Arquitectura de computadores
Práctica 4: Interrupciones

4.1. OBJETIVO.

El objetivo de esta práctica es el estudio del grupo más importante de excepciones del 68000 (causadas por eventos externos): las interrupciones.

Hasta ahora se han hecho prácticas con el microprocesador 68000 que no precisaban de más elementos de hardware que la memoria necesaria para almacenar el programa y las variables de la ejecución. Pero un sistema basado en un microprocesador debe incorporar otros elementos. Los fabricantes de microprocesadores fabrican también toda una gama de circuitos integrados especializados en determinadas tareas, entre las cuales se encuentran las de temporización, entrada/salida, etc. Estos circuitos forman lo que se conoce como "familia" de periféricos de un microprocesador. En esta práctica introduciremos un circuito de la familia del 68000, concretamente el 68230, diseñado como interface de entrada/salida en paralelo, y que además incorpora un temporizador capaz de generar interrupciones.

4.2. PRÁCTICA.

En la práctica anterior se han estudiado las excepciones generadas por el 68000 en respuesta a condiciones especiales de naturaleza interna. En esta práctica se estudian las excepciones producidas en respuesta a condiciones especiales sucedidas en circuitos externos al microprocesador: las interrupciones.

4.2.1. El sistema de interrupciones.

Uno de los aspectos fundamentales de los microprocesadores son las interrupciones. Todo microprocesador posee un sistema de interrupciones más o menos sofisticado que le permite atender eventos asíncronos. Debido a esta capacidad es posible liberar al microprocesador de la tarea de atender continuamente a todos sus dispositivos externos. En lugar de esto, cuando cualquier dispositivo requiere la atención del microprocesador, simplemente genera una petición de interrupción para aquél. Si tal interrupción es suficientemente importante, el microprocesador abandonará momentáneamente la tarea que se hallaba realizando, atenderá al dispositivo (mediante una rutina de servicio), y después retornará al estado anterior continuando con su tarea. Este esquema de funcionamiento es especialmente eficaz cuando los eventos causantes de las interrupciones son relativamente esporádicos. Esto sucede, por ejemplo, en las comunicaciones de velocidad media o baja, temporizadores, alarmas, etc.

Un símil adecuado de un sistema de interrupciones es, por ejemplo, el timbre de un teléfono: éste suena cuando alguien llama a nuestro número, por lo que no es necesario que descolguemos el teléfono de vez en cuando para saber si hay alguien en la línea. Obviamente ello nos permite atender a todo tipo de tareas sin preocuparnos del aparato. Siguiendo con el símil telefónico, cuando recibimos una llamada no tenemos por qué atenderla obligatoriamente; ello depende de lo ocupados que nos encontremos en ese momento y de la importancia de la tarea que estemos realizando. Además a nuestro número pueden telefonear varias personas simultáneamente, pero sólo una conseguirá establecer la comunicación, mientras que las demás tendrán que esperar a que aquélla finalice. La centralita telefónica (análoga al controlador de interrupciones del ordenador) se encarga de gestionar estos casos.

Aunque las ventajas de las interrupciones son evidentes, también poseen dos inconvenientes de importancia. En primer lugar, los sistemas de interrupciones requieren hardware adicional más o menos complejo según el tipo de microprocesador. En segundo lugar, la programación se hace más

difícil: las interrupciones, como ya hemos dicho, son causadas por eventos de naturaleza asíncrona, lo que significa que pueden producirse en cualquier instante. Esto debe ser tenido muy en cuenta por el programador, quien debe prever el hecho de que el microprocesador puede abandonar su tarea en cualquier momento y pasar a ejecutar otro programa (la rutina de servicio de la interrupción), el cual puede interferir de algún modo con la tarea que estaba siendo realizada anteriormente. A esto hay que añadir otra dificultad: los posibles fallos de un programa que utiliza el sistema de interrupciones no tienen por que reproducirse en cada ocasión en que ese programa es ejecutado. La razón es simple: una interrupción se puede producir durante una ejecución, mientras que en la siguiente cabe la posibilidad de que no se produzca, o de que ocurra en un momento diferente, por lo que los efectos pueden ser distintos en diferentes ejecuciones del mismo programa. Todo ello complica en gran medida la detección y corrección de los posibles errores.

