```
/*Programa3SRC
 * main.c
 *
 *  Created on: Feb 29, 2016
 *      Author: pedro
 */


#include "leon3_uart.h"
#include "leon3_bprint.h"
#include "leon3_traps.h"
#include "leon3_irqs.h"

void hw_irq_vector_0x11_handler(void){
	leon3_print_string("handler hw irq vector 0x11\n");
}

void trap_division_0_handler(void){
	leon3_print_string("error, division por cero\n");
}

//main normal
int main(){
	//Instalar como manerjador del trap 0x83 la rutina
	//que habilita las interrupciones
	leon3_install_handler(0x83,leon3_trap_handler_enable_irqs);

	//instalar el manejador del trap que 0x83 la rutina
	//que desabilita las interrupciones
	leon3_install_handler(0x84,leon3_trap_handler_disable_irqs);

	//instalar la funcion hw_irq_vector_0x11_handler como
	//manejador de la interrupcion de nivel 1
	leon3_install_handler(0x11,	hw_irq_vector_0x11_handler);

	//Habilitar las interrupciones
	leon3_sys_call_enable_irqs();

	//Apartado 4: Repetir la ejecución poniendo un breakpoint en la primera instrucción de la
	//función leon3_trap_handler_enable_irqs definida en leon3_irqs_asm.S y
	//otro en la llamada a la función leon3_sys_call_enable_irqs que se
	//encuentra en el main, y que se ha definido como wrapper del trap 3 (vector
	//0x83). Ejecutar paso a paso para comprobar cual es el comportamiento. ¿A qué
	//función salta tras la instrucción en ensamblador ta ? salta a la llamada/funcion leon3_trap_handler_enable_irqs

	//Apartado 6:Hacer lo mismo pero llamando a leon3_sys_call_disable_irqs() en vez de
	//a leon3_sys_call_enable_irqs() y comprobando que tampoco se genera ningún mensaje.
	//leon3_sys_call_disable_irqs();	//tampoco genera mensaje

	//desenmascarar las interrupcoones de nivel 1 (correspondientes
	//al vector 0x11)
	leon3_unmask_irq(1);

	//Apartado 5: Repetir la ejecución enmascarando la interrupción (usa leon3_mask_irq ) antes
	//de leon3_sparc_force_irq(1) y comprobar que no se genera ningún mensaje por pantalla.
	//leon3_mask_irq(1);	//salta el mensaje no lo muestra

	//fuerza la interrupcion
	leon3_sparc_force_irq(1);

	return 0;
}
```

```
//Apartado 7:Ejecutar el siguiente código. ¿Qué ocurre? Sabiendo que cuando se produce una
//división por 0 la rutina que lo gestiona llama al trap 0x82, ¿cómo utilizarías la
//función leon3_install_handler para conseguir que el programa no se cuelgue
//y en su lugar imprima un mensaje que diga “error, división por cero”?
/*int main(){
	uint8_t i;
	uint8_t j;

	//Instalamos la funcion que detecta divisiones entre 0
	leon3_install_handler(0x82,trap_division_0_handler);

	for(i=10; i>0; i--)
		j=j/(i-9);

	return 0;
}*/

/*
 * leon3_uart.c
 *
 *  Created on: Feb 5, 2016
 *      Author: pedro
 */

//archivo en el que se encuentra la funcion especicicada usada en main

//Declaraciones del uso de lib y demas
#include "leon3_uart.h"

//especificamos una mascara con la que compararemos para verificar si esta en un estado concreto un valor  de status
#define LEON3_UART_TFF (0x200) //0x200en exa es 512 en decimal que en binario es 0..1000000000 que es el bit que queremos obtener

//Declaracion del registro y su estructura
struct UART_regs{
	/** \brief UART Data Register */
	volatile uint32_t Data; // 0x80000100
	/** \brief UART Status Register */
	volatile uint32_t Status; // 0x80000104
	/** \brief UART Control Register */
	volatile uint32_t Ctrl; // 0x80000108
	/** \brief UART Scaler Register */
	volatile uint32_t Scaler; // 0x8000010C
};

//Definicion del registro de la estructura y lo ubucamos en 0x80000100
struct	UART_regs * pLEON3_UART_REGS = 0x80000100;

//funcion que recibe un caracter y espera hasta que le permitan escribir o venza el tiempo
int8_t leon3_putchar(char c){
	uint32_t write_timeout=0;

	//esperamos por permiso mientras tengamos tiempo
	while(((LEON3_UART_TFF & pLEON3_UART_REGS->Status)==0x200) && (write_timeout < 0xAAAAA)){
		write_timeout++;
	}

	//Si no hemos llegado al limite de tiempo escribir en data
	if(write_timeout < 0xAAAAA){
		pLEON3_UART_REGS->Data = (uint32_t)c;		//pasamos el valor a el registo en la posicion data
	}
```

```
	return (write_timeout == 0xAAAAA);		//salida en funcion de lo que haya pasado
}

/*
 * leon3_irqs_asm.S
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */

#include "leon3_asm.h"

.section ".text"

//leon3_trap_handler_disable_irqs rutina de atención a un trap que permite deshabilitar todas las interrupciones, independientemente de
//cómo esté configurada su máscara en el registro IMASK.
