/\*

\* Engr220L - Lab 10

\* Date: 11/8/2019

\* Author: Jacob Brink

\*

\* Does Lab 9 but on Higher Difficulty

\*

\*/

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

/\* INCLUDES \*/

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

.include "nios\_macros.s"

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

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

/\* CONSTANTS \*/

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

/\* TODO: add your own .equ statements here \*/

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

/\* 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 \*/

/\* The following happens when an external interrupt occurs: The cpu:

1. Copies the contents of the status register (ctl0) to estatus

(ctl1) saving the processor’s pre-exception status

2. Clears the U bit of the status register, forcing the processor

into supervisor mode

3. Clears the PIE bit of the status register, disabling external

processor interrupts

4. Writes the address of the instruction after the exception to

the ea register (r29)

5. Transfers execution to the address of the exception handler that

determines the cause of the interrupt

\*/

.org EXCEPTION\_VECTOR

ISR:

subi sp, sp, 12

stw r10, 0(sp)

stw r11, 4(sp)

stw r12, 8(sp)

/\* 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). \*/

/\* TODO: push to stack any registers that will change (except as noted above) \*/

/\* it is also allowable to push and pop for just certain cases of the ISR \*/

CHECK\_FOR\_INTR\_0:

/\* TODO: check if intr 0 needs service by checking bit 0 of ipending (ctl4). \*/

rdctl r11, ctl4

andi r10, r11, 0b1

beq r10, r0, CHECK\_FOR\_INTR\_1

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

RESPOND\_TO\_INTR\_0:

/\* TODO: The interrupt 0 action goes here, this should only execute if check-0==yes \*/

stwio r0, STATUS\_OFFSET(gp)

addi r10, r0, 0x40000

ldwio r12, COUNTER\_HERO(r0)

blt r12, r10, SKIP\_RESET

add r12, r0, r0

SKIP\_RESET:

addi r12, r12, 0x1

movia r17, LEDR\_BASE

stwio r12, 0(r17)

stwio r12, COUNTER\_HERO(r0)

CHECK\_FOR\_INTR\_1:

/\* TODO: check if intr 1 needs service by checking bit 1 of ipending (ctl4). \*/

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

rdctl r11, ctl4

movi r12, KEY\_MASK

and r10, r11, r12

beq r10, r0, END\_ISR

RESPOND\_TO\_INTR\_1:

/\* TODO: The interrupt 1 action goes here, this should only execute if check-1==yes \*/

/\* clear EDGE register \*/

movia r11, KEY\_BASE

stwio r0, EDGE\_OFFSET(r11)

/\* set period h to sw something idk really tbh \*/

movia r12, SW\_BASE

ldwio r11, 0(r12)

stwio r11, PERIODH\_OFFSET(gp)

/\* restart timer \*/

/\* Turning on START, CONT, and ITO bit to be on \*/

addi r10, r0, 0b111

stwio r10, CONTROL\_OFFSET(gp)

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:

/\* TODO: pop from stack anything that was pushed \*/

ldw r10, 0(sp)

ldw r11, 4(sp)

ldw r12, 8(sp)

addi sp, sp, 12

eret /\* Return from exception \*/

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

/\* END ISR CODE \*/

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

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

/\* MAIN PROGRAM CODE \*/

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

MAIN\_PROG\_INIT:

/\* TODO: the main program startup code goes here \*/

SP\_INIT:

/\* TODO: initialize stack pointer if needed \*/

movia sp, 0x7fff

GLOBAL\_VARIABLES\_INIT:

/\* TODO: initialize global variables if needed \*/

stwio r0, COUNTER\_HERO(r0)

IO\_BASE\_INIT:

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

IO\_DEVICE\_INIT:

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

TIMER\_INIT:

/\* TODO: initialize the timers if needed \*/

/\* initialize the timer with the proper timeout period \*/

movia gp, TIMER\_BASE

/\* initialize PERIODH and PERIODL to PERIOD \*/

addi r10, r0, 0x17D

stwio r10, PERIODH\_OFFSET(gp)

addi r10, r0, 0x783F

stwio r10, PERIODL\_OFFSET(gp)

/\* Turn on RUN bit in status\_control \*/

stwio r0, STATUS\_OFFSET(gp)

SET\_PORT\_INTR:

/\* TODO: Setup all I/O ports for interrupts: \*/

/\* 1) clear the intr flag in each port and \*/

/\* 2) turn interrupts on in each port. \*/

/\* clear status \*/

stwio r0, STATUS\_OFFSET(gp)

/\* Turning on START, CONT, and ITO bit to be on \*/

addi r10, r0, 0b111

stwio r10, CONTROL\_OFFSET(gp)

/\* 1.a Resetting edge status for buttons \*/

movia r11, KEY\_BASE

stwio r0, EDGE\_OFFSET(r11)

/\* 1.b Enabling interrupt on button 4 (KEY2) \*/

movi r10, 0x4

stwio r10, MASK\_OFFSET(r11)

SET\_IENABLE:

/\* TODO: enable each interrupt in the IENABLE reg \*/

/\* Set the IENABLE for control register # as needed, \*/

/\* ctl3 is shown as an example \*/

/\* 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 /\* 2: set interrupt for button \*/

/\* TODO: use additional ori instructions to turn on other interrupts \*/

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 \*/

/\* Infinite Loop \*/

DO\_NOTHING:

br DO\_NOTHING

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 \*/

/\* TODO replace MYGLOBALVAR with whichever global variable(s) you need\*/

COUNTER\_HERO: /\* "COUNTER\_HERO" is considered the name of the global var \*/

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

.end