4.2.2 El mecanismo de interrupciones del 68000.

Como ya sabemos, las interrupciones son tratadas por el 68000 como un tipo más de excepciones. Su particularidad principal consiste en que el número de vector no se genera automáticamente por el microprocesador, sino que es suministrado por el dispositivo causante de la interrupción, y obtenido por el 68000 mediante un ciclo de lectura de byte (salvo en el caso de las interrupciones autovectorizadas, que no estudiaremos debido a que el 68fil no las utiliza).

Recordemos que existen siete niveles de interrupción, siendo el nivel 7 el de mayor prioridad. Una interrupción es atendida por el 68000 si su nivel de prioridad es estrictamente mayor que el del microprocesador (que se halla codificado en los bits I0-I2 del `SR`), salvo en el caso del nivel 7, "no enmascarable". Si el nivel de la interrupción es menor o igual que el del microprocesador, entonces la interrupción se deja pendiente. Caso de ser atendida, la interrupción provoca una excepción; su secuencia de procesamiento es la habitual salvo en dos aspectos: primero, el microprocesador adopta el mismo nivel de prioridad que el de la interrupción; y segundo, el nº del vector se obtiene mediante un ciclo de lectura. La información que se salva en la pila consiste en el `SR` y el `PC` previos a la excepción.

Observación:

Es normal que el dispositivo causante de la interrupción mantenga la petición hasta que el microprocesador efectúe con él alguna operación determinada; tal operación se realiza generalmente dentro de la rutina de servicio de la interrupción. Ello significa que la petición puede continuar activa en el momento en que el 68000 comienza a ejecutar la rutina de servicio. Por tanto, es muy importante el hecho de que el microprocesador adopte el mismo nivel de prioridad que el de la interrupción recibida. Si no ocurriera de ese modo, en el preciso momento en que el 68000 iniciase la ejecución de la rutina de servicio volvería a atender la misma interrupción y se produciría un ciclo infinito.

El nivel 7 de interrupción es "no enmascarable". Esto significa que las peticiones de interrupción de ese nivel son atendidas por el microprocesador aunque éste se encuentre a la máxima prioridad (7). Por otro lado, para que el 68000 detecte una petición de interrupción de nivel 7 se ha de producir un cambio en las líneas IPL0-IPL2 de cualquier nivel más bajo (0-6) a nivel 7. De este modo, una vez que las líneas IPL0-IPL2 están a 7 aunque la interrupción de nivel 7 siga activa, el microprocesador no la atenderá de nuevo. Sí lo hará, sin embargo, si alguna instrucción reduce el nivel de prioridad del 68000 (cualquiera que afecte al SR, incluida RTE, puede hacerlo) mientras continua activada la petición de interrupción de nivel 7.

Por ejemplo:

En el 68fil es posible provocar una interrupción de nivel 7 mediante el pulsador ABORT. La interrupción se produce en el momento de soltar el pulsador tras haberlo presionado. Cuando

durante el procesamiento de la excepción el 68000 realiza el ciclo de lectura del nº de vector, un dispositivo pone en la parte baja del bus de datos (D0-D7) el valor $0C_{16}$ (12_{10}), por lo que el microprocesador extrae el nuevo contenido del PC de las posiciones de memoria 000030_{16}-000033_{16}. La petición de interrupción es retirada automáticamente durante la obtención del número de vector.

En varias ocasiones habremos hecho uso del pulsador ABORT para detener un programa en funcionamiento; ahora podemos saber por qué funciona. El monitor del 68fil instala una rutina de servicio para esta interrupción cada vez que se ejecuta un programa. Básicamente, esa rutina se limita a exponer un mensaje en el terminal y finalizar la ejecución del programa. También hay instalada una rutina de servicio para el ABORT cuando el monitor no se halla ejecutando un programa de usuario, pero tal rutina se reduce a una instrucción RTE.

