ECE3221 Lab 4 Report

Stephen Cole

3553803

Lab 4 Assembly Code:

.global \_start

\_start: br Start

# Macros

#------------------------------------------------------------------

.macro push rx

addi sp ,sp ,-4

stw \rx ,0(sp)

.endm

.macro pop rx

ldw \rx ,0(sp)

addi sp ,sp ,4

.endm

.EQU RED, 0x10000000

.EQU GREEN, 0x10000010

.EQU segs, 0x10000020

.EQU terminal, 0x10001000

#-------------------------------------------------------------------

# ISR

#-------------------------------------------------------------------

.org 0x0020

push ra

push r3

push r4

# determine source of interrupt

rdctl r3,ipending # r3 = pending interrupt bits

andi r4,r3,0x01 # r4 = pending int0 bit

bne r4,r0,int0 # service int0 if requested

andi r4,r3,0x02 # r4 = pending int1 bit

bne r4,r0,int1 # service int1 if requested

br endint # done

#-------------------------------------------------------------------

# IRQ0 service (interval timer)

int0:

call action0 # provide a specific response to the timer interrupt

# silence the interval timer interrupt request

movia r4,0x10002000 # r4 = interval timer addr

sthio r0,(r4) # set TO=0

br endint # done

#-------------------------------------------------------------------

#-------------------------------------------------------------------

# IRQ1 service (buttons)

int1:

call action1 # Set the green led value to the current red

movia r4,0x10000050 # r4 = button address

movia r3, 0x1

stwio r3,0xC(r4) # set pending bits to 1

br endint # done

#-------------------------------------------------------------------

endint:

pop r4

pop r3

pop ra

addi ea,ea,-4 # ISR done

eret

#-------------------------------------------------------------------

#-------------------------------------------------------------------

.org 0x0100

Start:

movia sp,stacktop # initialize the stack pointer

movia r3, 2

call init

movi r3, 1

br top

display:

push r3

call outhex32

movi r3, ' '

call outchar

movi r3, '='

call outchar

movi r3, ' '

call outchar

pop r3

call outint32

push r3

movi r3, '\n'

call outchar

pop r3

top:

addi r3, r3, 1

call isprime

bne r4, r0, display # display prime numbers

br top

#-------------------------------------------------------------------

# Subroutines

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Outputs to the terminal window a single ASCII character in r3.

# Input: r3

outchar:

push r3

push r5

push r6

movia r5, terminal

checkChar:

ldhio r6, 6(r5) # Check if we can write to the terminal

beq r6, r0, delayChar # if r6 == 0 wait

stbio r3, 0(r5) # write to buffer

endchar:

pop r6

pop r5

pop r3

ret

delayChar:

br checkChar

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Outputs to the terminal window a null-terminated ASCII string at the address provided in r3.

# Input: r3

outstr:

push ra

push r3

push r4

strLoop: # loop through chars in string

ldb r4, (r3)

beq r4, r0, endStr # if r4 == NULL at EOS

addi r3, r3, 1

push r3

mov r3, r4

call outchar # print char

pop r3

br strLoop

endStr:

pop r4

pop r3

pop ra

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Outputs to the terminal window a single ASCII character being the hexadecimal representation of the 4 least significant bits in r3.

# Input: r3

outhex:

push ra

push r3

push r4

andi r3, r3, 0xF # set all but bit0-3 to low

movia r4, lookup

add r3, r3, r4 # search lookup

ldb r3, (r3) # load from lookup

call outchar # print hex

pop r4

pop r3

pop ra

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Outputs to the terminal window ASCII ‘0’,‘x’, plus 8 characters being the hexadecimal representation of the 32-bit contents of r3.

# Input: r3

outhex32:

push ra

push r3

push r4

push r5

movi r4, 8 # check for hex outputs

# print 0x

push r3

movi r3, '0'

call outchar

movi r3, 'x'

call outchar

pop r3

hex32Loop:

roli r3, r3, 4 # take 4 bits and print the hex representation

call outhex

addi r4, r4, -1

beq r4, r0, endhex32 # if there are no values left to print end

br hex32Loop

endhex32:

pop r5

pop r4

pop r3

pop ra

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Outputs to the terminal window up to ten ASCII characters being the decimal representation of the 32-bit contents of r3. Include leading zero-suppression.

# Input: r3

outint32:

outint32:

push ra

push r3

push r4

push r5

push r6

push r7

push r8

push r9

movia r5, 0x5F5E100 # Store 100 million in reg to be our denominator

movia r6, 0xA # Decrease r5 by a zero each loop

movia r7, 0x1

movia r8, lookup

mov r9, r0

outint32Loop:

push r3

# Calculate division result

divu r3, r3, r5

mov r4, r3

# Dont display leading zeros

or r9, r9, r3

beq r9, r0, skipdisp

# Use the hexLUT values to display the decimal values

add r3, r3, r8

ldb r3, (r3)

call outchar

skipdisp:

pop r3

# Calculate remainder

mul r4, r4, r5

sub r3, r3, r4

# If the denominator becomes 1 then end

beq r5, r7, endoutint32

# Remove a zero from denominator

divu r5, r5, r6

br outint32Loop

endoutint32:

pop r9 # Restore regs

pop r8

pop r7

pop r6

pop r5

pop r4

pop r3

pop ra

ret

#-------------------------------------------------------------------

# Stephen Cole - June 12th

# Checks if r3 is divisble by r4

# Returns 1 (false) or 0 (true) in r5.

# Input: r3, r4

# Output r5

isdivisible:

push r6

push r7

# r3 % r4 and if 0 divisible, else not

divu r6, r3, r4

mul r6, r6, r4

sub r7, r3, r6

beq r7, r0, enddivisble # if the remainder is zero it is divisible

notdivisble:

movia r5, 0 # 0 = divisible

pop r7

pop r6

ret

enddivisble:

movia r5, 1 # 1 != divisible

pop r7

pop r6

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Determines if the unsigned integer contents of r3 are a prime number.

