/\*

\* Engr220L - Lab 10

\* Date: Nov 9, 2022

\* Author: ZeAi Sun

\* The lab 10 program.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*/

/\* INCLUDES \*/

/\*\*\*\*\*\*\*\*\*\*\*\*/

.include "nios\_macros.s"

.include "nios\_defsISR.s" /\* system specific definitions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* CONSTANTS \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

.equ MEM\_END\_ADDR, 0x8000

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* TEXT SECTION \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.text

/\* Place the main routine at the reset address \*/

.org RESET\_VECTOR

/\* Program start location must be identified \*/

.global \_start

\_start:

/\* jump over the ISR code in order to begin execution at MAIN\_PROG\_INIT \*/

br MAIN\_PROG\_INIT

/\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ISR CODE \*/

/\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ISR = Interrupt Service Routine \*/

.org EXCEPTION\_VECTOR

ISR:

/\* these three lines of code should not be changed \*/

rdctl et, ctl4 /\* Check if an external (hardware) intr has occurred \*/

beq et, r0, EXCEPTION\_ACTION

subi ea, ea, 4 /\* If yes, decrement ea to re-execute interrupted

instruction when you return from the ISR \*/

EXCEPTION\_ACTION:

/\* The interrupt-service/exception-handler routine: After determining

the source of the interrupt, the interrupt condition must be cleared.

Note that if any register besides r0, et, ea, and sp are used, they must

first be pushed on the stack and then pulled off the stack before

returning from the interrupt (eret). \*/

addi sp, sp, -36 # Push registers onto the stack.

stw r6, 32(sp)

stw r11, 28(sp)

stw r12, 24(sp)

stw r13, 20(sp)

stw r14, 16(sp)

stw r15, 12(sp)

stw r16, 8(sp)

stw r17, 4(sp)

stw r18, 0(sp)

CHECK\_FOR\_INTR\_0:

rdctl et, ctl4 # Read the IPENDING register ctl4.

andi et, et, TIMER\_MASK # Mask other bits except the bit 0.

beq r0, et, CHECK\_FOR\_INTR\_1 # If bit 0 is 0, check for the next interrupt.

RESPOND\_TO\_INTR\_0:

stwio r0, STATUS\_OFFSET(r6) # Clear the TO flag.

movia r11, ledDisplayValue # Store the global count value in r11.

ldw r12, 0(r11) # Load the global count value into r12.

addi r12, r12, 1 # Increment the value of r13.

movia r13, 0x40000 # Store the max value 0x40000 in r14.

beq r12, r13, GLOBAL\_RESET # Check if needs to reset the global count.

br NEXT\_STEP # If not, branch to the next step.

GLOBAL\_RESET: # If the global count >= 0x40000, reset to 0.

movia r12, 0

NEXT\_STEP:

stw r12, 0(r11) # Store the value on r12 into global count.

movia r14, LEDR\_BASE # Display the global count value on the LEDs.

stwio r12, 0(r14)

CHECK\_FOR\_INTR\_1:

rdctl et, ctl4 # Read the IPENDING register ctl4.

andi et, et, KEY\_MASK # Mask other bits except the bit 1.

beq r0, et, END\_ISR # If bit 1 is 0, jump to the end of ISR.

RESPOND\_TO\_INTR\_1:

movia r15, KEY\_BASE

stwio r0, EDGE\_OFFSET(r15) # Clear the EDGE register.

movia r15, SW\_BASE

ldwio r16, DATA\_OFFSET(r15) # Load the SW data into r16.

movia r17, TIMER\_BASE

stwio r16, PERIODH\_OFFSET(r17)# Store the SW data into PERIODH register.

stwio r0, STATUS\_OFFSET(r17) # Clear the STATUS register.

movia r18, 0b0111

stwio r18, CONTROL\_OFFSET(r17)# Store the value in the Control register.

CHECK\_FOR\_INTR\_2:

/\* TODO: check if intr 2 needs service by checking bit 2 of ipending (ctl4). \*/

/\* TODO: this should execute every time the ISR is called, even if another check-#==yes \*/

RESPOND\_TO\_INTR\_2:

/\* TODO: The interrupt 2 action goes here, this should only execute if check-2==yes \*/

/\* The cpu does the following when you return from the interrupt:

1. Copies contents of estatus (ctl1) to status (ctl0) (restores it).

2. Transfers program execution to the address in ea register (r29)

by copying what is in ea to the program counter (pc).

NOTE: If you pushed any registers on the stack, pop them off now. \*/

END\_ISR:

eret /\* Return from exception \*/

ldw r6, 32(sp) # Pop registers out from the stack.

ldw r11, 28(sp)

ldw r12, 24(sp)

ldw r13, 20(sp)

ldw r14, 16(sp)

ldw r15, 12(sp)

ldw r16, 8(sp)

ldw r17, 4(sp)

ldw r18, 0(sp)

addi sp, sp, 36

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* END ISR CODE \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* MAIN PROGRAM CODE \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MAIN\_PROG\_INIT:

# br MAIN\_PROG\_INIT # An infinite loop that does nothing.

SP\_INIT:

movia sp, MEM\_END\_ADDR

GLOBAL\_VARIABLES\_INIT:

movia r15, ledDisplayValue

stw r0, 0(r15)

IO\_BASE\_INIT:

/\* TODO: setup registers with I/O base addresses if needed \*/

IO\_DEVICE\_INIT:

/\* TODO: initialize PIO devices if needed \*/

TIMER\_INIT:

movia r6, TIMER\_BASE

movia r7, 0x7840 # Store the value in the PeriodL register.

stwio r7, PERIODL\_OFFSET(r6)

movia r8, 0x17D # Store the value in the PeriodH register.

stwio r8, PERIODH\_OFFSET(r6)

SET\_PORT\_INTR:

movia r9, 0b10 # Store the value in the Status register.

stwio r9, STATUS\_OFFSET(r6)

movia r10, 0b0111 # Store the value in the Control register.

stwio r10, CONTROL\_OFFSET(r6)

movia r11, KEY\_BASE

stwio r0, EDGE\_OFFSET(r11) # Clear the EDGE register.

movia r12, 0b0100 # Set the value of MASK register so that KEY2 can cause interrupt.

stwio r12, MASK\_OFFSET(r11)

SET\_IENABLE:

/\* Set the IENABLE for control register 3 \*/

rdctl et, ctl3 /\* Read the interrupt enable register = ctrl\_reg3 \*/

ori et, et, TIMER\_MASK /\* set the timer interrupt enable bit high \*/

ori et, et, KEY\_MASK # Set the key interrupt enable bit high.

wrctl ctl3, et /\* write the final pattern back to IENABLE (ctl3) \*/

SET\_STATUS:

/\* Now enable interrupts globally in the processor status register. \*/

rdctl et, ctl0 /\* Read the status register = ctrl\_reg0 \*/

ori et, et, PIE\_MASK /\* set the PIE bit to enable all interrupts \*/

wrctl ctl0, et /\* write the pattern back to STATUS (ctl0) \*/

/\* End MAIN\_PROG\_INIT \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MAIN\_PROG:

/\* TODO write main program functionality as needed here \*/

MAIN\_PROG\_END:

/\* infinite loop to keep out of global memory, useful for final breakpoint \*/

br MAIN\_PROG\_END

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* END MAIN PROGRAM CODE \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DATA SECTION \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.data

/\* TODO: if needed, add .word or .skip declarations here for global variables \*/

ledDisplayValue: /\* Renamed global variable \*/

.word 0 /\* allocate 4 bytes and initialize them to 0 \*/

.end