Trabajar con Programas C y Ensamblador en RIPES

Autor: Bruno Burgos Kosmalski

Índice:

Introducción:

RARS es un simulador que principalmente se usa para trabajar con programas simples escritos en lenguaje ensamblador del estándar RISC-V [1]. Sin embargo, este programa está adaptado para poder trabajar con excepciones y entrada salida por interrupciones, donde tendremos una serie de herramientas y registros de control para llevar a cabo esta tarea.

En este documento se van a detallar los distintos pasos necesarios para poder trabajar con excepciones e interrupciones en RARS. Para esto, se ha trabajado principalmente con las características RV32IN [2], donde se entiende que el propio simulador da soporte a otras extensiones, teniendo listadas las instrucciones en la propia documentación interna del simulador.

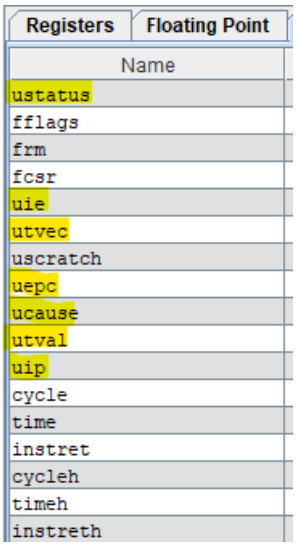
Registros de Control e Instrucciones útiles:

Para trabajar con excepciones e interrupciones hay que conocer los distintos registros de control del simulador y, como se puede trabajar con ellos. Es por esto, que este apartado se va a dedicar a algunos de los registros de control más importantes y sus funciones, junto a las instrucciones dedicadas para trabajar con los mismos.

Empezando por los registros de control, en RARS tenemos un listado bastante completo. Como se ve en la imagen a continuación cada uno de ellos, a parte del nombre, tiene asignado un identificador numérico que también podremos usar para referenciarlos.

Registers | Floating Point | Control and Status

Name	Number	Value
ustatus	0	0x00000000
fflags	1	0x00000000
frm	2	0x00000000
fcsr	3	0x00000000
uie	4	0x00000000
utvec	5	0x00000000
uscratch	64	0x00000000
uepc	65	0x00000000
ucause	66	0x00000000
utval	67	0x00000000
uip	68	0x00000000
cycle	3072	0x00000000
time	3073	0x00000000
instret	3074	0x00000000
cycleh	3200	0x00000000
timeh	3201	0x00000000
instreth	3202	0x00000000



Registros:

- ustatus: Principal registro de control de interrupciones y excepciones. Si se quieren habilitar las interrupciones o las excepciones se tiene que cambiar este registro, también, cuando se genera una interrupción este cambia automáticamente para inhibir las interrupciones y excepciones.
- uie: Registro secundario para manejo de interrupciones, a través de este registro puedes inhibir o permitir distintos tipos de interrupciones, es decir, puedes especificar qué tipo de interrupción se puede generar.
- utvec: Registro en el que se almacena la dirección de comienzo del handler, que en este caso es único tanto para interrupciones como excepciones, dirección a la que se saltará automáticamente al generarse una excepción o interrupción.

- uepc: Cuando se genera una excepción o interrupción se almacena en este registro el valor de retorno, es decir, la dirección de la instrucción que se estaba ejecutando cuando se generó la interrupción o excepción.
- ucause: En este registro se almacena el tipo de interrupción o excepción que se ha generado, donde cada una tiene un código específico que las identifica.
- utval: En este registro se almacena un valor adicional a la interrupción que se ha generado, y no necesariamente condicionado con el tipo de interrupción.
- uip: Muestra si hay alguna interrupción pendiente y de que tipo es esta, usando el identificador de la misma.

Una vez conocidos los registros de control, necesitaremos un medio por el cual manipular sus valores asociados. Aquí es donde entran en juego las instrucciones específicas del estándar. Aun cuando ya están ya están descritas en el manual oficial [3], se dará a continuación un breve resumen de funcionalidad.

Instrucciones:

Las instrucciones csr son instrucciones especializadas para la manipulación de los registros de control, puesto que garantizan la atomicidad en su ejecución. Estas instrucciones siguen el formato: csr... rd, fcsr, rs1 o para las variantes con datos inmediatos, csr... rd, fcsr, imm. Véase más información en el manual de instrucciones del estándar [3].