Nada nos impide que nuestros programas instalen sus propias rutinas de servicio para cualquier interrupción u otro tipo de excepción. Por ejemplo, el siguiente programa instala una rutina cuya única función consiste en finalizar la ejecución de la forma habitual. Una vez hecho esto, el programa se introduce en un bucle en el que se limita a incrementar el contenido del registro D7 en cada vuelta:

```
NUMVEC      EQU     12              ;Número del vector ABORT

            ORG     $1000

            MOVE.L  #RUTNIV7,4*NUMVEC  ;Instala rutina de servicio
            CLR.L   D7              ;Inicializa contador a cero
CICLO       ADDQ.L  #1,D7           ;Ciclo sin fin
            BRA.S   CICLO

RUTNIV7     TRAP    #15             ;Rutina de servivio del ABORT
            DC.W    0               ;que termina la ejecución.
```

Como práctica:

Introducir en el 68fil el programa del ejemplo anterior. Ejecútese en modo usuario y con nivel de prioridad menor que 7, comprobando su funcionamiento correcto. Una vez detenido el programa, es posible observar cómo el SR indica que el 68000 se encuentra en modo supervisor y nivel de prioridad 7. Compruébese que el programa funciona igual si se ejecuta con nivel de prioridad 7, lo que demuestra que el nivel 7 de interrupción es "no enmascarable".

4.2.3. Rutinas de servicio de las interrupciones.

En el ejemplo anterior la rutina de servicio se limita a detener el programa; esto no es lo habitual. En general, las rutinas de servicio de las interrupciones realizan una secuencia relativamente corta de acciones, tras las cuales se devuelve el control al punto donde se produjo la interrupción mediante la instrucción RTE.

El tiempo de ejecución de las rutinas de servicio de las interrupciones debe ser relativamente corto. De no ser así se produce una sobrecarga de tareas en el microprocesador, lo que implica una ralentización del programa principal.

Como ya sabemos, durante la ejecución de la rutina de servicio el 68000 adopta el mismo nivel de prioridad que el de la interrupción que está siendo atendida. Ello implica que, en esos momentos, toda petición de interrupción cuyo nivel de prioridad sea menor o igual no será atendida en ese instante. Si el controlador de interrupciones funciona correctamente, mantendrá las peticiones de interrupción hasta que sean atendidas por el 68000; por tanto, en el momento que el nivel de prioridad del microprocesador baje de nuevo (por ejemplo, tras el RTE) las interrupciones pendientes irán siendo atendidas por el 68000 en orden de prioridad. Sin embargo, en ciertas aplicaciones (especialmente las que deben funcionar en tiempo real) se requiere una respuesta muy rápida a la mayoría de las interrupciones. Si además el nivel de prioridad de cada interrupción no se encuentra asignado correctamente según su importancia, entonces puede ser necesario disminuir el nivel de prioridad del microprocesador dentro de la rutina de servicio de una interrupción de mayor prioridad, con el objetivo de que otras interrupciones puedan ser atendidas de forma inmediata.

Puesto que las interrupciones se pueden producir en cualquier momento, es de vital importancia que sus rutinas de servicio no modifiquen ninguno de los registros del microprocesador ni, por supuesto, el contenido de ninguna de las dos pilas. Por tanto, todo registro cuyo contenido se vea alterado durante la ejecución de la rutina de servicio debe ser salvado en la pila del supervisor al comienzo de la rutina para ser recuperado antes del retorno de excepción. Toda transferencia de información entre el programa y las rutinas de servicio de interrupción debe realizarse a través de posiciones fijas de memoria.