    .globl leon3_trap_handler_disable_irqs
leon3_trap_handler_disable_irqs:
    mov %l2, %l1
    add %l2, 4, %l2

    or %l0, 0x0f00, %l4    ! set PIL=15
    mov %l4, %psr
    nop;nop;nop
    jmp %l1
    rett %l2

//leon3_trap_handler_enable_irqs rutina de atención a un trap que permite habilitar todas las interrupciones que no estén enmascaradas en
//el registro IMASK .
    .globl leon3_trap_handler_enable_irqs
leon3_trap_handler_enable_irqs:
    mov %l2, %l1
    add %l2, 4, %l2

    andn %l0, 0xf00, %l4
    mov %l4, %psr
    nop;nop;nop
    jmp %l1
    rett %l2

//leon3_sys_call_enable_irqs llamada al sistema, efectuada a través de un TRAP, que permite llamar a la rutina de atención
//leon3_trap_handler_enable_irqs.
    .globl leon3_sys_call_enable_irqs
leon3_sys_call_enable_irqs:
    ta LEON3_SPARC_ENABLE_IRQ_TRAPNUM
    retl
    nop

//leon3_sys_call_disable_irqs(void) llamada al sistema, efectuada a través de un TRAP, que permite llamar a la rutina de atención
//leon3_trap_handler_disable_irqs.
    .globl leon3_sys_call_disable_irqs
leon3_sys_call_disable_irqs:
	ta LEON3_SPARC_DISABLE_IRQ_TRAPNUM
    retl
    nop

    .globl leon3_sparc_isr
leon3_sparc_isr:
```

```
    /* Test for window overflow */

    rd %wim, %l4
    srl %l4, %l0, %l6
    cmp %l6, 1
    bne dont_do_the_window  /* no? then skip all this stuff */
    nop

    /* Perform window overflow  */

    mov %g5,%l5
    srl %l4,1,%l6
    sll %l4,7,%g5
    or  %g5,%l6,%g5
    and %g5,0xff,%g5

    save
    mov %g5,%wim
    nop;nop;nop

    std %l0, [%sp + CPU_STACK_FRAME_L0_OFFSET]
    std %l2, [%sp + CPU_STACK_FRAME_L2_OFFSET]
    std %l4, [%sp + CPU_STACK_FRAME_L4_OFFSET]
    std %l6, [%sp + CPU_STACK_FRAME_L6_OFFSET]
    std %i0, [%sp + CPU_STACK_FRAME_I0_OFFSET]
    std %i2, [%sp + CPU_STACK_FRAME_I2_OFFSET]
    std %i4, [%sp + CPU_STACK_FRAME_I4_OFFSET]
    std %i6, [%sp + CPU_STACK_FRAME_I6_FP_OFFSET]
    restore

    mov %l5,%g5

dont_do_the_window:
    sub %fp, 0x60, %sp
    /* save global registers */
    st  %g1, [%sp + 0x40]
    rd  %y, %g1
    st  %g1, [%sp + 0x44]
    std %g2, [%sp + 0x48]
    std %g4, [%sp + 0x50]
    std %g6, [%sp + 0x58]

    or %l0, 0x0f00, %l4    ! set PIL=15
    mov %l4, %psr
    nop;nop;nop
    or %l4, PSR_ET, %l4    ! enable traps
    mov %l4, %psr
    nop;nop;nop

    set leon3_sparc_irqhandler_entry, %l4
    call %l4
    mov %l3, %o0

    mov %l0, %psr
    nop;nop;nop

    ld  [%sp + 0x44], %g1
    wr  %g1, %y
    ld  [%sp + 0x40], %g1
    ldd [%sp + 0x48], %g2
    ldd [%sp + 0x50], %g4
    ldd [%sp + 0x58], %g6
```

```
   jmp %l1
   rett %l2

/*
 * leon_irqs.c
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */
#include "leon3_types.h"  //Incluimos para poder utilizar las llamadas a las funciones

uint32_t * LEON3_ICLEAR =  (uint32_t *) (0x80000000 + 0x20c);    //registro iclear ubicado en 0x8000020c
