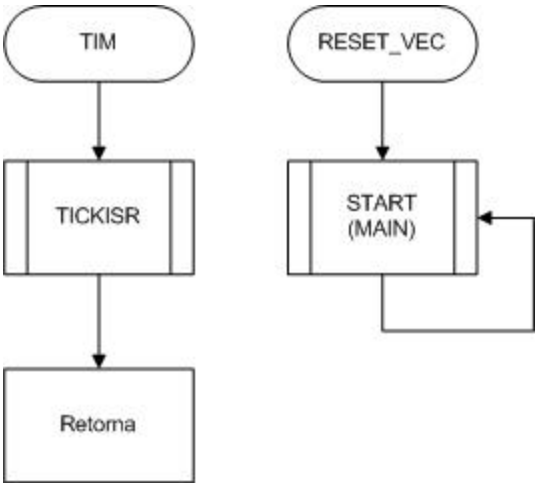

Figura No. 16: Diagrama de Flujo, Sección 4, Vectores de Usuario. Muestra los vectores de usuario programados y muestra que ejecutan al ser invocados.

Código

```
;=============================================;
;  Filename:         FARTK.asm                  ;
;  Revision #:       1                          ;
;  Author:           Rangel I. Alvarado         ;
;  -Use & Objectives-                           ;
;  Use:  Habilite suprimiendo el ";" de las siguientes lineas,  ;
;        dada las instrucciones.                ;
;  Objectives:  Prende el LED PTD7 por 1 seg. y cuenta  ;
;               ascendentemente hasta 15 por el ;
;               puerto D2 a D5 en intervalos de 3 segundos  ;
;=============================================;
;NOTA:  NO BLOQUEE EL KERNEL LA PRIMERA VEZ

;Sigua los siguientes pasos para generar su aplicación:

;PASO NO. 1:
; - HABILITE EL TICK DEL SISTEMA, PARA SIMULAR
;   ELIJA 015F Ó 6000 PARA SU APLICACIÓN, EJM.
;         TICK EQU $6000 (TICK DE 20 ms HABILITADO)
;         ;TICK EQU $015F (TICK RÁPIDO DESHABILITADO)

TICK            equ $6000               ;20 ms para una interrupción
;TICK           equ $015F               ;Interrupción rápida

;PASO NO. 2:
; - PROGRAME SUS APLICACIONES EN CÓDIGO DENTRO
;   DE LA SECCIÓN 3, EJEMPLO:
;
```

```
;TASK05
;      MI_APLICACIÓN
;      MÁS_DE_MI_APLICACIÓN
;      rts
;

;PASO NO. 3:
; - UBIQUE CUANTAS APLICACIONES TIENE TOTALES
;   Y COPIE EL NÚMERO BINARIO O HEXADECIMAL
;   CORRESPONDIENTE, EJEMPLO:
;   TASKS          equ $47
;   (TASKS 0,1,2,6 ARE ENABLED)

TASKS           equ $03                         ;Tareas estáticas (TASK00 & TASK01,
                                                ;enabled)

;=============================================;
;         Secciones en las que se segmenta el programa       ;
;=============================================;
;================================================;
;S00 $include 'equates.inc'                                    ;
;S01 $include 'main.inc'                                       ;
;S02 $include 'subs.inc'                                       ;
;S03 $include 'taskntable.inc'                                 ;
;S04 $include 'interrupt.inc'                                  ;
;================================================;

**********************************************
*           Sección00:  Equates              *
**********************************************
;=============================================;
; Igualdades Numéricas, números de bits y configuración de;
; registros                                                   ;
;=============================================;
NULL                    equ 0           ;Nulo
COPD                    equ 0           ;Bit0, Registro CONFIG1: COMPUTER
                                        ;OPERATION PROPERLY DISABLED
SCHEDULE_LOCK       equ 1               ;Bloquea el Kernel
TWO                     equ 2           ;Para ajustar el puntero de tareas (un puntero
                                        ;siempre es de 2 en 2)
TSTOP                   equ 5           ;Bit5: Registro TSC:  TIMER STOP
BIT7                    equ 7           ;Bit7 del Puerto
TOF                     equ 7           ;Bit7: Registro TSC:  TIMER STATUS AND
                                        ;CONTROL REGISTER
MAX_BITS                equ 7           ;Numero máximo de bits [empezando desde
                                        ;cero (0)]
DLAY_1SEC               equ 50T         ;Retardo de 1 segundos
DLAY_3SEC               equ 150T        ;Retardo de 3 segundos
ONE                     equ %00000100;Añade 1 a PTD2
MAX_COUNT               equ %00111100;Conteo Máximo (15)
BIT2-5&7                equ %10111100;Preserva los bits 2 @ 5 y 7
;TASKS                  equ $03         ;Tareas estáticas
TIM_CONFIG              equ $60         ;PS = 0, TSTOP = 1, TCHIE = 1
```

```
RAM_ORIGIN          equ $80       ;Inicio de la memoria RAM
IDLE_TASK           equ $80       ;Tarea Inactiva (si es 0 el bloque)
LED7_ON             equ $80       ;Bit 7 inicialmente alto (para el LEDPTD7)
BIT7M               equ $80       ;Máscara del Bit 7
OPCODE_RTS          equ $81       ;Código de Operación de RTS
RAM_USED            equ $8E       ;Fin del blanqueo de memoria RAM = $0080 +
                                  ;Posiciones a borrar + 1
OPCODE_JSR          equ $CD       ;Código de Operación de JSR
ALL_OUTPUTS         equ $FF       ;Todos como salidas
RAM_START           equ $0080     ;Origen de la memoria RAM

;=============================================;
;         MCU Memory Map or MCU Memory Map Usage     ;
;=============================================;
;=============================================;
;                    I/O Registers                    ;
