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

/\* Name: Daniel Ackuaku \*/

/\* Course: ENGR \*/

/\* Lab #:6 Part 1 \*/

/\* Date: 25th March, 2019 \*/

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

/\*TEMPLATE fill in the title block above \*/

.include "nios\_macros.s"

.include "nios\_iodefs.s" /\* include i/o constants \*/

.include "nios\_irqdefs.s" /\* include interrupt constants \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constant Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE define any main-code specific .equ constants here \*/

.equ TIMER\_CONTROL\_VALUE, 0b0111

.equ BITMASK\_TIMEOUT, 0x01

.equ BITMASK\_KEY\_PRESS\_EDGE, 0b01000

.equ MAX\_COUNTER, 0x40000

.text

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code Section \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org RESET\_VECTOR /\* Place the main routine at the reset address \*/

.global \_start

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

br MAIN\_PROG\_INIT /\* jump over the ISR code \*/

/\* make sure above code fits between the RESET and EXCEPTION addresses!! \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ISR=Interrupt Service Routine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org EXCEPTION\_VECTOR

ISR:

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

first be pushed on the stack before doing the work of the ISR. At the

end of the ISR, those same registers must be popped off the stack before

returning from the interrupt (eret) so that the interruptED code is not

affected by the work of the ISR. Do the pushing here. \*/

/\*TEMPLATE push registers if needed \*/

/\*this is where you push r9\*/

addi sp, sp, -0x4

stwio r9, 0(sp)

/\* NOTE: These three lines of code should NOT be changed! \*/

/\* Check to see if an external intr has occurred (IPENDING). \*/

rdctl et, ctl4 /\* ctl4 = IPENDING \*/

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. \*/

CHECKFOR\_INTR0:

/\*TEMPLATE check if intr 0 needs service by checking bit 0 of IPENDING (ctl4). \*/

rdctl et, ctl4

andi et, et, BITMASK\_TIMEOUT

bne et, r0, RESPONDTO\_INTR0

br CHECK\_KEY\_PRESS

RESPONDTO\_INTR0:

/\*TEMPLATE The interrupt 0 action goes here, including clearing the interrupt condition. \*/

movia et, TIMER\_BASE /\* loads the address of TIMER\_BASE into et\*/

stwio r0, STATUS\_OFFSET(et) /\* clears the flag by setting the Status of

TIMER\_BASE to 0\*/

movia et, COUNTER /\* loads the value of the counter into et\*/

ldw r9, 0(et) /\* stores et in r9\*/

addi r9, r9, 1

movia et, MAX\_COUNTER

bltu r9, et, DISPLAY /\* if et(MAX\_COUNTER) = r9 LED\_RESET else DISPLAY\*/

br LED\_RESET

LED\_RESET:

mov r9, r0

DISPLAY:

movia et, LEDR\_BASE /\* loads the address of the LEDR\_BASE into et\*/

stwio r9, DATA\_OFFSET(et) /\* stores r9 into et\*/

/\* updates the COUNTER \*/

movia et, COUNTER

stw r9, DATA\_OFFSET(et)

END\_ISR:

/\*TEMPLATE If you pushed any registers on the stack at the beginning, pop them off now. \*/

/\*this is where you pop r9\*/

ldwio r9, 0(sp)

addi sp, sp, 0x4

eret /\* Return from the exception \*/

/\* end of exception handler routine (ISR) \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

MAIN\_PROG\_INIT: /\* should be the first label after the ISR \*/

movia r20, LEDR\_BASE /\* base location of PIO registers for LED device \*/

movia r22, TIMER\_BASE /\* base location for timer I/O device \*/

movia r23, KEY\_BASE /\* base location for timer KEY device \*/

movia r25, SW\_BASE /\* base location for timer SW\_BASE \*/

/\*TEMPLATE initialize global variables if needed \*/

GLOBVAR\_INIT:

movia r10, MAX\_COUNTER /\* the limit for the counter i\*/

/\*TEMPLATE initialize timer port registers (period and control) if needed \*/

TIMER\_INIT:

movia r13, TIMER\_CONTROL\_VALUE

movi r14, 0x017d /\* the least significant bit for the timer\*/

movia r15, 0x7840 /\* the most significant bit for the timer\*/

stwio r15, PERIODL\_OFFSET(r22) /\* stores most sig period bits into period H \*/

stwio r14, PERIODH\_OFFSET(r22) /\* stores least sig period bits into period H \*/

stwio r13, CONTROL\_OFFSET(r22) /\* stores the stores into period H \*/

stwio r0, STATUS\_OFFSET(r22)