# Returns 1 (true) or 0 (false) in r4.

# Input: r3

# Output r4

isprime:

push ra

push r3

push r5

push r6

push r7

push r8

movia r4, 1 # counter

movia r5, 2 # check if 2

beq r3, r5, isprimeEnd

movia r8, 1 # check if 1

beq r8, r3, isNotPrime

movia r5, 3 # check if 3

beq r3, r5, isprimeEnd

andi r8, r3, 1 # check if even

beq r8, r0, isNotPrime

isprimeLoop:

addi r4, r4, 2 # Increment to the next odd number

call isdivisible

bne r5, r0, isNotPrime # If it is divisble then r3 is not prime

mul r8, r4, r4 # Check if N < n^2, if so then end(r3 is prime)

bgt r8, r3, isprimeEnd

br isprimeLoop

isNotPrime:

mov r4, r0 # return 0

pop r8 # pop used registers

pop r7

pop r6

pop r5

pop r3

pop ra

ret

isprimeEnd:

movia r4, 1 # return 1

pop r8 # pop used registers

pop r7

pop r6

pop r5

pop r3

pop ra

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th

# Clears the 7-segment displays, the green LEDs and the red LEDs. Calls setupInterrupts.

init:

push ra

push r3

push r4

movia r3, GREEN

stwio r0, (r3) # Turn off green leds

movia r3, segs

stwio r0, (r3) # clear 7seg

movia r3, RED

movia r4, 0x80000000

stwio r4, (r3) # Turn first red led on

call setupInterrupts

pop r4

pop r3

pop ra

ret

#-------------------------------------------------------------------

# set up for expected interrupts from the interval timer

setupInterrupts:

push ra

push r3

# SETUP INTERRUPTS IN THREE STEPS 1,2,3

# (1) set up a device to generate interrupts

call setupTimer # set up interval timer (IRQ0)

call setupButtons # set up button interrupt (IRQ1)

# (2) set up the processor to acknowledge specific interrupt(s)

rdctl r3,ienable # ienable: enable IRQn by setting Bitn = 1

ori r3,r3,0x03 # set bit0 = 1 for IRQ0 and bit1 =1 for IRQ1

wrctl ienable,r3 # update ienable register

# (3) turn on the processor master interrupt enable

rdctl r3,status # status: various processor control bits

ori r3,r3,0x03 # set PIE bit = 1 (Processor Interrupt Enable)

wrctl status ,r3 # update status register

pop r3

pop ra

ret

#-------------------------------------------------------------------

# setupTimer RTervo May 2020

# set up the DE2-115 interval timer

# to generate an interrupt every 0.1 sec

# no registers affected

setupTimer:

push r3

push r11

movia r11,0x10000000 # r11 = I/O base addr

# the interval counter is clocked at 50 MHz

# r3 = timing interval (clocks) in decimal

movia r3,5000000 # let delay = 0.1 sec

sthio r3,0x2008(r11) # set timer low half word

srli r3,r3,16 # shift to other 16 bits

sthio r3,0x200C(r11) # set timer high half word

sthio r0,0x2000(r11) # TO=0 (clear timeout bit)

ori r3,r0,0x0007 # START=1, CONT=1, ITO=1

sthio r3,0x2004(r11)

pop r11

pop r3

ret

#-------------------------------------------------------------------

# Stephen Cole - June 12th

# Clears 7 segment deisplay, red leds, green leds and calls setup interrupts

setupButtons:

push r3

push r4

movia r4,0x10000050 # r4 = I/O addr

movi r3,1 # Init PB0 interupt enable pin to enabled

stwio r3,0x8(r4)

pop r4

pop r3

ret

#-------------------------------------------------------------------

# Stephen Cole - June 9th - action0

action0:

push r3

push r4

push r5

push r6

movia r3, RED

movia r5, 1

ldwio r4, (r3) # Load red led's into r4

roll:

ror r4, r4, r5 # roll red led to the right

andi r6, r4, 0xFFFF

beq r0, r6, roll # If r4 > 0x8000 roll again

stwio r4, (r3) # display to red leds

pop r6

pop r5

pop r4

pop r3

ret

#-------------------------------------------------------------------

# Stephen Cole - June 12th - action1

action1:

push r3

push r4

push r5

movia r3, RED

movia r4, GREEN

ldwio r5, (r3) # load red leds into r5

stwio r5, (r4) # load r5 into green leds

pop r5

pop r4

pop r3

ret

#-------------------------------------------------------------------

# ALLOCATION

#-------------------------------------------------------------------

.skip 400 # stack = 200 bytes = 50 words

stacktop:

welcome: .asciz " ECE3221 Lab4 \n"

lookup: .ascii "0123456789ABCDEF"

#-------------------------------------------------------------------