```
/*Programa3INCLUDE
 * leon3_types.h
 *
 *  Created on: Feb 5, 2016
 *      Author: pedro
 */

//Declaracion de los tipos de datos con sus formatos concretos y sus tamaños

#ifndef LEON3_TYPES_H
#define LEON3_TYPES_H

typedef unsigned char byte_t;
typedef unsigned short int word12_t;
typedef unsigned int word32_t;
typedef unsigned long int word64_t;

typedef signed char int8_t;
typedef signed short int int16_t;
typedef signed int int32_t;
typedef signed long int int64_t;

typedef unsigned char uint8_t;
typedef unsigned short int uint16_t;
typedef unsigned int uint32_t;
typedef unsigned long int uint64_t;

#endif

/*
 * leon3_uart.h
 *
 *  Created on: Feb 5, 2016
 *      Author: pedro
 */
//Declracion de la funcion usada en el main
#ifndef LEON3_UART_H_
#define LEON3_UART_H_

#include "leon3_types.h"          //Incluye los tipos de tado de types.h

int8_t leon3_putchar(char c);   //especificacion de la fincion

#endif /* LEON3_UART_H_ */


/*
 * traps.h
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */

#ifndef TRAPS_H_
#define TRAPS_H_
```

```
#include "leon3_types.h"


// Public functions

int32_t leon3_install_handler(uint32_t trap_vector, void (* handl_routine) (void));


// Error list

#define LEON3_ERR_SUCCESS            0
#define LEON3_ERR_TRAP_HANDLER       -1
#define LEON3_ERR_INIT               -2


#define LEON3_SPARC_DIV_0_TRAPVECTOR                  0x82
#define LEON3_SPARC_ENABLE_IRQ_TRAPVECTOR   0x83
#define LEON3_SPARC_DISABLE_IRQ_TRAPVECTOR  0x84

#endif /* TRAPS_H_ */



/*
 * leon3_irqs.h
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */

#ifndef IRQS_H_
#define IRQS_H_

#include "leon3_types.h"

void leon3_trap_handler_enable_irqs(void);

void leon3_trap_handler_disable_irqs(void);

uint8_t leon3_mask_irq (int32_t irq_level);

uint8_t leon3_unmask_irq (int32_t irq_level);

void leon3_sys_call_enable_irqs(void);

void leon3_sys_call_disable_irqs(void);

uint8_t leon3_sparc_force_irq (int32_t irq_level);

void leon3_sparc_isr(void);

#endif


/*
 * leon3_bprint.h
```

```
 *
 *  Created on: Feb 12, 2016
 *      Author: pedro
 */

#ifndef LEON3_BPRINT_H_
#define LEON3_BPRINT_H_

#include "leon3_types.h"

//transmite a través del puerto serie la cadena de caracteres
//pasada por parámetro.
int8_t leon3_print_string(char* str);

//transmite a través del puerto serie el entero transformado en
//cadena de caracteres.
int8_t leon3_print_uint8(uint8_t i);

//Esta función implementa una función análoga a la función
//leon3_print_uint8 que se realizó en la práctica anterior, pero trabajando con
//un entero de 32 bits.
int8_t leon3_print_uint32(uint32_t i);


#endif /* LEON3_BPRINT_H_ */

/*
 * leon3_asm.h
 *
 *  Created on: Feb 28, 2013
 *      Author: user
 */

#ifndef ASM_H_
#define ASM_H_


#define CPU_STACK_ALIGNMENT        16

/*
 * WINDOW REGISTERS
 */

/*
 *  Offsets of fields with Context_Control for assembly routines.
