# Lab 6 Preparation and Quiz (3 marks)

1. How do you tell the JTAG UART to send read interrupts?

**Enable interrupt on the peripheral. (set CSR.bit0 = 1, ERI for UART)**
**Unmask the IRQ line corresponding to the peripheral on the processor. (set ctl3.bit8 = 1, IRQ8 for UART)**
**Enable interrupts globally in the processor. (set ctl0.bit0 = 1)**

2. How do you tell the timer to send timeout interrupts?

**Enable interrupt on the peripheral. (set timer control register.bit0 = 1)**
**Unmask the IRQ line corresponding to the peripheral on the processor. (set clt3.bit0 = 1, IRQ0 for timer)**
**Enable interrupts globally in the processor. (set ctl0.bit0 = 1)**

3. How do you tell the processor to accept interrupts from both Timer1 and the terminal JTAG UART?

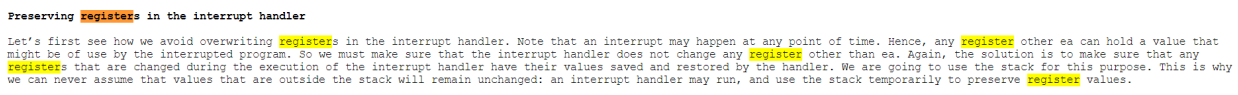
**Unmask the IRQ lines corresponding to the peripherals on the processor. (set clt3.bit0 = 1 and clt3.bit8 = 1; IRQ0 for timer, IRQ8 for UART)**
**Enable interrupts globally in the processor. (set ctl0.bit0 = 1)**

4. Write a line of code to adjust ea before returning from the interrupt handler so that the aborted instruction will be re-executed.

**subi ea, ea, 4**

5. Which registers must you backup before overwriting inside an interrupt handler?

**Whichever registers are used inside the interrupt handler. Anything except ea**



**Preserving registers in the interrupt handler**

Let's first see how we avoid overwriting registers in the interrupt handler. Note that an interrupt may happen at any point of time. Hence, any register other ea can hold a value that might be of use by the interrupted program. So we must make sure that the interrupt handler does not change any register other than ea. Again, the solution is to make sure that any registers that are changed during the execution of the interrupt handler have their values saved and restored by the handler. We are going to use the stack for this purpose. This is why we can never assume that values that are outside the stack will remain unchanged: an interrupt handler may run, and use the stack temporarily to preserve register values.

6. If you want to call a function implemented in C from inside your interrupt handler, which registers must you back up?

**caller-saved registers (r2-r15), and ra, since the interrupt handler is the caller**

7. Write a simple test program to blink an LED with a period of 1 second using timer interrupts.

```
.section .exceptions, "ax"
handler:
addi sp, sp, -8              # save r9/r10
stw r9, 0(sp)
stw r10, 4(sp)
movia r9, 0xFF200000   # leds io addr
ldwio r10, 0(r9)       # read LEDs
```

```
xori r10, r10, 1       # blink LED0
stwio r10, 0(r9)       # store to LEDs
movia r9, 0xFF202000   # timer io addr
stwio r0, 0(r9)              # clear timer
subi ea, ea, 4
ldw r9, 0(sp)                # recover r9/r10
ldw r10, 4(sp)
addi sp, sp, 8
eret

.text
.global _start
_start:
init:
movia sp, 0x17fff80
movia r8, 0xFF202000         # timer IO addr
movi r2, 1
stwio r2, 4(r8)              # enable read interrupt on timer1
wrctl ctl0, r2               # enable interrupts globally
wrctl ctl3, r2               # enable IRQ0 for timer1
movia r3, 100000000          # 1 second with 100MHz clock
srli r4, r3, 16              # upper 16-bits of r3 are in r4
stwio r4, 12(r8)             # store upper bits of period
stwio r3, 8(r8)              # store lower bits of period
ori r2, r2, 0b10
movia r5, 7
stwio r5, 4(r8)              # set timer to count mode, automatic restart

fever:                       # filler code, waiting for interrupt
movia r11, 0xFF202000        # timer io addr
ldwio r13, 16(r11)           # read snapshot timer
br fever
.end
```