;=============================================;
PORTD               equ $0003     ;Puerto de Registro de Control D
DDRD                equ $0007     ;Registro de Direccionamiento de Datos
TSC                 equ $0020     ;TIM Registro de Estado y Control
TMODH               equ $0023     ;Registro Alto, Módulo del
                                  ;Contador del TIM
TMODL               equ $0024     ;Registro Bajo, Módulo del Contador del TIM
CONFIG1             equ $001F     ;Registro de Configuración No. 1

;=============================================;
;              RAM (Preserve RAM Space)              ;
;=============================================;
OS_TASK             equ $0080     ;Almacena tareas
OS_BIT              equ $0081     ;Simboliza el bit en el que se está
BIT_COUNTER         equ $0082     ;Almacena el contador de bits
OS_TASK_RUN         equ $0083     ;Para ir a la tarea correcta
OS_PTASK            equ $0084     ;Almacena la tarea más prioritaria
OS_TASK_PTR         equ $0087     ;Almacena el Puntero de tareas
CONTADOR            equ $0088     ;Contador
CONTADOR1           equ $0089     ;Contador1
DELAY               equ $008A     ;Variable de retardo
DELAY1              equ $008B     ;Variable de retardo1
COUNTVAR            equ $008C     ;Variable de conteo BCD
OS_FSCR             equ $008D     ;Registro de Control del Kernel

;=============================================;
;                    Flash Memory                     ;
;=============================================;
FLASH_START         equ $EC00     ;Origen de la memoria FLASH

;=============================================;
;                 Vectores de Usuario                 ;
;=============================================;
TIM                 equ $FFF2     ;Dirección del vector de ;interrupción TIMER
RESET_VEC           equ $FFFE     ;Dirección del vector de interrupción RESET

***********************************************
*    S01:  Inicio del Programa Principal   *
```

```
*********************************************
;===========================================;
; Sección 01.1:  Inicializa variables y registros.          ;
; OBJETIVO:      Inicializa el sistema                      ;
; ENTRADA:       Ninguna                                    ;
; SALIDA:        Pila en 00FF, ejecución de la tarea siguiente ;
; REGISTROS AFECTADOS: Ninguno                              ;
; SECCIONES UTILIZADAS: S02.1, S02.2, S02.6                 ;
;===========================================;
            org FLASH_START       ;Inicio Mem. FLASH
START                             ;Al darse el reset, cae a ejecutar
                                  ;desde el inicio del programa
            rsp                   ;inic.Stack = $00FF
            bset COPD,CONFIG1     ;desactiva watchdog
            jsr CLEAR_OS_GLOBAL   ;S02.6 -> Borra RAM utilizada
            jsr OS_INIT_SYS       ;S02.1 -> Inicializa puertos y variables
            jsr OS_INIT_TICK      ;S02.2 -> Inicializa el Quantum o TimeSlice
            wait                  ;Alista el sistema (1 Tick)
S01.LOOP                          ;AQUÍ SE EJECUTARIA EL SCHEDULER
            jsr OS_TASK_RUN          ;Ejecuta la tarea
            bra S01.LOOP             ;Salta a hacer el lazo infinito

*********************************************
*         S02:  Subrutinas y Macros         *
*********************************************
;===========================================;
; Sección 02.1:  Inicializa variables y registros.          ;
; OBJETIVO:      Inicializa el sistema                      ;
; ENTRADA:       Ninguna                                    ;
; SALIDA:        Registros inicializados                    ;
; REGISTROS AFECTADOS: A, X, DDRD, RAM usada                ;
; SECCIONES UTILIZADAS: Ninguna                             ;
;===========================================;
OS_INIT_SYS                                   ;Inicializa puertos y variables
            clra                              ;A = 0
            clrx                              ;X = 0
            mov #LED7_ON,PORTD                ;LedPD7 Inicialmente
                                              ;encendido (PTD)
            mov #ALL_OUTPUTS,DDRD             ;Todos Salidas (PTD)
            mov #TASKS,OS_TASK                ;Mueve las tareas estáticas
            mov #OPCODE_JSR,OS_TASK_RUN       ;Almacena el código de
                                              ;operación en RAM
                                              ;(JSR=$CD)
            mov #OPCODE_RTS,OS_TASK_RUN+3     ;Almacena el código de
                                              ;operación en RAM
                                              ;(RTS=$81)
            rts                               ;retorna