 */

/* Global registers */
#define G0_OFFSET    0x00
#define G1_OFFSET    0x04
#define G2_OFFSET    0x08
#define G3_OFFSET    0x0C
#define G4_OFFSET    0x10
#define G5_OFFSET    0x14
#define G6_OFFSET    0x18
#define G7_OFFSET    0x1C
```

```
/*  Local registers */
#define L0_OFFSET    0x20
#define L1_OFFSET    0x24
#define L2_OFFSET    0x28
#define L3_OFFSET    0x2C
#define L4_OFFSET    0x30
#define L5_OFFSET    0x34
#define L6_OFFSET    0x38
#define L7_OFFSET    0x3C

/*  Input registers */
#define I0_OFFSET    0x40
#define I1_OFFSET    0x44
#define I2_OFFSET    0x48
#define I3_OFFSET    0x4C
#define I4_OFFSET    0x50
#define I5_OFFSET    0x54
#define I6_FP_OFFSET 0x58
#define I7_OFFSET    0x5C

/*  Output registers    */
#define O0_OFFSET    0x60
#define O1_OFFSET    0x64
#define O2_OFFSET    0x68
#define O3_OFFSET    0x6C
#define O4_OFFSET    0x70
#define O5_OFFSET    0x74
#define O6_SP_OFFSET 0x78
#define O7_OFFSET    0x7C

/** PSR register    */
#define PSR_OFFSET   0x80

#define Y_OFFSET   0x84

/**  Context control size    */
#define CONTEXT_CONTROL_SIZE 0x88

#define CPU_STACK_FRAME_L0_OFFSET          0x00
#define CPU_STACK_FRAME_L1_OFFSET          0x04
#define CPU_STACK_FRAME_L2_OFFSET          0x08
#define CPU_STACK_FRAME_L3_OFFSET          0x0c
#define CPU_STACK_FRAME_L4_OFFSET          0x10
#define CPU_STACK_FRAME_L5_OFFSET          0x14
#define CPU_STACK_FRAME_L6_OFFSET          0x18
#define CPU_STACK_FRAME_L7_OFFSET          0x1c
#define CPU_STACK_FRAME_I0_OFFSET          0x20
#define CPU_STACK_FRAME_I1_OFFSET          0x24
#define CPU_STACK_FRAME_I2_OFFSET          0x28
#define CPU_STACK_FRAME_I3_OFFSET          0x2c
#define CPU_STACK_FRAME_I4_OFFSET          0x30
#define CPU_STACK_FRAME_I5_OFFSET          0x34
#define CPU_STACK_FRAME_I6_FP_OFFSET       0x38
#define CPU_STACK_FRAME_I7_OFFSET          0x3c
```

```
/*  Minimum Stack frame size    */
#define CPU_MINIMUM_STACK_FRAME_SIZE         0x60


#define CONTEXT_CONTROL_INTERRUPT_FRAME_SIZE CPU_MINIMUM_STACK_FRAME_SIZE + 0x50


#define SPARC_NUMBER_OF_REGISTER_WINDOWS     8


#define LEON3_SPARC_ENABLE_IRQ_TRAPNUM  3
#define LEON3_SPARC_DISABLE_IRQ_TRAPNUM 4


/*******************************************************************************
 * PSR REGISTER MASKS
 *******************************************************************************/

/* The SPARC PSR fields are laid out as the following:
 *
 *  ------------------------------------------------------------------------
 *  | impl  | vers  | icc   | resv  | EC | EF | PIL  | S | PS | ET |  CWP
 *  |
 *  | 31-28 | 27-24 | 23-20 | 19-14 | 13 | 12 | 11-8 | 7 | 6  | 5  |  4-0
 *  |
 *  ------------------------------------------------------------------------
 */
#define PSR_CWP     0x0000001f       /**< \brief Current window pointer     */
#define PSR_ET      0x00000020       /**< \brief Enable traps field */
#define PSR_PS      0x00000040       /**< \brief Previous privilege level   */
#define PSR_S       0x00000080       /**< \brief Current privilege level    */
#define PSR_PIL     0x00000f00       /**< \brief Processor interrupt level  */
#define PSR_EF      0x00001000       /**< \brief Enable floating point      */
#define PSR_EC      0x00002000       /**< \brief Enable co-processor */
#define PSR_LE      0x00008000       /**< \brief SuperSparcII little-endian */
#define PSR_ICC     0x00f00000       /**< \brief Integer condition codes    */
#define PSR_C       0x00100000       /**< \brief Carry bit */
#define PSR_V       0x00200000       /**< \brief Overflow bit */
#define PSR_Z       0x00400000       /**< \brief Zero bit */
#define PSR_N       0x00800000       /**< \brief Negative bit */
#define PSR_VERS    0x0f000000       /**< \brief CPU-version field */
#define PSR_IMPL    0xf0000000       /**< \brief CPU-implementation field   */


//------------------------------------------------------------------------

/*******************************************************************************
 * TRAPS
 *******************************************************************************/

/**
 * Software trap
 * This macro provokes a software trap
 */
#define SOFT_TRAP    ta 0;nop;nop;nop;
```

```
/**
 * This macro is for traps we should NEVER get
 * The PSR is stored in the %lo register
 * The TRAP TYPE is stored in the %l7 register
 * The WIM is stored in the %l3 register
 */
/*#define BAD_TRAP(num) \
       rd %psr, %l0; mov num, %l7; b bad_trap_handler; rd %wim, %l3;
*/
/**
 * The TRAP_ENTRY macro:
 *  - moves the status register (PSR) into the L0 register
 *   - moves the window invalid register (WIM) into the L4 register
 *   - moves the trap type into the L3 register
 *   - branch to the low_trap entry that manages the trap interrupt
 */
#define TRAP_ENTRY(type) mov %psr, %l0; mov %wim, %l4;\
                         b irq_entry;mov type, %l3

/**
 *  Entry for traps which jump to a programmer-specified trap handler.