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

/\*\*\*\*\*\*\*\*\* Interrupt Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE Now enable each interrupt in the IENABLE register (one is shown). \*/

SET\_IENABLE:

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

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

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

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

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

SET\_STATUS:

rdctl et, ctl0 /\* Read the status register \*/

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

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

/\*TEMPLATE main program code goes here \*/

MAIN\_PROG\_EXEC:

br CHECK\_KEY\_PRESS

/\* Checks for a key press\*/

CHECK\_KEY\_PRESS:

ldwio r21, EDGE\_OFFSET(r23) /\* reads the value of 12(KEYPRESS) into r23 \*/

andi r21, r21, BITMASK\_KEY\_PRESS\_EDGE /\* uses a mask to get the value of Key3 \*/

bne r21, r0, UPDATE\_TIMER

br MAIN\_PROG\_EXEC

UPDATE\_TIMER:

stwio r0, EDGE\_OFFSET(r23) /\* resets the value of EDGE(KEYPRESS) \*/

ldwio r21, DATA\_OFFSET(r25) /\* EDGE(KEYPRESS) \*/

stwio r21, PERIODH\_OFFSET(r22) /\* Update the period High\*/

stwio r13, CONTROL\_OFFSET(r22) /\* resets the value of timer CONTROL \*/

mov r21, r0

br CHECK\_KEY\_PRESS

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variable Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.data

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

COUNTER:

.end

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

/\* Name: Daniel Ackuaku \*/

/\* Course: ENGR \*/

/\* Lab #:6 Part 2 \*/

/\* Date: 25th March, 2019 \*/

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

/\*TEMPLATE fill in the title block above \*/

.include "nios\_macros.s"

.include "nios\_iodefs.s" /\* include i/o constants \*/

.include "nios\_irqdefs.s" /\* include interrupt constants \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constant Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE define any main-code specific .equ constants here \*/

.equ TIMER\_CONTROL\_VALUE, 0b0111

.equ BITMASK\_TIMEOUT, 0x01

.equ BITMASK\_KEY\_PRESS\_EDGE, 0b01000

.equ BITMASK\_KEY\_INTERUPT\_CHECK, 0b10

.equ MAX\_COUNTER, 0x40000

.text

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code Section \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org RESET\_VECTOR /\* Place the main routine at the reset address \*/

.global \_start

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

br MAIN\_PROG\_INIT /\* jump over the ISR code \*/

/\* make sure above code fits between the RESET and EXCEPTION addresses!! \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ISR=Interrupt Service Routine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org EXCEPTION\_VECTOR

ISR:

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

first be pushed on the stack before doing the work of the ISR. At the

end of the ISR, those same registers must be popped off the stack before

returning from the interrupt (eret) so that the interruptED code is not

affected by the work of the ISR. Do the pushing here. \*/

/\*TEMPLATE push registers if needed \*/

/\*this is where you push registers used in the ISR\*/

addi sp, sp, -0x8

stwio r9, 0(sp)

stwio r13, 4(sp)

/\* NOTE: These three lines of code should NOT be changed! \*/

/\* Check to see if an external intr has occurred (IPENDING). \*/

rdctl et, ctl4 /\* ctl4 = IPENDING \*/

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. \*/

CHECKFOR\_INTR0:

/\*TEMPLATE check if intr 0 needs service by checking bit 0 of IPENDING (ctl4). \*/

rdctl et, ctl4 /\* loads ipending into et\*/

andi et, et, BITMASK\_TIMEOUT /\* perform bit mask \*/

bne et, r0, RESPONDTO\_INTR0 /\* if et is 0 RESPONDTO\_INTR0\*/

br CHECKFOR\_INTR1

RESPONDTO\_INTR0:

/\*TEMPLATE The interrupt 0 action goes here, including clearing the interrupt condition. \*/

movia et, TIMER\_BASE /\* loads the address of TIMER\_BASE into et\*/

stwio r0, STATUS\_OFFSET(et) /\* clears the flag by setting the Status of

TIMER\_BASE to 0\*/

movia et, COUNTER /\* loads the value of the counter into et\*/

ldw r9, 0(et) /\* stores et in r9\*/

addi r9, r9, 1

movia et, MAX\_COUNTER

bltu r9, et, DISPLAY /\* if et(MAX\_COUNTER) = r9 LED\_RESET else DISPLAY\*/

br LED\_RESET

LED\_RESET:

mov r9, r0

DISPLAY:

movia et, LEDR\_BASE /\* loads the address of the LEDR\_BASE into et\*/

stwio r9, DATA\_OFFSET(et) /\* stores r9 into et\*/

/\* updates the COUNTER \*/

movia et, COUNTER

stw r9, DATA\_OFFSET(et)