- csrrc (Read/Clear): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 0 todo bit que se encuentre activo en el registro rs1.
- csrrci (Read/Clear Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 0 los bits indicados por el dato inmediato (imm).
- csrrs (Read/Set): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 1 todo bit que se encuentre activo en el registro rs1.
- csrrci (Read/Clear Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 1 los bits indicados por el dato inmediato (imm).
- csrrw (Read/Write): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control escribe el valor actual del registro rs1.
- csrrwi (Read/Write Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control escribe el valor definido por el dato inmediato (imm).
- uret: Instrucción que nos permite salir de manera automática de la rutina de tratamiento de interrupciones, es decir, reactiva las interrupciones y excepciones y retorna a la instrucción que se estaba ejecutando cuando se produce la interrupción o excepción.

Como ejemplo sencillo, si queremos poner el valor 1 en el registro de control uie, podemos hacerlo con la instrucción csrrwi x0, uie, 1. De todas formas, en los siguientes apartados se mostrarán ejemplos que muestran el uso práctico de estos registros.

Excepciones:

Las excepciones son fundamentales cuando estamos tratando con la detección de errores, y aun cuando muchos simuladores no las implementan por simplicidad, RARS implementa 7 tipos distintos de excepciones: instruction address misaligned (0), instruction access fault (1), illegal instruction (2), load address misaligned (4), load access fault (5), store address misaligned (6), store access fault (7), and environment call (8)

Adicionalmente, trabajar con excepciones en RARS es relativamente sencillo, solo hay que configurar el valor de los registros de control utvec, para la dirección del handler, y ustatus para que se puedan activar las excepciones. La dificultad radica en la implementación del handler, lo que se tiene que hacer completamente por software.

Para poner un ejemplo sencillo, si queremos un programa cuya manera de responder ante las excepciones sea saltando a la siguiente línea lo podemos hacer de la siguiente forma:

```
1  .globl __start
2  .text
3  __start:
4  la t0, handler # escribimos la direccion del handler en el registro t0
5  csrrw zero, utvec, t0  # pasamos la direccion del handler al registro de vectorización utvec
6  csrrwi zero, ustatus, 1 # activamos las excepciones
7  lw zero, 0 # acceso ilegal de memoria
8  addi t1, x0, 1
9  jal x0, exit
10 handler:
11 csrrw t0, uepc, zero
12 addi t0, t0, 4
13 csrrw zero, uepc, t0
14 uret
15 exit:
16 addi zero, zero, 0
```

Este programa comienza inicializando los registros de control ustatus y utvec, donde el bit 1 del registro ustatus es el que habilita las excepciones e interrupciones, y el handler es donde se encuentra la rutina de tratamiento. Esta primera secuencia será común para la mayoría de los programas que trabajen con excepciones o interrupciones.

```
4  la t0, handler # escribimos la direccion del handler en el re
5  csrrw zero, utvec, t0  # pasamos la direccion del handler al
6  csrrwi zero, ustatus, 1 # activamos las excepciones
```

Por otro lado, como se ha decidido implementar un programa que continúe ejecutando en la siguiente instrucción, para la implementación del handler se recoge el valor del registro uepc, que, en este caso, contiene la dirección de la instrucción que generó la excepción, le suma 4 (tamaño de palabra) y retorna con la instrucción uret. Al valor del registro uepc se le suma 4 puesto que si se retornara tal cual del handler nos encontraríamos en un bucle infinito donde la instrucción que generó la excepción volvería a hacerlo. Como veremos en el siguiente apartado, para las interrupciones no se pasa a la siguiente instrucción, sino que se retorna a la que se ha visto interrumpida.

```
10 handler:
11 csrrw t0, uepc, zero
12 addi t0, t0, 4
13 csrrw zero, uepc, t0
14 uret
```

Este ha sido un ejemplo sencillo, sin embargo, se puede hacer un handler tan complejo como se necesite para cada caso, inclusive se puede diferenciar entre los distintos tipos

de excepciones a través del su código. Para este caso, cuando se salta al handler, el registro ucause toma el valor 5 puesto que se hace un acceso ilegal en la línea 7, en caso de haberse generado otro tipo de excepción el código hubiese sido distinto.