 */
/*
#define TRAP(_vector, _handler)  \
  mov   %psr, %l0 ; \
  sethi %hi(_handler), %l4 ; \
  jmp   %l4+%lo(_handler); \
  mov   _vector, %l3
*/

#define PUBLIC(sym) .globl sym

/**
 * This macro decrements the value of a variable passed as an argument.
 * The value of the variable is stored in the reg parameter.
 *
 * @param variable: this parameter is the variable to be decremented.
 * @param reg:      this parameter is the register that stores the final value
 * of the variable.
 *
 * The macro use the %l4 local register.
 */
#define DEC_VARIABLE(variable, reg) \
    sethi   %hi(variable), %l4;   \
    ld      [%l4 + %lo(variable)], %reg;   \
    dec     %reg; \
    st      %reg, [%l4 + %lo(variable)];

/**
 * This macro increments the value of a variable passed as an argument.
 * The value of the variable is stored in the reg parameter.
 *
 * @param variable: this parameter is the variable to be incremented.
 * @param reg:      this parameter is the register that stores the final value
 * of the variable.
 *
```

```
 * The macro use the %l4 local register.
 */
#define INC_VARIABLE(variable, reg) \
   sethi   %hi(variable), %l4;   \
   ld      [%l4 + %lo(variable)], %reg;   \
   inc     %reg;   \
   st      %reg, [%l4 + %lo(variable)];

/**
 * This macro verifies the value of the variable passed as argument and branch
 * to the specified label in case of the variable is set.
 *
 * @param variable: This parameter is the variable that must be tested.
 * @param label:    This parameter is the branch label.
 *
 * The macro use the %l4 and %l5 local registers.
 */
#define BRANCH_IF_VARIABLE_SET(variable, label)    \
   sethi   %hi(variable), %l4;   \
   ld      [%l4 + %lo(variable)], %l5;   \
   cmp     %l5, 0;   \
   bnz     label;   \
   nop;

/**
 * This macro sets the variable passed as paramenter.
 *
 * @param variable: This parameter is the parameter to be set.
 *
 * The macro use the %l5 and %l4 registers.
 */
#define SET_VARIABLE(variable)  \
   mov     1, %l5;     \
   sethi   %hi(variable), %l4;   \
   st      %l5, [%l4 + %lo(variable)];     \

/**
 * This macro saves the PIL stored in the reg register.
 *
 * @param reg:  This parameter is the registers that will store the PIL
 */
#define SAVE_PIL(reg)   \
   rd %psr, %reg;   \
   and %reg, PSR_PIL, %reg;

/**
 * This macro restores the PIL stored in the reg register.
 *
 * @param reg:  This parameter is the registers that stores the PIL
 * @param raux:   This parameter is an auxiliar register
 */
#define RESTORE_PIL(reg,raux)   \
   rd %psr, %raux; \
   wr %raux, %reg, %psr; \
   WRITE_PAUSE
```

```
/**
 * This macro performs a pause
 */
#define WRITE_PAUSE nop;nop;nop

/**
 * This macro installs a new PIL.
 *
 * @level:  This parameter is the interrupt level
 * @oreg:   This parameter stores the new PIL value
 */
#define CALCULATE_PIL(level,oreg)   \
    and %level, 0xF, %oreg;   \
    sll %oreg, 8, %oreg;   \

#define RTRAP(_vector, _handler)  \
    mov   %g0, %l0 ; \
    sethi %hi(_handler), %l4 ; \
    jmp   %l4+%lo(_handler); \
    mov   _vector, %l3


#endif /* ASM_H_ */
```