Interrupciones dentro de los Microprocesadores

Tobías García Mejía

3 de Julio del 2020

¿Qué es una Interrupción?

Dentro del ámbito de los microprocesadores, una interrupción es una señal que puede ser enviada desde el mundo exterior o emitida desde dentro del microprocesador que, gracias a un mecanismo del que se hablará más adelante, es capaz de hacer que este detenga el proceso que estaba ejecutando, y en respuesta a dicha señal (la interrupción), ejecute un conjunto de instrucciones predefinidas, denominadas ISR (*Interrupt Service Routine*), para luego reanudar la ejecución del proceso anterior exactamente en las mismas condiciones en las que fue interrumpido.

Tipos de Interrupción

A grandes rasgos existen dos tipos de interrupciones, las internas o también denominadas como interrupciones por software, son aquellas que son emitidas por el mismo microprocesador en sincronía con el reloj de este, es decir, están programadas para generarse cuando se ejecuta una instrucción particular dentro del flujo del programa; su principal propósito es el manejo de excepciones, como por ejemplo el desbordamiento de una variable, una división por 0 o una violación de segmento, aunque también pueden ser programadas para lanzarse cuando una instrucción de interés sea ejecutada.

En contraposición, las interrupciones externas o también denominadas como interrupciones por hardware, se refieren a las que se generan desde fuera del microprocesador, por ejemplo desde un periférico como un teclado o un mouse, o desde un sensor de temperatura o humedad; estas interrupciones son de carácter asincrónico, pues pueden suceder en cualquier momento independientemente del reloj del microprocesador, y su propósito principal es reportar que una condición de interés se ha cumplido en el mundo exterior.

Breve Historia de las Interrupciones

Las interrupciones internas fueron las primeras en ser implementadas, exclusivamente para el manejo de excepciones. La computadora UNIVAC-I (1951) fue la primera en contar con el manejo de excepciones, aunque como se mencionó antes, las interrupciones internas son sincrónicas y la UNIVAC-I sólo contaba con este tipo de interrupciones [1]; la primera computadora en implementar interrupciones asincrónicas fue la NBS DYSEAC (1954), una computadora desarrollada para la US Army Signal Corps, que extendía el concepto de interrupción a las operaciones de entrada y salida convirtiéndose así en la primera en implementar interrupciones externas [2]. Posteriormente, en la computadora UNIVAC 1103A (1956), sucesora de la UNIVAC 1103 (1953), se implementó un sistema de interrupciones cuyo propósito era la recolección de datos en tiempo real de un túnel de viento perteneciente a la NASA [2], convirtiéndola en la primera computadora en utilizar las interrupciones para este fin.

Un año después, alrededor de 1957, el sistema de interrupciones se mejoró aún más con la implementación del vector de interrupciones (concepto que será abarcado más adelante), idea desarrollada en paralelo en EEUU dentro del diseño de las computadoras IBM Stretch (1957) y Lincoln Labs TX-2 (1957), y en Ámsterdam dentro del diseño de la computadora Electrologica X-1 (1958) [1], en donde trabajó el mismo Edsger W. Dijkstra (1930 - 2002), quien realizó su PhD en el manejo de interrupciones [2].

Tres años después, hacía 1960, las interrupciones ya eran una parte fundamental del funcionamiento de la mayoría de computadoras, a excepción de unas cuentas cuyo propósito era buscar una alternativa a las interrupciones y otras que en su lugar utilizaban el método del *polling*[1] [1]. Desde entonces, las interrupciones se continuaron implementando en las grandes y ostentosas computadoras de aquella época, que fueron evolucionando hasta convertirse en los diminutos microprocesadores de hoy día.

[1] Es un método alternativo a las interrupciones externas, aunque mucho más ineficiente, pues consiste en que el propio microprocesador sondee periódicamente los dispositivos de entrada y salida revisando si alguno necesita establecer comunicación con él; además de que si se usa el *polling* para censar variables externas, se corre el riesgo de perder información en caso de que el periodo de sondeo no sea suficientemente pequeño para registrar los eventos de interés.

Implementación de las Interrupciones a Nivel de Hardware

Para el manejo de interrupciones los microprocesadores cuentan con una unidad de interrupciones que se encarga de todo el proceso. Como se mencionó antes, en respuesta a una solicitud de interrupción, denominada técnicamente como una IRQ (*Interrupt Request*), el microprocesador ejecuta la ISR correspondiente, pero para conocer cual ISR corresponde a cada interrupción, se utiliza la tabla de vectores de interrupción, en donde cada índice representa una interrupción y su contenido es la dirección de memoria donde se encuentra almacenada la respectiva ISR. Esta tabla se encuentra ordenada según la prioridad de las interrupciones, aunque el tema de la prioridad será explicado en detalle más adelante.