uint32_t * LEON3_IMASK =  (uint32_t *) (0x80000000 + 0x240);     //registro imask ubicado en 0x80000240
uint32_t * LEON3_IFORCE =  (uint32_t *) (0x80000000 + 0x208);    //registro iforce ubicado en 0x80000208

//leon3_force_irq(int32_t irq_level) permite forzar el disparo de uno de los 15 niveles de interrupción externa poniendo a 1 en el
registro
//IFORCE el bit correspondiente al nivel (este registro está ubicado en la dirección 0x80000208). El número de nivel debe
suministrarse mediante
//el parámetro irq_level
uint8_t leon3_sparc_force_irq (int32_t irq_level) {
        uint8_t error=0;
        if(irq_level <16){ //en caso de ser una interpcion valida de 0 a 15
                *LEON3_IFORCE=(*LEON3_IFORCE | (0x1<<(irq_level)));          //<< significa desplazamiento
                //*LEON3_IFORCE=(1<<irq_level);
                // COMPLETAR Poniendo a 1 SOLO el bit correspondiente al irq_level de LEON3_IFORCE
                // irq_level=0 corresponde al bit de menor peso
                // mientras que irq_level=15 corresponde al de mayor peso.
        }else
                error=1;
        return error;
}

//leon3_unmask_irq(int32_t irq_level) permite desenmascarar uno de los 15 niveles de interrupción externa poniendo a 1 en el
registro
//registro IMASK el bit correspondiente al nivel. El número de nivel debe suministrarse mediante el parámetro irq_level
uint8_t leon3_unmask_irq (int32_t irq_level){
        uint8_t error=0;
        if(irq_level<16){  //en caso de ser una interpcion valida de 0 a 15
                *LEON3_ICLEAR = (1 << irq_level);         // clear any pending irq of that level
                *LEON3_IMASK=(*LEON3_IMASK | (0x1<<(irq_level)));  //<< significa desplazamiento
                //*LEON3_IMASK=(1<<irq_level);
                // COMPLETAR Poniendo a 1 SOLO el bit correspondiente al irq_level de LEON3_IMASK
                // irq_level=0 corresponde al bit de menor peso
                // mientras que irq_level=15 corresponde al de mayor peso.
        }else
                error=1;
        return error;
}

//leon3_mask_irq(int32_t irq_level) permite enmascarar uno de los 15 niveles de interrupción externa poniendo a 0 en el
registro registro
//IMASK el bit correspondiente al nivel (este registro está ubicado en la dirección 0x80000240). El número de nivel debe
suministrarse mediante
//el parámetro irq_level.
uint8_t leon3_mask_irq (int32_t irq_level) {
        uint8_t error=0;
        if(irq_level <16){ //en caso de ser una interpcion valida de 0 a 15
                *LEON3_IMASK=~((~*LEON3_IMASK) | (0x1<<(irq_level))); //<< significa desplazamiento
                //*LEON3_IMASK=(0<<irq_level);
                // COMPLETAR Poniendo a 0 SOLO el bit correspondiente al irq_level de LEON3_IMASK
```

```
                // irq_level=0 corresponde al bit de menor peso
                // mientras que irq_level=15 corresponde al de mayor peso.