/\*TEMPLATE if needed, check interrupt 1 for servicing - bit 1 of IPENDING \*/

CHECKFOR\_INTR1:

rdctl et, ctl4 /\* loads ipending into et\*/

andi et, et, BITMASK\_KEY\_INTERUPT\_CHECK /\* perform bit mask \*/

bne et, r0, RESPONDTO\_INTR1 /\* if et is 0 RESPONDTO\_INTR1\*/

br CHECKFOR\_INTR2

/\*TEMPLATE The interrupt 1 action goes here, including clearing the interrupt condition. \*/

RESPONDTO\_INTR1:

movia et, KEY\_BASE /\* loads the address of KEY\_BASE into et\*/

mov r9, r0

stwio r0, EDGE\_OFFSET(et) /\* clear edge flag \*/

movia et, SW\_BASE

ldwio r9, DATA\_OFFSET(et) /\* get switch value \*/

movia et, TIMER\_BASE

stwio r9, PERIODH\_OFFSET(et) /\* new of PERIODH\_OFFSET(TIMER\_BASE) \*/

movia r13, TIMER\_CONTROL\_VALUE

stwio r13, CONTROL\_OFFSET(et) /\* restart timer \*/

END\_ISR:

/\*TEMPLATE If you pushed any registers on the stack at the beginning, pop them off now. \*/

/\*this is where you pop registers used in the ISR\*/

ldwio r9, 0(sp)

ldwio r13, 4(sp)

addi sp, sp, 0x8

eret /\* Return from the exception \*/

/\* end of exception handler routine (ISR) \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

MAIN\_PROG\_INIT: /\* should be the first label after the ISR \*/

movia r20, LEDR\_BASE /\* base location of PIO registers for LED device \*/

movia r22, TIMER\_BASE /\* base location for timer I/O device \*/

movia r25, SW\_BASE /\* base location for timer SW\_BASE \*/

/\*TEMPLATE initialize global variables if needed \*/

GLOBVAR\_INIT:

movia r10, MAX\_COUNTER /\* the limit for the counter i\*/

/\*TEMPLATE init CPU registers for holding I/O port base addresses if needed \*/

IOBASE\_INIT:

movia r22, TIMER\_BASE /\* base location for timer I/O device \*/

/\*TEMPLATE initialize PIO port registers, if needed \*/

PIODEVICE\_INIT:

movia r20, LEDR\_BASE /\* base location of PIO registers for LED device \*/

movia r23, KEY\_BASE /\* base location for timer KEY device \*/

stwio r0, EDGE\_OFFSET(r23) /\* clear the edge flag \*/

movi r19, 0b01000

stwio r19, MASK\_OFFSET(r23) /\* enable interrupt for key3 \*/

/\*TEMPLATE initialize timer port registers (period and control) if needed \*/

TIMER\_INIT:

movia r13, TIMER\_CONTROL\_VALUE

movi r14, 0x017d /\* the least significant bit for the timer\*/

movia r15, 0x7840 /\* the most significant bit for the timer\*/

stwio r15, PERIODL\_OFFSET(r22) /\* stores most sig period bits into periodH \*/

stwio r14, PERIODH\_OFFSET(r22) /\* stores least sig period bits into periodH \*/

stwio r13, CONTROL\_OFFSET(r22) /\* stores the stores into period H \*/

stwio r0, STATUS\_OFFSET(r22)

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

/\*\*\*\*\*\*\*\*\* Interrupt Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE Setup all I/O ports for interrupts: 1)clear the intr flag \*/

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

SET\_PORT\_INTR:

/\*TEMPLATE Now enable each interrupt in the IENABLE register (one is shown). \*/

SET\_IENABLE:

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

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

ori et, et, KEY\_MASK /\* set the key interrupt enable bit high \*/

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

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

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

SET\_STATUS:

rdctl et, ctl0 /\* Read the status register \*/

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

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

/\*TEMPLATE main program code goes here \*/

MAIN\_PROG\_EXEC:

br EMPTY

EMPTY:

br MAIN\_PROG\_EXEC

PROG\_END:

br PROG\_END /\* useful for final breakpoint \*/

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

/\*\*\*\*\*\*\*\*\* Subroutines \*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE subroutine code goes here if you use any \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variable Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.data

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

COUNTER:

.end

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

/\* Name: Daniel Ackuaku \*/

/\* Course: ENGR \*/

/\* Lab #:6 Part3 \*/

/\* Date: 25th March, 2019 \*/

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

/\*TEMPLATE fill in the title block above \*/

.include "nios\_macros.s"