;===========================================;
; Sección 02.2: Inicializa interrupcion del sistema         ;
; OBJETIVO: Inicializa el tick de 20 ms                     ;
; ENTRADA: H:X = TICK                                       ;
; SALIDA: TMOD[H:L] = TICK, Timer corriendo                 ;
; REGISTROS AFECTADOS: H:X, TSC, CCR                        ;
```

```
; SECCIONES UTILIZADAS: Ninguna                ;
; EJEMPLO:                                     ;
; Entrada:  H:X = $6000                        ;
; Salida:   TMOD[H:L]                          ;
;==============================================;
OS_INIT_TICK                                   ;Subrutina de tick de reloj del
                                               ;sistema
            ldhx #TICK                         ;H:X = TICK
            sthx TMODH                         ;H:X -> TMOD[H:L]
            mov #TIM_CONFIG,TSC                ;TOIE = TSTOP = 1
            ldhx #NULL                         ;H:X = 0
            cli                                ;Habilita interrupciones
            bclr TSTOP,TSC                     ;Habilita el temporizador
            rts                                ;retorna
;==============================================;
; Sección 02.3: Ajusta el puntero de tareas    ;
; OBJETIVO: Multiplica por 2 la posición del registro
;            de tareas                         ;
; ENTRADA:  OS_BIT                             ;
; SALIDA:   OS_TASK_PTR                        ;
; REGISTROS AFECTADOS: A, X, RAM (0081 y 0087) ;
; SECCIONES UTILIZADAS: Ninguna                ;
; EJEMPLO:                                     ;
; Entarda: A = OS_BIT = $04                    ;
; Salida: A = OS_TAKS_PTR = $08                ;
;==============================================;
ADJUST_POINTER                                 ;Ajusta el puntero de la tarea
            lda OS_BIT                         ;A = valor de prioridad
            ldx #TWO                           ;Carga con 2
            mul                                ;Multiplica 2*A
            sta OS_TASK_PTR                    ;Lo almacena en el puntero de
                                               ;direcciones
            rts                                ;retorna
;==============================================;
; Sección 02.4:  Determina que tarea correr    ;
; OBJETIVO:  Busca la tarea prioritaria        ;
; ENTRADA: OS_TASK_PTR, TASK_LIST              ;
; SALIDA: OS_PTASK, OS_PTASK+1                 ;
; REGISTROS AFECTADOS: A, X                    ;
; SECCIONES UTILIZADAS: Ninguna                ;
; EJEMPLO:                                     ;
; Entrada:  X = OS_TASK_PTR = $05              ;
;            A = TASK_LIST = TASK05            ;
; Salida:   OS_PTASK (byte alto de la tarea)   ;
;           OS_PTASK+1 (byte bajo de la tarea) ;
;==============================================;
TASK_TO_RUN                                    ;Determina que tarea correr
            ldx OS_TASK_PTR                    ;Carga variable OS_TASK en
                                               ;X
            lda TASK_LIST,x                    ;Carga parte alta de la palabra
                                               ;(WORD = 2 bytes) de
                                               ;dirección de la tarea
                                               ;prioritaria
            sta OS_PTASK                       ;Lo almacena en la parte alta
```

```
                                                ;de la memoria ram
                lda TASK_LIST+1,x               ;Carga parte baja de la
                                                ;palabra (WORD = 2 bytes) de
                                                ;dirección de la tarea
                                                ;prioritaria
                sta OS_PTASK+1                  ;Lo almacena en la parte baja
                                                ;de la memoria ram
                                                ;y lo almacena en 0083 = RTS
                rts                             ;retorna
;=============================================;
; Sección 02.5:  Comparador de retardos       ;
; OBJETIVO: Compara variables de retardo      ;
; ENTRADA: CONTADOR, DELAY                    ;
; SALIDA: Ninguna                             ;
; REGISTROS AFECTADOS: A                      ;
; SECCIONES UTILIZADAS: Ninguna               ;
; EJEMPLO:                                    ;
; Entrada:  A = CONTADOR = 50                 ;
;         DELAY = 150                         ;
; Salida:  Ninguna                            ;
;=============================================;
SUBDELAY                                        ;subrutina de retardo
                lda CONTADOR,x                  ;Carga la variable de retardo