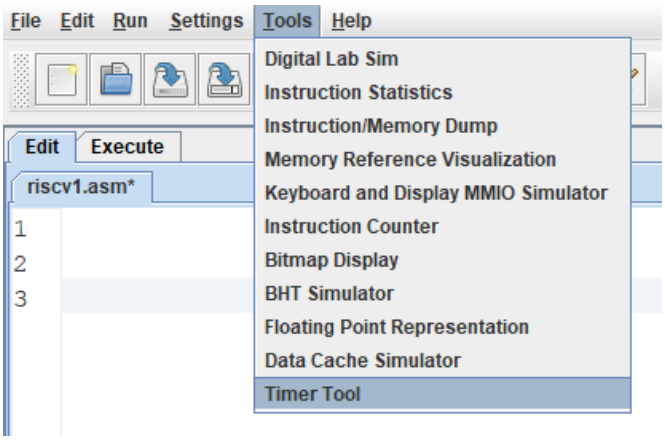
Entrada/Salida por Interrupciones:

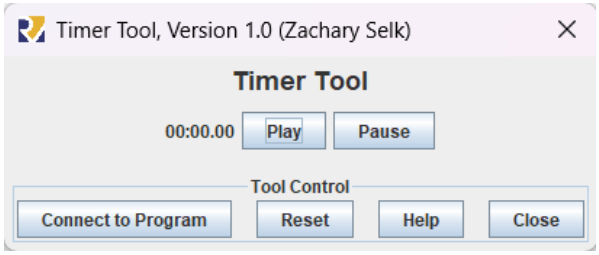
Siguiendo con el apartado anterior, está claro que RARS tiene implementado un sistema de interrupciones, entonces, si lo que queremos es trabajar con las interrupciones externas del simulador, es decir, la entrada salida mediante interrupciones; RARS nos ofrece una serie de herramientas ya configuradas para esta tarea.

Dentro del simulador tenemos un total de 4 herramientas ya implementadas, de entre las cuales para entrada salida mediante interrupciones podemos usar solo 3, como son: un teclado y un display, un timer, y un teclado hexadecimal. En este apartado nos vamos a centrar solo en el timer y en el teclado y display, que ya son suficientes para una visión bastante completa del entorno. Nótese que los ejemplos también están simplificados suponiendo siempre o la mejor de las situaciones o la segunda mejor.

El tratamiento de las interrupciones es muy parecido a como se hacía el de las excepciones, la única diferencia es que ahora, también se tienen que inicializar otros registros de control, como es el uie, o los propios registros de control de las herramientas.

Comenzando con el timer: este dispositivo es muy sencillo en el sentido de que solo tiene una tarea, que es interrumpir una vez pasado el umbral de tiempo establecido.





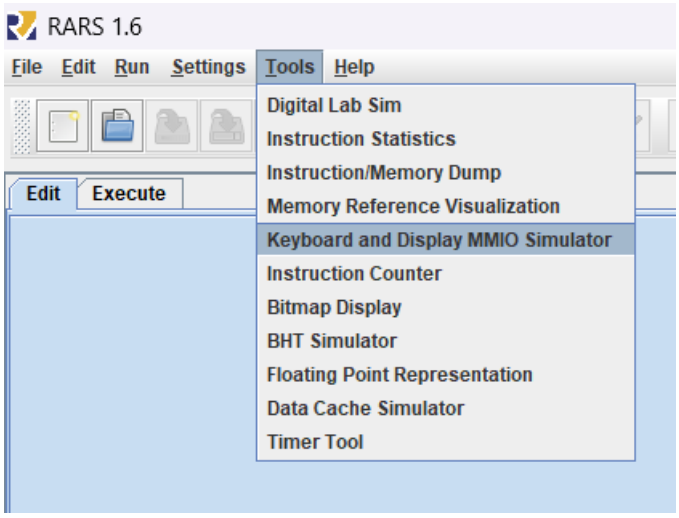
Las interacciones entre los programas y los dispositivos se hacen mayormente mediante MMIO, con una serie de direcciones reservadas para los registros de control internos del dispositivo u otros. Para el caso del timer, hay dos palabras reservadas donde se almacena el valor en milisegundos del umbral que hay que superar para que se active la interrupción externa. Es por esto que cuando inicie el programa una de las tareas que tendremos es la de almacenar en estas direcciones el valor que busquemos. Nótese que no hay manera de reiniciar el proceso del timer de forma automática, una vez que ha interrumpido, para volver a interrumpir se necesitará reiniciar de forma manual la herramienta.