Por ejemplo:

Vamos a ver a continuación un programa cuya única actividad consiste en incrementar continuamente un contador y presentar su valor en el terminal. Es posible poner a cero el contador pulsando el interruptor ABORT. Para realizar esta sencilla tarea basta con mantener el valor del contador en una posición de memoria; la rutina de servicio de la interrupción ABORT escribe un cero en esa posición.

```
             INCLUDE  /usr/local/68k/sermones.inc

NUMVEC       EQU      12            Número del vector ABORT

             ORG      $1000

             MOVE.L   #RUTNIV7,4*NUMVEC  ;Instala rutina de servicio.
             CLR.L    CONT        ;Inicializa contador a cero.
             LEA      CADENA,A0   ;A0 <- comienzo de la cadena.
             MOVEA.L  A0,A1       ;Salva una copia de A0 en A1.
CICLO        MOVE.L   CONT,D7     ;Convierte el contador a código
             SERMON   ITOA        ;ASCII dentro de la cadena.
             MOVEA.L  A1,A0       ;Recupera el valor previo de
             SERMON   LNPUTS      ;A0 y muestra la cadena en pantalla.
             ADDQ.L   #1,C0NT     ;Incrementa el contador.
             BRA.S    CICLO

;---------------------------------------------------------------------
;Rutina de servicio de INTERRUPCION DE NIVEL 7 (ABORT)
;Función: Pone contador a cero (CONT(L)=0)

RUTNIV7      CLR.L    CONT        ;Contador a cero.
             RTE                  ;y retorna.
;---------------------------------------------------------------------

             ORG      $2000

CONT         DS.L     1           Contador (largo)
CADENA       DS.B     9           Cadena (8 del número + terminador)
```

Como práctica:

Introdúzcase en el 68fil el programa del ejemplo anterior y estúdiese su funcionamiento. Compruébese cómo se inicializa el contador a cero cada vez que se presiona el interruptor ABORT.

4.2.4. Temporización mediante interrupciones.

Vamos a ver a continuación cómo es posible utilizar el temporizador (*timer*) incluído en el PI/T 68230 en combinación con el sistema de interrupciones del 68000.

El temporizador del 68230 consiste en un contador descendente síncrono de 24 bits. El contenido del contador se decrementa con cada pulso de una señal de entrada (la cual no tiene por que ser periódica). Tal señal puede ser de procedencia externa, siendo introducida a través de su entrada `TIN`; o interna, procedente de la señal `CLK` aplicada al 68230. En este último caso, la señal `CLK` pasa a través de un divisor de 5 bits que reduce su frecuencia en un factor de 32 (2^5); por tanto, dado que en el 68fil la frecuencia de `CLK` es 8 MHz, la señal interna de entrada al timer es de 250 KHz.

Cuando el contenido del contador llega a cero, automáticamente se activa el bit `ZDS` (Zero Detect Status) del registro de estado `TSR` del temporizador. Además, su señal de salida `TOUT` cambia de estado de una forma u otra dependiendo del modo de funcionamiento. `TOUT` se puede utilizar, por ejemplo, para generar interrupciones periódicas, ondas cuadradas, etc.

La programación del temporizador es muy sencilla: se configura mediante un único registro de control (`TCR`), con el que podemos establecer diversos modos de funcionamiento. Sin embargo, el `TCR` no es el único registro del timer. Éste posee en total cinco registros, que vamos a describir a continuación:

• **Timer Control Register (`TCR`).**

Este registro es el nº 16 del PI/T 68230. Por tanto en el entrenador 68fil se encuentra en la dirección 800021_{16} (todos los registros del PI/T 68230 están mapeados en direcciónes impares). El significado de sus bits se muestra a continuación:

7 6 5	4	3	2 1	0
Control de TOUT/TIACK	Control de Z.D.	*	Control de entrada	Timer ON/OFF

`TCR [7-5]`: Estos tres bits configuran las señales `TOUT` (salida) y $\overline{\texttt{TIACK}}$ (entrada) del timer. Las combinaciones más utilizadas son:

`[01X]:` `TOUT` genera una onda cuadrada
$\overline{\texttt{TIACK}}$ no tiene significado.

`[101]:` `TOUT` genera una petición de interrupción
$\overline{\texttt{TIACK}}$ es un reconocimiento de interrupción.

[111]: TOUT genera una petición de interrupción
$\overline{\text{TIACK}}$ no tiene significado.