                cmp DELAY,x                     ;comparo con contador
                rts                             ;retorna
;=============================================;
; Sección 02.6:  Borra la Ram utilizada       ;
; OBJETIVO: Borra registros en RAM            ;
; ENTRADA: RAM_ORIGIN, RAM_USED               ;
; SALIDA: Registros inicializados en 0        ;
; REGISTROS AFECTADOS: RAM, H:X               ;
; SECCIONES UTILIZADAS: Ningunaq              ;
; EJEMPLO:                                    ;
; Entrada RAM_ORIGIN = 0080, RAM_USED = 0090  ;
;       X = 7                                 ;
; Salida: 0080 @ 0087 = 0                     ;
;=============================================;
CLEAR_OS_GLOBAL                                 ;Borra registros a usar
                ldx #RAM_ORIGIN                 ;Carga con el origen
S02.FILL        clr ,x                          ;rellena con "0" la posición
                                                ;actual
                incx                            ;incrementa puntero de RAM
                cphx #RAM_USED                  ;Compara hasta el final
                                                ;deseado
                bne S02.FILL                    ;Si no concuerda entonces
                                                ;sigue limpiando
                rts                             ;retorna

*********************************************
*       S03:  Lista de Tareas y Tablas      *
*********************************************
;=============================================;
; Sección 03.1:  Lista de tareas en ROM       ;
```

```
; OBJETIVO:  Lista en ROM de tareas de 0 a 7                ;
; ENTRADA: Ninguna                                          ;
; SALIDA:  Ninguna                                          ;
; REGISTROS AFECTADOS: Ninguno                              ;
; SECCIONES UTILIZADAS: Ninguna                             ;
; EJEMPLO:                                                  ;
;===========================================;
TASK_LIST   ;B0      B1      B2      B3      B4      B5      B6      B7
        dw TASK00,TASK01,TASK02,TASK03,TASK04,TASK05,TASK06,TASK07
;===========================================;
; Sección 03.2:  Tareas a programar                         ;
; OBJETIVO: Tareas residentes en ROM activas o              ;
;               durmientes                                  ;
; ENTRADA: Depende del código de la tarea                   ;
; SALIDA:  Depende del código de la tarea                   ;
; REGISTROS AFECTADOS: Depende de la tarea                  ;
; SECCIONES UTILIZADAS: Depende de la tarea                 ;
;===========================================;

;===========================================;
; Sección 03.2.1:  Prende y apaga el LEDPTD7                ;
; OBJETIVO: Conmuta el LED en intervalos de 1 segundo       ;
; ENTRADA: DELAY, DLAY_1SEC, ...                            ;
; SALIDA:  PTD7 ON 1 SEG., PTD7 OFF 1 SEG.                  ;
; REGISTROS AFECTADOS: A, X, PORTD, RAM ...                 ;
; SECCIONES UTILIZADAS: S02.5                               ;
;===========================================;
TASK00                                          ;Prende y apaga un led
            brset BIT7,PORTD,S03.BITON          ;Si esta encendido, enciendelo
                                                ;por un tiempo T, sino,
                                                ;apágalo por el mismo tiempo
                                                ;T
            bclr BIT7,PORTD                     ;LedPD7 Apagado
            mov #DLAY_1SEC,DELAY                ;Carga del temporizador T =
                                                ;20*DELAY (en ms)
            ldx #0                              ;Apunta a Delay,x donde X = 0
            sei                                 ;Deshabilita interrupciones
            jsr SUBDELAY                        ;S02.5 -> Apunta al Delay
                                                ;correspondiente
            blo S03.SALIR                       ;Si no son iguales
                                                ;(CONTADOR Y DELAY) sal y
                                                ;no esperes
            cli                                 ;Habilita interrupciones
            clr CONTADOR                        ;Borra el contador (TICKISR)
S03.BITON   bset BIT7,PORTD                     ;Prende el PORTD7
            mov #DLAY_1SEC,DELAY                ;Carga del temporizador T =
                                                ;20*DELAY (en ms)
            ldx #0                              ;Apunta a 0
            sei                                 ;Deshabilita interrupciones
            jsr SUBDELAY                        ;S02.5 -> Apunta al Delay