        }else
                error=1;
        return error;
}

/*
 * leon3_bprint.c
 *
 *  Created on: Feb 12, 2016
 *      Author: pedro
 */

#include "leon3_bprint.h"
#include "leon3_uart.h"

int8_t leon3_print_string(char* str){
        //variables aux
        int cont=0;

        while(str[cont]!='\0'){      //recorre el arry de char hasta encontrar el /0
                leon3_putchar(str[cont]);
                cont++;
        }
        return 0;
}

int8_t leon3_print_uint8(uint8_t i){
        int aux=0;

        if (i<=9){              //caso de solo unidades
                aux=i;                                  //unidades
                leon3_putchar('0'+aux);
        }else if(i>9 && i<=99){     //caso de decenas y unidades
                aux=((i/10)%10); //decenas
                leon3_putchar('0'+aux);
                aux=i%10;                               //unidades
                leon3_putchar('0'+aux);
        }else{                  //caso de centenas decenas y unidades
                aux=((i/10)/10);   //centenas
                leon3_putchar('0'+aux);
                aux=((i/10)%10); //decenas
                leon3_putchar('0'+aux);
                aux=i%10;                               //unidades
                leon3_putchar('0'+aux);
        }
        leon3_putchar('\n');        //salto de linea para que sea mas legible
        return 0;
}

int8_t leon3_print_uint32(uint32_t i){
                int aux=i;
                int vector[10];
                int cont=0;

                do{     //alamcenamos en orden desde unidades hacia arriba
                        vector[cont]=aux%10;
                        aux=aux/10;
                        cont++;
                }while(aux%10!=0);
```

```
                do{     //mostramos en orden desde el ultimo insertado hasta unidades
                        cont--;
                        leon3_putchar('0'+vector[cont]);
                }while(cont!=0);

                leon3_putchar('\n');        //salto de linea para que sea mas legible
                return 0;
}


/*
 * leon3_traps.c
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */
#include "leon3_traps.h"  //Incluimos los archivos necesario para poder usar sus funciones
#include "leon3_irqs.h"
#include "leon3_bprint.h"

uint32_t _rdtbr();

asm(".text\n"
"_rdtbr:\n"
"    retl \n"
"    mov %tbr, %o0");

typedef void (*pfunc_t)(void);

#define SPARC_NUM_HW_IRQS 0xF

static pfunc_t hw_irq_handler[SPARC_NUM_HW_IRQS];

//void leon3_sparc_set_trap_handler(uint32_t trap_num, uint32_t handler_routine)
void leon3_sparc_set_trap_handler(uint32_t trap_num, void (* handl_routine) (void)){
        uint32_t handler_routine=(uint32_t)handl_routine;
   uint32_t * trap_address;
   uint32_t tbr = _rdtbr();

   // This are the instructions we want to code:
   // A1 48 00 00  rd  %psr, %l0
   // 29 00 80 0f  sethi  %hi(handler_routine), %l4
   // 81 c5 21 a0  jmp  %l4 + %lo(handler_routine)
   // a6 10 20 05  mov trap_num, %l3

   trap_address = (uint32_t *) ((tbr & ~0x0fff) | ((uint32_t) trap_num << 4));
   *trap_address = 0xA1480000UL;
   trap_address++;
   *trap_address = (0x29000000UL | (handler_routine >> 10));
   trap_address++;
   *trap_address = (0x81C52000UL | ((handler_routine & ((1 << 10) - 1))));
   trap_address++;
   *trap_address = (0xA6102000UL | (trap_num & 0x1FFF));
}

//leon3_traps.c. Implementa la función leon3_install_handler que permite
//instalar una rutina de atención a un evento, independientemente de si es una
//interrupción, un trap o una excepción. El prototipo de la función es el siguiente,
//donde vector_num es el número de vector y hanlder la rutina a instalar
int32_t leon3_install_handler(uint32_t trap_num, void (* handl_routine) (void)){
   // do not modify sytem traps
        if ((trap_num < 0x11)){
```

```
        return LEON3_ERR_TRAP_HANDLER;
    }

    if ((trap_num > 0x1F)){
        leon3_sparc_set_trap_handler(trap_num,  handl_routine);
    }
    else{
        leon3_sparc_set_trap_handler(trap_num, leon3_sparc_isr);
        hw_irq_handler[trap_num - 0x11] = handl_routine;
    }
    return LEON3_ERR_SUCCESS;
}

void leon3_sparc_irqhandler_entry(uint32_t hw_irq_level){
            leon3_print_string("hw irq level ");
            leon3_print_uint32(hw_irq_level- 0x11);
            leon3_print_string("\n");
    if(hw_irq_handler[hw_irq_level - 0x11]){
            hw_irq_handler[hw_irq_level - 0x11]();
    }
}
```