8. Write a simple test program to echo characters on the terminal JTAG UART using read interrupts.

```
.equ JTAG_UART_BASE, 0xFF201000
.equ JTAG_UART_RR, 0
.equ JTAG_UART_TR, 0
.equ JTAG_UART_CSR, 4

.global _start
_start:
movia sp, 0x17fff80            # initiate sp
movia r8, JTAG_UART_BASE       # UART IO addr
movi r2, 1
stwio r2, JTAG_UART_CSR(r8)          # enable read interrupt on UART
wrctl ctl0, r2           # enable interrupts globally
movi r2, 0x100
wrctl ctl3, r2           # enable IRQ8 for UART
```

```
fever:
movia r8, JTAG_UART_BASE       # UART IO addr
ldwio r2, JTAG_UART_CSR(r8)    # read CSR
br fever

.section .exceptions, "ax"
handler:
addi sp, sp, -12               # save r9/r10/r11
stw r9, 0(sp)
stw r10, 4(sp)
stw r11, 8(sp)
movia r9, JTAG_UART_BASE
ldwio r10, JTAG_UART_RR(r9)    # read RR
#andi r10, r10, 0xff           # character received, copy lower 8 bits
wait:
ldwio r11, JTAG_UART_CSR(r9)   # read CSR
srli r11, r11, 16              # keep only the upper 16 bits
beq r2, r0, wait               # if upper 16 bits are zero keep trying
stwio r10, JTAG_UART_TR(r9)    # place it in the output FIFO
ldw r9, 0(sp)                  # recover r9/r10
ldw r10, 4(sp)
ldw r11, 8(sp)
addi sp, sp, 12
subi ea, ea, 4                 # make sure we execute instruction interrupted
eret                           # return from interrupt
```