                                                ;correspondiente
            blo S03.SALIR                       ;Si no son iguales
                                                ;(CONTADOR Y DELAY) sal y
                                                ;no esperes
```

```
                cli                                     ;Habilita interrupciones
                clr CONTADOR                            ;Borra el contador
                bclr BIT7,PORTD                         ;LedPD7 apagado
S03.SALIR       cli                                     ;Habilita interrupciones
                rts                                     ;retorna
;=========================================;
; Sección 03.2.1:  Prende y apaga el LEDPTD7            ;
; OBJETIVO: Cuenta en intervalos de 3 segundos          ;
; ENTRADA: PTD, DELAY+1, DLAY_3SEC, ...                 ;
;          COUNTVAR, CONTADOR+1 ...                     ;
; SALIDA:  PTD[2:5] ON 3 SEG. PTD[2:5] OFF 3 SEG.       ;
; REGISTROS AFECTADOS: A, X, PORTD, RAM ...             ;
; SECCIONES UTILIZADAS: S02.5                           ;
;=========================================;
TASK01                                                  ;Conteo BCD hasta 15
                lda PORTD                               ;Carga el valor del Puerto
                and #BIT2-5&7                           ;A = A && PORTD (Máscaras
                                                        ;de PTD2@5, PTD7)
                add #ONE                                ;A = A + 1
                sta COUNTVAR                            ;A -> COUNTVAR (variable de
                                                        ;conteo BCD)
                mov #DLAY_3SEC,DELAY+1                  ;Retardo de 3 segundos
                ldx #$01                                ;Cargo el Delay,x donde X = 1
                sei                                     ;Deshabilita interrupciones
                jsr SUBDELAY                            ;Salto a cargar
                bne S03.SALIR0                          ;Si es menor o igual salgo
                cli                                     ;habilita interrupciones
                clr CONTADOR+1                          ;Sino, se completo el retardo y
                                                        ;borro el contador
                mov COUNTVAR,PORTD                      ;Contador BCD->PORTD
                and #MAX_COUNT                          ;And lógico con el máximo de
                                                        ;la cuenta (15)
                cmp #MAX_COUNT                          ;Comparo con el máximo de la
                                                        ;cuenta (15)
                ble S03.SALIR00                         ;Si no es igual salir
                and #BIT7M                              ;Si lo es preserva el estado
                                                        ;del BIT7 encendido
                sta COUNTVAR                            ;A -> COUNTVAR
S03.SALIR0      cli                                     ;habilita interrupciones
                rts                                     ;retorna
;=========================================;
; Sección 03.2.2:  Tareas Durmientes                    ;
; OBJETIVO:  Tareas residentes en ROM y no activadas     ;
; ENTRADA: Ninguna                                      ;
; SALIDA:  Ninguna                                      ;
; REGISTROS AFECTADOS: Ninguno                          ;
; SECCIONES UTILIZADAS: Ninguna                         ;
;=========================================;
TASK02
                rts
TASK03
                rts
TASK04
```

```
                rts
TASK05
                rts
TASK06
                rts
TASK07
                rts
;=============================================;
; Sección 03.3:  Lista de Bit encendido       ;
; OBJETIVO:  Verifica que bit está encendido  ;
; ENTRADA: Ninguna                            ;
; SALIDA:  Ninguna                            ;
; REGISTROS AFECTADOS:  Ninguno               ;
; SECCIONES UTILIZADAS: Ninguna               ;
;=============================================;
BIT.                                          ;Prueba el Bit de la
                                              ;Tarea
                db %00000001                  ;Bit 0 encendido
                db %00000010                  ;Bit 1 encendido
                db %00000100                  ;Bit 2 encendido
                db %00001000                  ;   .
                db %00010000                  ;   .
                db %00100000                  ;   .
                db %01000000                  ;   .
                db %10000000                  ;Bit 7 encendido