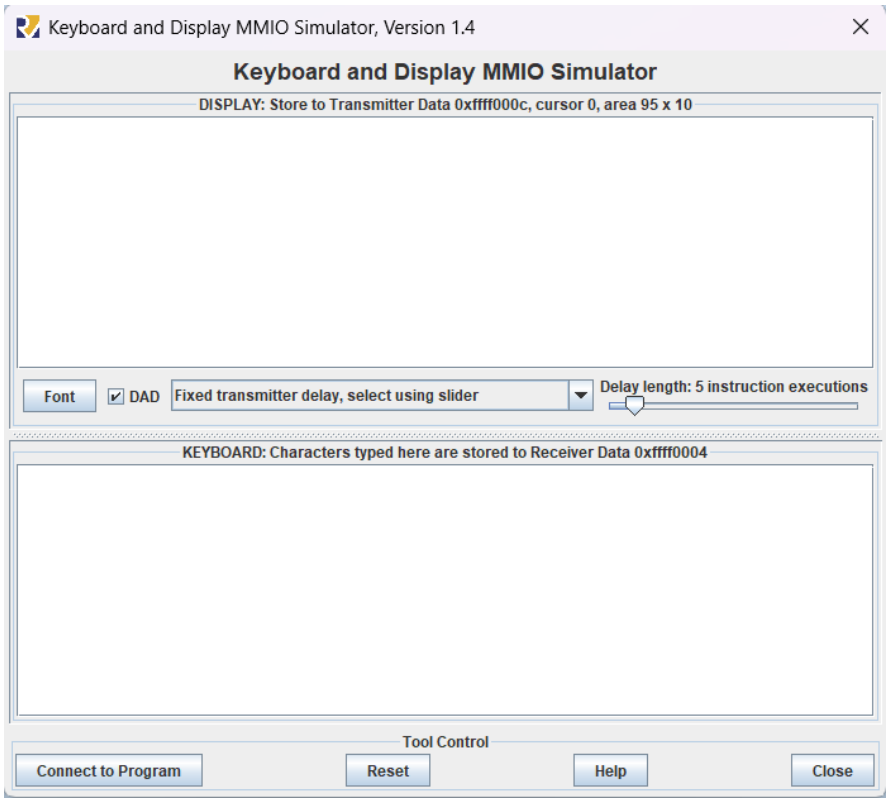
```
1  .globl _start
2  .data
3  texto: .asciz "Texto de prueba"
4  .text
5  _start:
6  la t0,handler
7  csrrw zero, utvec, t0  # set utvec (5) to the handlers address
8  csrrwi zero, uie, 16   # set the timer interrupt enable bit in uie (4)
9  li t0, 0xFFFF0020
10 addi t1, zero, 1000    # time when the timer will interrupt (1000 ms)
11 sw t1, 0(t0)
12 li t0, 0xFFFF0024
13 sw zero, 0(t0)
14 csrrsi zero, ustatus, 1 # set interrupt enable bit in ustatus (0)
15 buc:
16 addi x0, x0, 0
17 jal x0, buc
18
19 buc2:
20 addi x0, x0, 0
21 jal x0, buc2
22
23 handler:
24 addi x0, x0, 0
25 jal x0, buc2
```

En este ejemplo, se busca que el programa se mantenga en un bucle infinito y cuando el timer interrumpa el handler salte a otro bucle infinito distinto al primero. En este caso se ha entendido que el timer solo interrumpe una vez, es por esto que no se reestablecen los registros de control. Para la configuración inicial de este dispositivo, tenemos que activar el bit 4 del registro de control uie y adicionalmente, tendremos que almacenar en las dos palabras reservadas el valor del umbral en milisegundos en el que esperamos que se genere la interrupción: 32 bits menos significativos dirección 0xFFFF0020, 32 bits más significativos dirección 0xFFFF0024.

```
8  csrrwi zero, uie, 16   # set the timer interrupt enable bit in uie (4)
9  li t0, 0xFFFF0020
10 addi t1, zero, 1000    # time when the timer will interrupt (1000 ms)
11 sw t1, 0(t0)
12 li t0, 0xFFFF0024
13 sw zero, 0(t0)
```

Aumentando un poco el nivel de dificultad con respecto al ejemplo anterior, tenemos la herramienta del teclado y display, que nos ofrece un apartado de recepción y otro de transmisión.