TCR [4]: Si este bit es cero, cada vez que el contador llega a cero éste se vuelve a cargar con el contenido del registro de precarga (CPR- *Counter Preload Register*). Si es uno, el contador pasa de 000000_{16} a $FFFFFF_{16}$ y continua su descenso.

TCR [3]: Este bit no tiene significado.

TCR [2-1]: Estos bits especifican el significado de las señales TIN y CLK para el temporizador. Las combinaciones posibles son:

[00]: TIN no se utiliza
CLK se utiliza como base de tiempo a través del divisor

[01]: TIN se utiliza como activación/desactivación del temporizador
CLK se utiliza como base de tiempo a través del divisor

[10]: TIN se utiliza como base de tiempo a través del divisor
CLK no se utiliza

[11]: TIN se utiliza como base de tiempo directamente
CLK no se utiliza

TCR [0]: Este bit determina si el temporizador está parado (TCR[0]=0) o en marcha (TCR[0]=1).

· Timer Interrupt Vector Register (TIVR).

Este registro contiene el nº del vector de interrupción que es suministrado por el PI/T 68230 en respuesta a la señal $\overline{\text{TIACK}}$ cuando aquél tiene activada una petición de interrupción mediante TOUT. Para que esto ocurra, los bits TCR [7-5] deben ser [1 0 1]. El contenido inicial de este registro es el número $0F_{16}$ (nº de vector 15: interrupción no inicializada). TIVR es el registro nº 17 del 68230, y ocupa por tanto la posición 800023_{16} en el 68fil. Lo apropiado es configurar un número de vector entre el 64 y el 255 en este registro.

· Counter Preload Register (CPR).

Es éste un registro de 32 bits cuyos 24 bits más bajos constituyen la cuenta inicial (precarga) que se transfiere al contador del timer cuando éste comienza a funcionar, o cuando alcanza el valor cero hallándose el bit TCR [4] desactivado. Este registro se puede leer y escribir en cualquier momento sin que ello afecte al funcionamiento del timer. CPR consta de cuatro registros consecutivos de 8 bits del 68230: los números 18 (que siempre es cero), 19, 20 y 21, que ocupan en el 68fil las direcciones impares consecutivas 800025_{16}, 800027_{16}, 800029_{16} y $80002B_{16}$. Para acceder al CPR en su totalidad lo más sencillo es utilizar la instrucción del 68000 MOVEP.L (Mover a/de Periféricos), utilizando como origen o destino la dirección base del CPR (800025_{16}).

· Count Register (CNTR).

Este registro posee el mismo formato que el CPR. Su contenido representa el valor actual del contador del timer. Es un registro que sólo admite lectura; ésta, además, únicamente se debe realizar cuando el timer está detenido. Consultando el CNTR es posible saber con gran precisión cuánto tiempo ha transcurrido desde que comenzó la cuenta descendente. El CNTR se compone de los registros nº 22 (que siempre es cero), 23, 24 y 25, situados en las direcciónes del 68fil $80002D_{16}$, $80002F_{16}$, 800031_{16} y 800033_{16} respectivamente.

• **Timer Status Register (TSR).**

El único bit significativo del TSR es el bit 0, denominado ZDS: estado de detección de cero. Este bit, realizado mediante un flip-flop sensible a flancos, se activa (TSR[0]=1) cuando el contenido del contador del timer (CNTR) pasa de $000001 a $000000. *Después ZDS continúa activado hasta que es puesto a cero mediante una operación de borrado directo*, o hasta que el timer es detenido mediante un RESET externo. El borrado directo se efectúa mediante una operación de escritura de **un 1** en el bit ZDS del TSR. Es decir, para hacer TSR[0]=0, basta con realizar un BSET de ese mismo bit.

El bit ZDS del TSR es responsable de la activación del TOUT como petición de interrupción: así, la única forma de desactivar la interrupción es borrar el bit ZDS. Por tanto es fundamental que la rutina de servicio de una interrupción provocada por el timer se encargue de poner a cero el bit ZDS del TSR mediante una operación de borrado directo. Si esto no se hiciera, la interrupción continuaría activa tras el retorno desde la rutina de servicio y la consiguiente vuelta del 68000 a un nivel menor de prioridad; como consecuencia el microprocesador volvería a atender la misma interrupción, repitiéndose todo el proceso de forma indefinida.