9. Enhance your solution to Lab 5 to display speed and sensor state over the JTAG UART, as described above. Use interrupts both to read from the terminal JTAG UART and to check the Timer for a timeout.

```
.equ JTAG_UART_BASE, 0xFF201000
.equ JTAG_UART_RR, 0
.equ JTAG_UART_TR, 0
.equ JTAG_UART_CSR, 4
.equ TIMER1_BASE, 0xFF202000
.equ TIMER1_SR, 0
.equ TIMER1_CR, 4

.section .reset, "ax"
movia sp, 0x17fff80 /* initialize stack */
movia ra, _start
ret

.section .exceptions, "ax"
.align 2

handler:
# PROLOGUE
subi  sp, sp, 12               # we will be saving two registers on the stack
stw   r9, 0(sp)                # save r9
stw   r10, 4(sp)               # save r10
stw   r11, 8(sp)               # save r11
```

```
movia r9, JTAG_UART_BASE
ldwio r10, JTAG_UART_CSR(r9)   # read the status register
andi  r10, r10, 0x100          # is 8th bit 1 because of UART interrupt?
beq   r10, r0, TIMER_INT       # if not then it must be the timer

UART_INT:                      # changes modes between "s" and "r" via r19
movia r9, JTAG_UART_BASE
ldwio r10, JTAG_UART_RR(r9)   # read RR
andi r10, r10, 0xff            # character received, copy lower 8 bits
wait:
ldwio r11, JTAG_UART_CSR(r9)  # read CSR
srli r11, r11, 16              # keep only the upper 16 bits
beq r2, r0, wait               # if upper 16 bits are zero keep trying
movia r11, 0x73
beq r10, r11, S_MODE
R_MODE:
movia r19, 1
br epilogue                    # done, go to the epilogue
S_MODE:
movia r19, 0
br epilogue                    # done, go to the epilogue


TIMER_INT:               # gets reading, displays it, r3 = sensor, r6 = speed
movia r9, TIMER1_BASE          # timer io addr
stwio r0, 0(r9)                # clear timer
movia r9, JTAG_UART_BASE
movia r11, 0x1b
stwio r11, 0(r9)
movia r11, 0x5b
stwio r11, 0(r9)
movia r11, 0x32
stwio r11, 0(r9)
movia r11, 0x4b
stwio r11, 0(r9)                     # clear line
movia r11, 0x1b
stwio r11, 0(r9)
movia r11, 0x5b
stwio r11, 0(r9)
movia r11, 0x48
stwio r11, 0(r9)                     # move cursor home
bne r0, r19, SENSOR_MODE       # check mode and go to appropriate place
br SPEED_MODE

SENSOR_MODE:                         # r19 = 0, display r3
movia r9, JTAG_UART_BASE # r9 now contains the base address of other JTAG

CONVERSION:
add r10, r3, r0
Srli r10, r10, 4
Movia r11, 0xF
# top 4 bits of input into r10
And r10, r10, r11
Movia r11, 0xA
Blt r10, r11, number2
Br letter2
```

```
number2:
Addi r10, r10, 0x30
Br display2
letter2:
subi r10, r10, 9
Addi r10, r10, 0x40
Br display2
display2:
stwio r10, 0(r9)              # Write top 4 bits of r10 to the JTAG

# bottom 4 bits of input into r10
Movia r11, 0xF
And r10, r3, r11
Movia r11, 0xA
Blt r10, r11, number
Br letter

number:
Addi r10, r10, 0x30
Br display
letter:
subi r10, r10, 9
Addi r10, r10, 0x40
Br display
display:
stwio r10, 0(r9)              # Write bottom 4 bits of r10 to the JTAG

br epilogue

SPEED_MODE:                         # r19 = 1, display r6
movia r9, JTAG_UART_BASE      # r9 now contains the base address of other JTAG

CONVERSION2:
add r10, r6, r0
Srli r10, r10, 4
Movia r11, 0xF
# top 4 bits of input into r10
And r10, r10, r11
Movia r11, 0xA
Blt r10, r11, number4
Br letter4

number4:
Addi r10, r10, 0x30
Br display4
letter4:
subi r10, r10, 9
Addi r10, r10, 0x40
Br display4
display4:
stwio r10, 0(r9)              # Write top 4 bits of r10 to the JTAG

# bottom 4 bits of input into r10
Movia r11, 0xF
And r10, r6, r11
Movia r11, 0xA
Blt r10, r11, number3
```

```
Br letter3

number3:
Addi r10, r10, 0x30
Br display3
letter3:
subi r10, r10, 9
Addi r10, r10, 0x40
Br display3
display3:
stwio r10, 0(r9)              # Write bottom 4 bits of r10 to the JTAG

br epilogue

epilogue:               # EPILOGUE
ldw   r9, 0(sp)               # restore r5
ldw   r10, 4(sp)              # restore r2
ldw   r11, 8(sp)              # restore r2