La ubicación de la tabla comienza en la dirección 0x0000 con la interrupción de RESET, la cual es la de mayor prioridad [3], no obstante, otros autores consideran que a términos prácticos, la tabla comienza en 0x004h, pues la interrupción RESET siempre está predefinida y habilitada para todos los modelos de microcontroladores [4]. El resto de interrupciones varían según el modelo del microcontrolador, por lo cual a la hora de implementar una interrupción, el hardware utilizado sí afecta el proceso, pues como ejemplo, en la plataforma Arduino el modelo Due puede recibir interrupciones externas por todos sus pines digitales, mientras que en el modelo UNO, las interrupciones externas sólo pueden ser recibidas por los pines 2 y 3 [5].

Para controlar cuales interrupciones pueden ser aceptadas por un microprocesador existen registros especiales, el más importante es el bit GIE (*Global Interrupt Enable*) que en caso de encontrarse en 0 desactiva todas las interrupciones a excepción de la de RESET, no obstante, aunque el bit GIE se encuentre en 1, cada interrupción de la tabla posee su propio bit de permiso para ser habilitada [6].

Todo el flujo del proceso de una interrupción se puede describir de la siguiente forma:

1. La señal de la interrupción llega a la unidad de interrupciones en donde una bandera es activada, luego de que en el mundo real se cumpla una condición de interés y nuestro circuito lo detecta, o cuando se genera una interrupción interna desde dentro del microprocesador. En caso de ser una interrupción externa, la bandera se mantiene activada incluso si la señal del exterior se detuvo, esto es para evitar posibles omisiones [4].

2. Al final de cada instrucción el microprocesador consulta a la unidad de interrupciones si se ha generado alguna IRQ [7]; en caso de que el bit GIE se encuentre activo y efectivamente se haya generado una interrupción, en lugar de ejecutar la siguiente instrucción, el microprocesador guarda el estado de registros y banderas actuales, además de la dirección de memoria del contador del programa (PC, *Program Counter* o IP, *Instruction Pointer*) y procede a desactivar el bit GIE para evitar que otra interrupción se genere [8].

3. Posteriormente el microprocesador consulta la tabla de vectores de interrupción en busca de la fuente, cuando es encontrada y su correspondiente bit de permiso está habilitado, se salta a la dirección indicada en la tabla para comenzar la ejecución de la respectiva ISR.

4. Se ejecutan las instrucciones de la ISR, luego se activa de nuevo el bit GIE, además de restaurar los registros, banderas y el contador del programa guardados, para continuar con la ejecución de la siguiente instrucción del proceso que fue interrumpido.

En el ítem tres se menciona que justo antes de comenzar con la ejecución de la ISR, el bit GIE es desactivado para no recibir otra interrupción durante el proceso; en caso de que se requiera procesar otra posible interrupción dentro de la ISR, es necesario habilitar explícitamente el bit GIE por medio de las instrucciones que esta contiene [7].

Un último aspecto para mencionar sobre el funcionamiento de las interrupciones es la prioridad. Cuando dos interrupciones se presentan, la primera en ser atendida es la que primero se encuentre dentro de la tabla de vectores de interrupción, mientras que la segunda es postergada al momento en que se concluye la ejecución de la ISR de la primera, por lo cual el orden en que se colocan las interrupciones dentro de la tabla les asigna una prioridad respecto a las demás; sin embargo, dentro de la gama alta de microcontroladores se pueden encontrar modelos que pueden clasificar las interrupciones como de alta o baja prioridad, permitiendo que la ejecución de la ISR de una interrupción de baja prioridad pueda ser interrumpida por una de alta, mientras que una de alta no puede ser interrumpida por una de baja [9].

Implementación de las Interrupciones a Nivel de Software[2]

Las interrupciones, al ser utilizadas muy comúnmente en los sistemas embebidos, pueden ser implementadas directamente en Assembly, pero también existen alternativas de más alto nivel como C, o en la actualidad incluso se puede usar Python 3 por medio de MicroPython[3], por lo cual la forma de implementar una interrupción dentro de un microcontrolador puede varía dependiendo del lenguaje que se utilice para programarlo, en este caso se tomará la plataforma Arduino como referencia para describir la implementación de una interrupción externa, en particular el modelo UNO, pues es en este último en que está implementado el ejemplo demostrativo que se encuentra con este documento.