En un principio, la manera de trabajar con esta herramienta es muy parecida a como se hace con el timer: se tiene tanto para el teclado como para el display una dirección reservada que actúa como registro de control (0xFFFF0000 para recepción y 0xFFFF0008 para transmisión) donde los dos únicos bits que nos interesan son los menos significativos. El bit 0 de este registro de control es conocido como ready bit, y es el que nos indica si la recepción o transmisión están disponibles; por otra parte, el bit 1 es el que permite el trabajo mediante interrupciones, si lo activamos se podrán generar interrupciones cuando el ready bit se active. Y al igual que con los registros de control, tendremos otras dos direcciones reservadas para el envío y recibo de los caracteres (0xFFFF0004 para recepción y 0xFFFF000C para transmisión), si por ejemplo queremos que aparezca la “a” en el display tendremos que almacenar en la dirección 0xFFFF000C el 61 decimal (“a” en ascii), y esperar a que termine la operación.

Como primer ejemplo, hemos creado un programa que solo solicita interrupciones a través del teclado.

```
1  .globl _start
2  .text
3  _start:
4  la t0, handler
5  csrrw zero, utvec, t0 # set utvec (5) to the handlers address
6  addi t0, x0, 256
7  csrrw zero, uie, t0 # set external interrupt bit uie bit(8)
8  li t0, 0xFFFF0004   # Reciber Data register
9  li t1, 0XFFFF0000   # Reciber Control register
10 lw t2, 0(t1)
11 ori t2, t2, 2
12 sw t2, 0(t1)        # activa el bit de interrupciones del registro de control del teclado
13 csrrwi zero, ustatus, 1
14
15 buc: # bucle infinito que no hace nada
16 addi x0, x0, 0
17 jal x0, buc
18
19 handler:
20 lw t2, 0(t0)       # se recibe el caracter
21 li t0, 0xFFFF0008  # Transmiter Control Register
22 li t1, 0xFFFF000C  # Transmiter Data Register
23 sw t2, 0(t1)       # se envia el caracter recibido
24 li t0, 0xFFFF0004  # Reciber Data register
25 li t1, 0XFFFF0000  # Reciber Control register
26 uret
```

Primero inicializamos los registros de control necesarios para trabajar con interrupciones, aunque sea solo para la recepción: activamos el bit 8 del registro de control uie, que es el que permite las interrupciones de esta herramienta, y luego activamos también el bit de interrupciones del registro de control de la herramienta para las recepciones (bit 1 dirección 0xFFFF0000).

```
4  la t0, handler
5  csrrw zero, utvec, t0 # set utvec (5) to the handlers addre
6  addi t0, x0, 256
7  csrrw zero, uie, t0 # set external interrupt bit uie bit(8)
8  li t0, 0xFFFF0004   # Reciber Data register
9  li t1, 0XFFFF0000   # Reciber Control register
10 lw t2, 0(t1)
11 ori t2, t2, 2
12 sw t2, 0(t1)        # activa el bit de interrupciones del r
13 csrrwi zero, ustatus, 1
```

Una vez que hecho esto entramos en un bucle infinito hasta que se active una interrupción, para ese momento no es necesario comprobar el ready bit del registro de control de recepción puesto que sabremos que si se ha interrumpido es porque está activo. Cuando leemos el carácter del registro de datos de recepción (dirección 0xFFFF0004) automáticamente se actualizará el registro de control del mismo, dándose la misma situación para la transmisión.

Una vez obtenido el carácter lo enviamos para la transmisión, en este ejemplo por simplicidad no se comprueba si la transmisión está disponible o no, sino que simplemente se asume, después de esto se retorna de la rutina de tratamiento.

```
19 handler:
20 lw t2, 0(t0)       # se recibe el caracter
21 li t0, 0xFFFF0008  # Transmiter Control Register
22 li t1, 0xFFFF000C  # Transmiter Data Register
23 sw t2, 0(t1)       # se envia el caracter recibido
24 li t0, 0xFFFF0004  # Reciber Data register
25 li t1, 0XFFFF0000  # Reciber Control register
26 uret
```

Como problemáticas o inconvenientes a destacar, tenemos: ni la recepción ni la transmisión pueden almacenar más de un carácter al mismo tiempo, por ejemplo, si se pulsa otra tecla del teclado antes de haber leído la anterior esta se pierde; las interrupciones solo se generan cuando cambia el estado del dispositivo, es decir cuando cambia el bit de ready, para las recepciones no es algo tan problemático, sin embargo, para la transmisión aún cuando se cambia el registro de control no se interrumpe, puesto que aunque esté disponible para interrumpir, como no cambia el estado no comprueba si puede interrumpir.