addi  sp, sp, 12              # we will be saving two registers on the stack
subi  ea, ea, 4               # make sure we execute the instruction that was
interrupted
eret                         # return from interrupt

.text
.global _start
_start:
movia sp, 0x17ff80            # initiate sp

# interrupt setup
movia r8, JTAG_UART_BASE            # UART IO addr
movia r10, TIMER1_BASE              # timer IO addr
movi r2, 1
stwio r2, JTAG_UART_CSR(r8)         # enable read interrupt on UART
stwio r2, TIMER1_CR(r10)       # enable read interrupt on timer
movi r2, 0x101                      # bits for IRQ0 and IRQ8
wrctl ctl3, r2                      # enable and IRQ0 IRQ8 for timer and UART
wrctl ctl0, r2                      # enable interrupts globally

# timer setup
movia r3, 100000000                 # 1 second with 100MHz clock
srli r4, r3, 16                     # upper 16-bits of r3 are in r4
stwio r4, 12(r10)                   # store upper bits of period
stwio r3, 8(r10)                    # store lower bits of period
movia r5, 7
stwio r5, 4(r10)              # set timer to count mode, automatic restart

lab5:
# copy code from previous lab here
/*
Register allocation:
r2 - used to help get sensor reading in r3
r3 - sensor reading
r4 - parameter into ACC_CONTROL, sets acceleration to that value
r5 - parameter into STEER_CONTROL, sets steering to that value
r6 - speed reading
r7 - used to store the UART address
r8 - used to request a sensor reading by writing 0x2 to UART
```

```
r9 - used to request a sensor reading by writing 0x2 to UART
r10 - used for comparing steering values
r11 - error register (if there is a weird sensor reading)
r16 - callee saved registers used in helper functions
r17 - callee saved registers used in helper functions
r18 - callee saved registers used in helper functions
r19 - global variable, display mode for lab 6. 0 = sensor, 1 = speed
*/
movia sp, 0x17fff80            # initialise stack pointer
movia r2, 0x0                  # reset all regs
movia r3, 0x0
movia r4, 0x0
movia r5, 0x0
movia r7, 0x10001020
movia r8, 0x0
movia r9, 0x0
movia r10, 0x0
movia r11, 0x0
movia r16, 0x0
movia r17, 0x0
movia r18, 0x0

movi r5, 0x00
call STEER
call WORLDRESET                # reset UART

movi r4, 0x7F                  # high initial acceleration
call ACC_CONTROL               # set an initial acceleration until car
encounters first corner

# Main Function Loop
ReadSensorsAndSpeed:

# Request a reading
MAIN_WRITE_POLL:
ldwio r8, 4(r7) /* Load from the JTAG */
srli  r8, r8, 16 /* Check only the write available bits */
beq   r8, r0, MAIN_WRITE_POLL /* If this is 0 (branch true), data cannot be
sent */
movui r9, 0x2 /* packet type is 2 (sensor)*/
stwio r9, 0(r7) /* Write the byte to the JTAG */

# Parse the reading
MAIN_READ_POLL:
ldwio r2, 0(r7) /* Load from the JTAG */
andi  r3, r2, 0x8000 /* Mask other bits */
beq   r3, r0, MAIN_READ_POLL /* If this is 0 (branch true), data is not valid
*/
andi  r3, r2, 0x00FF /* Data read is now in r3 */

MAIN_READ_POLL2:
ldwio r2, 0(r7) /* Load from the JTAG */
andi  r6, r2, 0x8000 /* Mask other bits */
beq   r6, r0, MAIN_READ_POLL2 /* If this is 0 (branch true), data is not valid
*/
andi  r6, r2, 0x00FF /* Data read is now in r6 */
```

```
# Compare scenarios
movi r10, 0x1f                      # if sen = 0x1f, br all on track
beq r3, r10, AllOnTrack

movi r10, 0x1e                      # if sen = 0x1e, br left on track
beq r3, r10, LeftOffTrack

movi r10, 0x1c                      # if sen = 0x1c, br two left off track
beq r3, r10, TwoLeftOffTrack

movi r10, 0x0f                      # if sen = 0x0f, br right off track
beq r3, r10, RightOffTrack

movi r10, 0x07                      # if sen = 0x07, br two right off track
beq r3, r10, TwoRightOffTrack

br ReadSensorsAndSpeed

# Scenarios
AllOnTrack: #  sensors are 0x1f = all sensors on track, steer straight