.include "nios\_iodefs.s" /\* include i/o constants \*/

.include "nios\_irqdefs.s" /\* include interrupt constants \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constant Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE define any main-code specific .equ constants here \*/

.equ TIMER\_CONTROL\_VALUE, 0b0111

.equ BITMASK\_TIMEOUT, 0x01

.equ BITMASK\_KEY\_PRESS\_EDGE, 0b01000

.equ BITMASK\_KEY\_INTERUPT\_CHECK, 0b10

.equ MAX\_COUNTER, 0x40000

.text

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code Section \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org RESET\_VECTOR /\* Place the main routine at the reset address \*/

.global \_start

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

br MAIN\_PROG\_INIT /\* jump over the ISR code \*/

/\* make sure above code fits between the RESET and EXCEPTION addresses!! \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ISR=Interrupt Service Routine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

.org EXCEPTION\_VECTOR

ISR:

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

first be pushed on the stack before doing the work of the ISR. At the

end of the ISR, those same registers must be popped off the stack before

returning from the interrupt (eret) so that the interruptED code is not

affected by the work of the ISR. Do the pushing here. \*/

/\*TEMPLATE push registers if needed \*/

/\*this is where you push registers used in the ISR\*/

addi sp, sp, -0x8

stwio r9, 0(sp)

stwio r13, 4(sp)

/\* NOTE: These three lines of code should NOT be changed! \*/

/\* Check to see if an external intr has occurred (IPENDING). \*/

rdctl et, ctl4 /\* ctl4 = IPENDING \*/

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. \*/

CHECKFOR\_INTR0:

/\*TEMPLATE check if intr 0 needs service by checking bit 0 of IPENDING (ctl4). \*/

rdctl et, ctl4 /\* loads ipending into et\*/

andi et, et, BITMASK\_TIMEOUT /\* perform bit mask \*/

bne et, r0, RESPONDTO\_INTR0 /\* if et is 0 RESPONDTO\_INTR0\*/

br CHECKFOR\_INTR1

RESPONDTO\_INTR0:

/\*TEMPLATE The interrupt 0 action goes here, including clearing the interrupt condition. \*/

movia et, TIMER\_BASE /\* loads the address of TIMER\_BASE into et\*/

/\* clears the flag by setting the Status of TIMER\_BASE to 0\*/

stwio r0, STATUS\_OFFSET(et)

movia et, COUNTER /\* loads the value of the counter into et\*/

ldw r9, 0(et) /\* stores et in r9\*/

addi r9, r9, 1

movia et, MAX\_COUNTER

bltu r9, et, DISPLAY /\* if et(MAX\_COUNTER) = r9 LED\_RESET else DISPLAY\*/

br LED\_RESET

LED\_RESET:

mov r9, r0

DISPLAY:

movia et, LEDR\_BASE /\* loads the address of the LEDR\_BASE into et\*/

stwio r9, DATA\_OFFSET(et) /\* stores r9 into et\*/

/\* updates the COUNTER \*/

movia et, COUNTER

stw r9, DATA\_OFFSET(et)

/\*TEMPLATE if needed, check interrupt 1 for servicing - bit 1 of IPENDING \*/

CHECKFOR\_INTR1:

rdctl et, ctl4 /\* loads ipending into et\*/

andi et, et, BITMASK\_KEY\_INTERUPT\_CHECK /\* perform bit mask \*/

bne et, r0, RESPONDTO\_INTR1 /\* if et is 0 RESPONDTO\_INTR1\*/

br CHECKFOR\_INTR2

/\*TEMPLATE The interrupt 1 action goes here, including clearing the interrupt condition. \*/

RESPONDTO\_INTR1:

movia et, KEY\_BASE /\* loads the address of KEY\_BASE into et\*/

mov r9, r0

stwio r0, EDGE\_OFFSET(et) /\* clear edge flag \*/

movia et, SW\_BASE

ldwio r9, DATA\_OFFSET(et) /\* get switch value \*/

movia et, TIMER\_BASE

stwio r9, PERIODH\_OFFSET(et) /\* new of PERIODH\_OFFSET(TIMER\_BASE) \*/

movia r13, TIMER\_CONTROL\_VALUE

stwio r13, CONTROL\_OFFSET(et) /\* restart timer \*/

END\_ISR:

/\*TEMPLATE If you pushed any registers on the stack at the beginning, pop them off now. \*/