TSR es el registro nº 26 del 68230, ocupando por tanto la dirección 800035_{16} en el entrenador 68fil.

Por ejemplo:

Vamos a ver a continuación un programa que configura el *timer* de forma que produce una interrupción cada segundo. La rutina de servicio de esa interrupción incrementa en uno una posición de memoria de forma que su contenido nos indica cuántos segundos han pasado desde la puesta en marcha del programa. También visualiza en el terminal un mensaje indicando los segundos transcurridos. Mientras tanto, el programa principal se concentra en un bucle esperando a que pasen quince segundos (es decir, a que se produzcan otras tantas interrupciones). Transcurrido ese tiempo, el programa detiene el *timer* y finaliza. Para fijar el nivel de prioridad de la interrupción procedente del *timer* consúltese con el/la profesor/a de prácticas.

```
            INCLUDE  /usr/local/68k/sermones.inc

PI_T EQU    $800000             ;Dirección base PI/T
TCR         EQU     $21         ;Desplazamiento TCR
TIVR EQU    $23                 ;      ""       TIVR
CPR         EQU     $25         ;      ""       CPR
TSR         EQU     $35         ;      ""       TSR
CUENTA      EQU     250000      ;Cuenta inicial para 1 segundo
NUMVEC      EQU     64          ;Numero de vector (arbitrario)

            ORG     $1000

            CLR.B   SEGS               ;Segundos a cero.
            MOVE.L  #TIMRUT,4*NUMVEC   ;Intercepta el vector número 64.
            LEA     PI_T,A0            ;A0 apunta a la base del PI/T.
            MOVE.B  #%10100000,TCR(A0)      ;Interrupción. periódica,
                                       timer parado.
            MOVE.B  #NUMVEC,TIVR(A0)   ;Número de vector a suministrar.
            MOVE.L  #CUENTA,D0         ;Cuenta inicial = 250000, para
            MOVEP.L D0,CPR(A0)         ;un periodo de 1 segundo
                                       ;(8 MHz/32).
            BSET    #0,TCR(A0)         ;Timer en marcha.
ESP15       CMPI.B  #$15,SEGS          ;Han pasado 15 (BCD) segundos?
            BLO.S   ESP15              ;No -> sigue esperando.
            BCLR    #0,TCR(A0)         ;Sí -> se detiene el timer.
            SERMON  FPROG              ;Fin.
```

```
;-------------------------------------------------------------------------
;Rutina de servicio para INTERRUPCION DEL TIMER
;(periódica, una por segundo)
;Función:    Incrementa contador BCD (dirección, SEGS)
;            y visualiza un mensaje.

TIMRUT      MOVEM.L  D6-D7/A0,-(A7)    ;Se usan estos registros.
            ORI      #%00010000,CCR    ;Suma 1 a SEGS (en BCD) poniendo
            CLR.B    D6                ;X a 1 y sumando 0 (en D6). El
            CLR.L    D7                ;resultado queda en D7 (para
            MOVE.B   SEGS,D7           ;cuando se llame a ITOA) y
            ABCD     D6,D7             ;se almacena en SEGS de nuevo.
            MOVE.B   D7,SEGS
            LEA      NUMERO,A0         ;Convierte segundos a ASCII.
            SERMON   ITOA
            LEA      CADENA,A0         ;Visualiza el mensaje.
            SERMON   LNPUTS
            BSET     #0,PI_T+TSR       ;Desactiva interrupción (bit ZDS)
            MOVEM.L  (A7)+,D6-D7/A0    ;Recupera registros y retorna
            RTE                        ;de la rutina de servicio.
;-------------------------------------------------------------------------

            ORG      $2000

    CADENA  DC.B     'Interrupción del timer: '
    NUMERO  DS.B     9
    SEGS    DS.B     1
```

Como práctica:

Introdúzcase en el 68fil el programa del ejemplo anterior y estúdiese *detenidamente* su funcionamiento.