Para el último ejemplo se van a jugar con la recepción y transmisión por interrupciones, teniendo en cuenta la limitación anterior, en este caso se ha diseñado el programa para que compruebe si el bit de ready está activo en la transmisión, en caso de estarlo almacena el carácter directamente, y solo si no está disponible activa las interrupciones. También, cuando se recibe una interrupción de transmisión se desactivan las interrupciones para evitar que esté constantemente interrumpiendo. En un programa más completo se usaría un almacenamiento de buffers para evitar este problema, sin embargo, en este caso por simplicidad no se ha implementado.

```
.globl _start
.text
_start:
la t0, handler
csrrw zero, utvec, t0 # set utvec (5) to the handlers address
addi t0, x0, 256
csrrw zero, uie, t0   # set key_dipl interrupt bit (8) in uie
li s0, 0xFFFF0000
lw t0, 0(s0)
ori t0, t0, 2
sw t0, 0(s0)
csrrsi zero, ustatus, 1 # set interrupt enable bit in ustatus (0)

buc: # bucle infinito
addi x0, x0, 0
jal x0, buc

handler:
# recoger el valor de interrupción
csrrw t0, 67, zero # utval
csrrw zero, 67, t0
andi t0, t0, 64
beq t0, x0, transmision
# interrupcion por recepcion
lw a0, 4(s0)
lw t0, 8(s0)
andi t0, t0, 1
beq t0, x0, activa
sw a0, 12(s0)
jal x0, fin_RTI
activa:
lw t0, 8(s0)
ori t0, t0, 2
sw t0, 8(s0)
jal x0, fin_RTI
# interrupcion por transmision
transmision:
sw a0, 12(s0)
lw t0, 8(s0)
andi t0, t0, 1
sw t0, 8(s0)
# fin del la subrutina
fin_RTI:
uret
```

Como se puede ver en el código, en este caso la complejidad del programa radica en la rutina de tratamiento de interrupción, que tiene que diferenciar entre los distintos casos posibles, lo que es gracias al registro de control utval, identificador 67, que dependiendo de si se ha generado una interrupción de recepción o de transmisión toma un valor

distinto (0x40 para la transmisión y 0x80 para la recepción). En este código para las direcciones de los registros de control de los dispositivos, se han descrito a partir de la dirección 0xFFFF0000, calculando su desplazamiento, añadiendo esta dirección en el registro s0 por simplicidad.

Resumen:

RARS es un simulador que nos permite trabajar con excepciones junto a entrada salida por interrupciones. Un recurso del que simuladores como RIPES carecen. Para trabajar con las interrupciones en RARS hay que conocer y manejar los registros de control de los que dispone el simulador. Aunque es verdad que hay partes que son comunes independientemente del tipo de interrupción o de excepción.

Para activar de manera general las interrupciones y excepciones, tendremos que activar el bit 0 del registro ustatus, esto, una vez que hayamos cargado la dirección de comienzo de la rutina de tratamiento en el registro de vectorización utvec. Lo que se puede hacer con las siguientes líneas de código:

```
4  la t0, handler # escribimos la direccion del handler en el re
5  csrrw zero, utvec, t0  # pasamos la direccion del handler al
6  csrrwi zero, ustatus, 1 # activamos las excepciones
```

En este caso suponemos “handler” como la etiqueta asociada al comienzo de la rutina de tratamiento de interrupciones.

Por otra parte, cuando se genera una interrupción, para identificar la misma podremos usar los registros de control utval y ucause, donde ucause nos da el código asociado al tipo de interrupción que se ha generado, y utval contendrá el valor asociado a la interrupción misma.

Para volver de la rutina de tratamiento de interrupciones podremos usar la instrucción “uret”, aunque siempre entendiendo que se vuelve a la instrucción que ha sido interrumpida o a aquella que ha generado la excepción, y no a la siguiente.

Y ya si se quiere trabajar con entrada salida por interrupciones, habrá que tener en cuenta cuales son las características del dispositivo o herramienta que vamos a usar: si tiene registros de control internos y como se configuran, cual es el tipo de interrupción que genera y que bit del registro de control uie se tiene que activar para que se puedan generar interrupciones, etc. En el ejemplo del teclado y display, que se ha visto en el apartado de interrupciones, veíamos como era necesario activar el bit 8 del registro de control uie, y mirar las direcciones asociadas a los registros de control internos del dispositivo, sin embargo, para el timer, se tenía que activar el bit 4 del registro de control uie y especificar el umbral sobre el que queríamos que la herramienta interrumpiese.

Referencias:

[1] https://riscv.org

[2] https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[3] https://drive.google.com/file/d/1uviu1nH-tScFfgrovvFCrj7Omv8tFtkp/view