************************************************
*S04:  Pedido de Interrupción del Sistema  *
*y vectores de Interrupción                *
************************************************
;=============================================;
; Sección 04.1:  Corazon del Kernel           ;
; OBJETIVO:  Decide que tarea sigue           ;
; ENTRADA: CONTADOR, CONTADOR1, OS_TASK,      ;
;         BIT_COUNTER                         ;
; SALIDA:  Tarea Siguiente                    ;
; REGISTROS AFECTADOS: A, H:X, RAM            ;
; SECCIONES UTILIZADAS: S02.3, S02.4          ;
;=============================================;
TICKISR
                                              ;Las siguientes 5  ;líneas  y el
                                              ;label
                                              ;SEC04.NORM_OP, ;pueden
                                              ;ser quitados, pero siempre
                                              ;tiene que haber una tarea en
                                              ;el sistema
                pshh                          ;empuja H a la pila
                inc CONTADOR                  ;Contador de 1 byte
                                              ;(actua como el temporizador)
                inc CONTADOR1                 ;Contador de 1 byte (para
                                              ;contar secuencialmente)
                lda #SCHEDULE_LOCK            ;Carga variable de bloqueo del
                                              ;kernel
                bit OS_FSCR                   ;A = SCHEDULE_LOCK & A
```

```
                bne S04.LOC                     ;A = 0?, si no es igual salta a
                                                ;salir del kernel
                lda OS_TASK                     ;Carga las tareas
                bne S04.NOR                     ;Si las hay sigue la operación
                                                ;normal
                mov #IDLE_TASK,OS_TASK          ;Sino, ve a la tarea inactiva
                                                ;Máximo = 0.16s
S04.NOR     ldx BIT_COUNTER                     ;Bit en el que se está parado
S04.NEXT    cmpx #MAX_BITS                      ;compara BITCOUNTER con
                                                ;7
                ble S04.COM                     ;Si es <= salta a comparar
                clr BIT_COUNTER                 ;sino, hay un sobreflujo y
                                                ;borra
                ldx BIT_COUNTER                 ;X = BITCOUNTER = 0
                bra S04.NEX                     ;Salta a buscar tarea inicial
S04.COM     lda BIT.,x                          ;A = Bit de la tabla en la
                                                ;posición X
                and OS_TASK                     ;A = A && OS_TASK
                beq S04.INC                     ;No hay, entonces revisa el
                                                ;siguiente bit
                stx OS_BIT                      ;X -> OS_BIT
                inc BIT_COUNTER                 ;incremento BIT_COUNTER
                                                ;EVALUO A QUE TAREA
                                                ;SALTAR
                jsr ADJUST_POINTER              ;S02.3 -> Ajusta el Puntero
                jsr TASK_TO_RUN                 ;S02.4 -> Decide que tarea es
                                                ;prioritaria
S04.LOC     pulh                                ;retorna H de la Pila
                bclr TOF,TSC                    ;Borra la interrupción
                rti                             ;retorno
S04.INC      incx                               ;incrementa X
                inc BIT_COUNTER                 ;incremento BIT_COUNTER
                bra S04.NEXT                    ;reviso el siguiente bit
;=============================================;
; Sección 04.2:  Vectores de Interrupción     ;
; OBJETIVO:  Vectores asíncronos del usuario  ;
; ENTRADA: Ninguna                            ;
; SALIDA:  Ninguna                            ;
; REGISTROS AFECTA DOS: Ninguno               ;
; SECCIONES UTILIZADAS: Ninguna               ;
;=============================================;
;===== Vector de Interrupción del Temporizador =====;
                org TIM                         ;Puntero del Vector del
                                                ;Temporizador
                dw TICKISR                      ;Busca la dirección a la que
                                                ;apunta TIM
;===== Vector de Reinicio de Sistema =======;
                org RESET_VEC                   ;Puntero del Vector Reinicio
                dw START                        ;Busca la dirección a la que
                                                ;apunta Start
```