/\*this is where you pop registers used in the ISR\*/

ldwio r9, 0(sp)

ldwio r13, 4(sp)

addi sp, sp, 0x8

eret /\* Return from the exception \*/

/\* end of exception handler routine (ISR) \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

MAIN\_PROG\_INIT: /\* should be the first label after the ISR \*/

movia r20, LEDR\_BASE /\* base location of PIO registers for LED device \*/

movia r22, TIMER\_BASE /\* base location for timer I/O device \*/

movia r25, SW\_BASE /\* base location for timer SW\_BASE \*/

/\*TEMPLATE initialize global variables if needed \*/

GLOBVAR\_INIT:

movia r10, MAX\_COUNTER /\* the limit for the counter i\*/

/\*TEMPLATE init CPU registers for holding I/O port base addresses if needed \*/

IOBASE\_INIT:

movia r22, TIMER\_BASE /\* base location for timer I/O device \*/

/\*TEMPLATE initialize PIO port registers, if needed \*/

PIODEVICE\_INIT:

movia r20, LEDR\_BASE /\* base location of PIO registers for LED device \*/

movia r23, KEY\_BASE /\* base location for timer KEY device \*/

stwio r0, EDGE\_OFFSET(r23) /\* clear the edge flag \*/

movi r19, 0b01000

stwio r19, MASK\_OFFSET(r23) /\* enable interrupt for key3 \*/

/\*TEMPLATE initialize timer port registers (period and control) if needed \*/

TIMER\_INIT:

movia r13, TIMER\_CONTROL\_VALUE

movia r14, 0x0000 /\* the least significant bit for the timer\*/

movia r15, 0x75 /\* the most significant bit for the timer\*/

stwio r15, PERIODL\_OFFSET(r22) /\* stores the most sig period bits into period H \*/

stwio r14, PERIODH\_OFFSET(r22) /\* stores the least sig period bits into period H \*/

stwio r13, CONTROL\_OFFSET(r22) /\* stores the stores into period H \*/

stwio r0, STATUS\_OFFSET(r22)

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

/\*\*\*\*\*\*\*\*\* Interrupt Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*TEMPLATE Setup all I/O ports for interrupts: 1)clear the intr flag \*/

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

SET\_PORT\_INTR:

/\*TEMPLATE Now enable each interrupt in the IENABLE register (one is shown). \*/

SET\_IENABLE:

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

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

ori et, et, KEY\_MASK /\* set the key interrupt enable bit high \*/

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

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

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

SET\_STATUS:

rdctl et, ctl0 /\* Read the status register \*/

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

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

/\*TEMPLATE main program code goes here \*/

MAIN\_PROG\_EXEC:

mov r17, r0

mov r18, r0

movi r18, 0x45

br forLoop

forLoop:

call doSomeWork

call advanceProgressDisplay

addi r17, r17, 1

bne r17, r18, forLoop

call displayProgressComplete

DO\_NOTHING:

br PROG\_END

PROG\_END:

br DO\_NOTHING /\* useful for final breakpoint \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variable Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

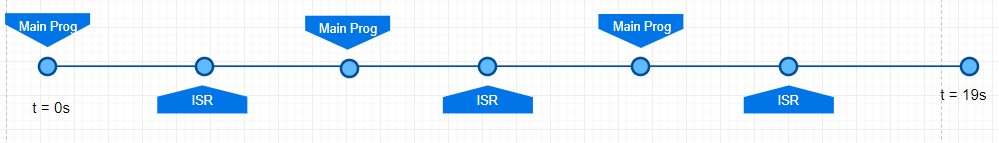
.data

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

COUNTER:

.end

|  |  |  |  |
| --- | --- | --- | --- |
| **PeriodH Value** | **PeriodL Value** | **Period (seconds)** | **Execution Time (seconds)** |
| No interrupts active | | | 8.99 |
| 0x017d | 0x7840 | 0.5 | 9.04 |
| 0x0001 | 0x7840 | 1.67 | 9.09 |
| 0000 | 0x840 | 5.36 | 9.49 |
| 0000 | 0x500 | 1.9 \* 10-3 | 9.49 |
| 0000 | 0x100 | 5.12 \* 10-6 | 11.86 |
| 0000 | 0x75 | 2.34 \* 10-6 | 18.99 |



NB this occurs more times than I was able to represent on the timeline approx. 800000 times

* 1. C. Estimate the execution time of the ISR (the duration of one ISR portion on the timeline).
  2. The ISR part is half of the time line = 9.495 secs.
  4. D. Determine the number of machine code instructions in the ISR and use the answer from Q-C to estimate the amount of time it takes to execute each instruction.

Number of time to execute each instruction:

=