En el Arduino UNO las únicas interrupciones externas son INT0 e INT1, las cuales corresponden a los pines 2 y 3 respectivamente. Para asignar una función como la ISR de una de estas dos interrupciones se utiliza la sentencia:

attachInterrupt(num_interrupt, ISR, modo)

Donde *num_interrupt* corresponde al índice dentro de la tabla de vectores de interrupción de la interrupción que activará la ISR, una recomendación es utilizar digitalPinToInterrupt(n_pin), que devuelve el índice dentro de la tabla correspondiente a la interrupción del pin *n_pin*, esto es para facilitar la portabilidad del código a otro modelo de Arduino; *ISR* es el identificador de la función que será ejecutada en respuesta a la IRQ, y finalmente *modo* corresponde a la forma en que la interrupción será disparada, el cual puede ser cualquiera de los siguientes:

- **LOW:** Se activa la interrupción cuando se encuentra un valor de 0 V en el pin designado.
- **CHANGE:** Se activa la interrupción cuando se detecta un cambio en el voltaje o valor lógico del pin.
- **RISING:** Cuando hay un flanco de subida, es decir, se pasa de un estado de LOW a HIGH.
- **FALLING:** Cuando hay un flanco de bajada, es decir, se pasa de un estado de HIGH a LOW.

Adicionalmente para desasociar una función con una interrupción se puede utilizar la sentencia:

detachInterrupt(num_interrupt)

Donde de nuevo *num_interrupt* corresponde al índice de la tabla.

Finalmente, concluimos esta última sección con algunas recomendaciones para implementar una ISR:

1. Durante la ejecución de la ISR, no se actualiza la función **millis**() y **micros**(), por lo cual la función **delay**() no funciona pues utiliza internamente a **millis**().
2. Las ISR deben poder ejecutarse lo más rápido posible, pues interrumpen la ejecución de algún otro proceso, por lo cual no es recomendable utilizar funciones de entrada y salida como **Serial.print()** o **Serial.read()**.
3. Las variables externas a la ISR deben ser declaradas utilizando el modificador **volatile**, esto le indicará al compilador que estas variables pueden modificar su valor en cualquier momento de la ejecución.

En este enlace se puede ver un ejemplo de la implementaciónd de una interrupción en Arduino UNO.

[2] La información de esta última sección fue tomada en su mayoría de [3], [5] y [10].

[3] MicroPython es una implementación del lenguaje de programación Python 3 optimizada para correr en microcontroladores, ofreciendo una forma de utilizar Python 3 para trabajar con operaciones de bajo nivel [11].

Referencias

[1] I. Ahmad. (2011). History of Interrupts, dirección: https://virtualirfan.com/history-of-interrupts (visitado 2020).

[2] M. Smotherman. (2017). Interrupts, dirección: https://people.cs.clemson.edu/~mark/interrupts.html (visitado 2020).

[3] F. Reyes Cortés y J. Cid Monjaraz, *ARDUINO: Aplicaciones en Robótica, Mecatrónica e Ingenierías, Manejo de Interrupciones*, Alfaomega, ed. 2015, ISBN: 9788426729460.

[4] C. Fonseca, "Interrupciones en los Microcontroladores."

[5] Arduino Reference. (2019). attachInterrupt(), dirección: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/ (visitado 2020).

[6] D. Apaza Condori, *Microcontroladores PIC Fundamentos y Aplicaciones un Enfoque Didáctico*. 2017.

[7] Materia: Arquitectura, Sistemas Operativos y Redes de Computadoras, "Arquitectura de Computadoras - Interrupciones," Facultad de Ingeniería - Universidad de la República - Uruguay.

[8] M. F. Campos Cerda, R. Castañeda Pérez y A. C. Contreras Torres, "Implementación de un Sistema de Desarrollo Utilizando los Microcontroladores PIC Microchip Technology," Universidad de Guadalajara - Centro Universitario de Ciencias Exactas e Ingeniería, cap. 5.

[9] R. J. Dominguez Arellano, "El Microcontrolador 8051," Departamenteo de Ingeniería Electrónica - Instituto Tecnologíco del Mar, Mazatlán.

[10] L. Llamas. (2016). Qué son y cómo usar interrupciones en Arduino, dirección: https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/ (visitado 2020).

[11] MicroPython. (2018). MicroPython - Python for Microcontrollers, dirección: https://micropython.org/ (visitado 2020).