Referencias

(1) Curso de un PLC que explica como se ejecutan las tareas en estos sistemas
http://scsx01.sc.ehu.es/sbweb/webcentro/automatica/WebCS1/orden_de_ejecucion_de_las_tareas_ciclicas.htm

(2) Sistema Multitarea con el microcontrolador 68HC11
http://www.geocities.com/isusnea/mtask.html

(3) Sección de sistemas de tiempo real
http://www.embedded.com.ar/

(4) Página del uso de µC/OS-II The Real Time Kernel
http://www.engineering.usu.edu/classes/ece/5780/node6.html

(5) Filosofía y comparación de un Kernel (DARK) con µC/OS
http://web-cat.cs.vt.edu/PEBB/CPES02-Singh.pdf

(6) Análisis de Tiempo de Sistemas de Tiempo Real
http://www.realtime-info.be/magazine/01q3/2001q3_p010.pdf

(7) Conceptos Básicos de Sistemas de Tiempo Real
http://www.eletro.ufrgs.br/~cpereira/temporeal_pos/www/rt-labross_intro.htm
http://www.algonet.se/~staffann/developer/rtbasics.htm#Synronisation%20and%20communication%20primitives

(8) Ver Filmina 15, habla sobre µKernels vs. Kernels.
http://laurel.datsi.fi.upm.es/~ssoo/SOD/Archive/introduccion2001-6pp.pdf

(9) Página que desglosa un sistema de tiempo real de la A - Z
http://gsyc.escet.urjc.es/~caguero/proyectos/pera/node2.html

(10) PDF, sincronización de tareas y método de selección de cada una de ellas.
http://www.disa.bi.ehu.es/spanish/asignaturas/17225/Tareas_comunicacion_sincronizacion_I.pdf
http://www.linti.unlp.edu.ar/catedras/Laboratorio/Teorias/clase8.PDF

(11) Robot Eyebot, su núcleo es un sistema multitareas
http://gsyc.escet.urjc.es/robotica/manual-eyebot/manualeyebot.html

(12) Planificación de Sistemas de Tiempo Real
http://atc1.aut.uah.es/~sistel/Trps/Planificacion_de_tareas_en_STR.ppt
http://apollo.cps.unizar.es/~joseluis/SE_Nucleos.ppt
http://apollo.cps.unizar.es/~joseluis/SE_Nucleos.ppt