movi r5, 0x00
call STEER
movi r20, 30
bgt r6, r20, OVER
movi r20, 20
blt r6, r20, BELOW
movi r4, 100 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

LeftOffTrack: # sensors are 0x1e = leftmost sensor off track, turn right by
sending 0x32 = 50 to steering
movi r5, 50
call STEER
movi r20, 30
bgt r6, r20, OVER
movi r20, 20
blt r6, r20, BELOW
movi r4, -60 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

TwoLeftOffTrack:# sensors are 0x1c = 2 leftmost sensors off track, turn hard
right by sending 0x64 = 100 to steering
movi r5, 127
call STEER
movi r20, 30
bgt r6, r20, OVER
movi r20, 20
blt r6, r20, BELOW
movi r4, -127 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

RightOffTrack:# sensors are 0x0f = rightmost sensor off track, turn left by
sending 0xCE = -50 to steering
movi r5, -50
```

```
call STEER
movi r20, 30
bgt r6, r20, OVER
movi r20, 20
blt r6, r20, BELOW
movi r4, -60 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

TwoRightOffTrack:# sensors are 0x07 = 2 rightmost sensors off track, turn hard
left by sending 0x9C = -100 to steering
movi r5, -127
call STEER
movi r20, 30
bgt r6, r20, OVER
movi r20, 20
blt r6, r20, BELOW
movi r4, -127 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

BELOW:
movi r4, 127 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed

OVER:
movi r4, -80 # decelerate to keep constant speed after first corner
call ACC_CONTROL
br ReadSensorsAndSpeed
# Helper functions

WORLDRESET: # writes 0x00 (op-stop) and drains reads. Uses callee saved
registers r16-r18
movia r16, 0x10001020 /* r16 now contains the base address of UART*/
RESET_WRITE_POLL:
ldwio r17, 4(r16) /* Load from the JTAG */
srli  r17, r17, 16 /* Check only the write available bits */
beq   r17, r0, RESET_WRITE_POLL /* If this is 0 (branch true), data cannot be
sent */
stwio r0, 0(r16) /* Write the byte 0x00 to the JTAG canceling the wait for new
packet*/
RESET_READ_POLL:
ldwio r17, 0(r16) /* Load from the JTAG */
andi  r18, r17, 0x8000 /* Mask other bits */
bne   r18, r0, RESET_READ_POLL /* If this is 1, data is valid, so continue
draining */
ret

STEER: # write r5 into packet type 5, uses callee saved registers r16-r19
addi sp, sp, -12
stw r16, 0(sp)
stw r17, 4(sp)
stw r18, 8(sp)
movia r16, 0x10001020 /* r16 now contains the base address of UART*/
STEER_WRITE_POLL:
ldwio r17, 4(r16) /* Load from the JTAG */
```

```
srli  r17, r17, 16 /* Check only the write available bits */
beq   r17, r0, STEER_WRITE_POLL /* If this is 0 (branch true), data cannot be
sent */
movui r18, 0x5 /* packet type is 5 (steering)*/
stwio r18, 0(r16) /* Write the byte to the JTAG */
STEER_WRITE_POLL2:
ldwio r17, 4(r16) /* Load from the JTAG */
srli  r17, r17, 16 /* Check only the write available bits */
beq   r17, r0, STEER_WRITE_POLL2 /* If this is 0 (branch true), data cannot be
sent */
stwio r5, 0(r16) /* Write the steering value in r5 to the JTAG */
ldw r16, 0(sp)
ldw r17, 4(sp)
ldw r18, 8(sp)
addi sp, sp, 12
ret

ACC_CONTROL: # write r4 into packet type 4 uses callee saved registers r16-r18
addi sp, sp, -12
stw r16, 0(sp)
stw r17, 4(sp)
stw r18, 8(sp)
movia r16, 0x10001020 /* r16 now contains the base address */
ACC_WRITE_POLL:
ldwio r17, 4(r16) /* Load from the JTAG */
srli  r17, r17, 16 /* Check only the write available bits */
beq   r17, r0, ACC_WRITE_POLL /* If this is 0 (branch true), data cannot be
sent */
movui r18, 0x4 /* packet type is 4 (acceleration)*/
stwio r18, 0(r16) /* Write the byte to the JTAG */
ACC_WRITE_POLL2:
ldwio r17, 4(r16) /* Load from the JTAG */
srli  r17, r17, 16 /* Check only the write available bits */
beq   r17, r0, ACC_WRITE_POLL2 /* If this is 0 (branch true), data cannot be
sent */
stwio r4, 0(r16) /* Write the acceleration value to the JTAG */
ldw r16, 0(sp)
ldw r17, 4(sp)
ldw r18, 8(sp)
addi sp, sp, 12
ret
```