ECE 375 Homework 3

Computer Organization and Assembly Language Programming

Winter Term 2018

Jeremy Fischer

1 Homework Questions

1. The AVR code below (with some information missing) is designed to initialize and service interrupts from three I/O devices (DevA, DevB, and DevC).

   a There are 8 external interrupt pins (INT0-INT7) in AVR. Which of the three interrupt pins are these I/O devices connected to and what is the immediate value needed in line (1)?

   Answer: The three I/O devices are connnected to INT0, INT3, and INT5. The immediate value needed in line (1) is: 0b00101001.

   b Which I/O device's interrupt is detected on a falling edge?

   Answer: INT3 is detected on a falling edge.

   c The interrupt pins referred to in part (a) are connected to two of the 7 ports (PORTA-PORTG) in AVR. Which ports are they?

   Answer: INT0 and INT3 are connected to PORTD and INT5 is connected to PORTE.

   d There are important instructions missing in lines 2-3 of the code. Fill in the missing instructions in lines 2-3 so that the code will work correctly.

   Answer: Shown Below.

   e Suppose DevA requires that no interrupts are detected while it is being serviced. Fill in lines 4-5 with the necessary code to clear any latched interrupts at the end of ISR_DevA.

   Answer: Shown Below.

   f Suppose interrupts are detected from all three interrupt pins at the same time. Which subroutine (ISR_DevA, ISR_DevB, or ISR_DevC) will be executed first?

   Answer: INT0 would be triggered, meaning that ISR_DevA would be called.

```
.include "m128def.inc"
.def mpr = r16

START:
.org $0000
   RJMP INIT
.org $0002
   RJMP ISR_DevA
.org $0008
   RJMP ISR_DevB
.org $000C
   RJMP ISR_DevC

INIT:
   ldi mpr, 0b10000011
   sts EICRA, mpr
   ldi mpr, 0b00001100
   out EICRB, mpr
   ldi mpr, 0b00101001   ;(1)
```

```
    out EIMSK, mpr
    ldi mpr, $00
    out DDRD, mpr
    out DDRE, mpr          ;(2)
    ldi mpr,0b00001000
    out PORTD, mpr         ;(3)
    sei
    ...
MAIN:
    {...
    ...do something...
    ...}
ISR_DevA:
    ...
    ...
    sbr mpr, (1<<0)|(1<<3)|(1<<5)      ;(4)
    out EIFR, mpr                      ;(5)
    RETI
ISR_DevB:
    {...
    ...
    RETI
    }
ISR_DevC:
    {...
    ...
    RETI
    }
```

2. Consider the WAIT subroutine for Tekbot discussed in class that waits for 1 sec. and returns. Rewrite the WAIT subroutine so that it waits for 1 sec. using the 16-bit Timer/Counter1 with the highest resolution. Assume that the system clock frequency is 16 MHz and the Timer/Counter1 is operating under the CTC mode. This is done by doing the following:

   (a) Timer/Counter1 is initialized to operate in the CTC mode.

   (b) The WAIT subroutine loads the proper value into OCR1A and waits until OCF1 is set. Once OCF1 is set, it is cleared and the WAIT subroutine returns.

   Use the skeleton code shown below. Also, show the necessary calculations for determining *value* and *prescale*. Note that your code may not use any other GPRs besides mpr.

   Answer:

```
.include "m128def.inc"
.def mpr = mpr

.ORG $0000
   RJMP INIT
.ORG $0046                    ;End of interrupt vectors
INIT:
   ;Initialize Stack Pointer
   ldi mpr, low(RAMEND)       ;Low Byte of End SRAM Address
   out SPL, mpr               ;Write byte to SPL
   ldi mpr, high(RAMEND)      ;High Byte of End SRAM Address
   out SPH, mpr               ;Write byte to SPH

   ;clearOC1A on compare match. pre=256, mode=CTC
   ldi mpr, (1<<COM1A1)|(1<<WGM12)|(1<<CS12)
   out TCCR1B, mpr            ;timer clock = system clock/256

   ldi mpr, (0<<WGM11)|(0<<WGM10)
   out TCCR1A, mpr            ;CTC mode lower 2-bits

WAIT:
   ldi mpr, 0b00100011        ;OCR1A holds TOP = 62499
   out OCR1AL, mpr
   ldi mpr, 0b11110100
   out OCR1AH, mpr
   ldi mpr, $00               ;set counter to 0
   out TCNT1L, mpr
   out TCNT1H, mpr

   WAITLOOP:
      in mpr, TIFR            ;Read in OCF1A in TIFR
      cpi mpr, (1<<OCF1A)
      brne WAITLOOP           ;OCF1A not set? Loop again
      ldi mpr, (1<<OCF1A)     ;Clear OCF1A by writing one to it
```

```
    out TIFR, mpr

RET
```

3. Consider the following AVR assembly code that waits for 1 sec. using the 8-bit Timer/-Counter0 with the system clock frequency of 16 MHz operating under Normal mode. Write and explain the instructions in lines (1-10) necessary to make this code work properly. More specifically,

   a Fill in lines (1-2) with the necessary code to enable the interrupt on Timer/Counter0 overflow.

   b Fill in lines (3-4) with the necessary code to load the value for delay that generates an interrupt after 5 ms.

   c Fill in lines (5-6) with the necessary code to set the prescalar value and the mode of operation.

   d Fill in line (7) with the necessary code to set counter.

   e Fill in lines (8-9) with the necessary code to reload the value for delay.

   f Fill in line (10) with the necessary code to decrement counter.

   Answer:

```
.include "m128def.inc"
.def mpr = r16
.def counter = r17

.ORG $0000
   RJMP Initialize
.ORG $0020              ;Timer/Counter0 overflow interrupt vector
   JMP Reload_counter
.ORG $0046              ;End of interrupt vectors
Initialize:
   ldi mpr, (1<<TOIE0) ;(1) ;Enable interrupt on Timer/Counter0 overflow
   out TIMSK, mpr       ;(2)
   sei                  ;Enable global interrupt
   ldi mpr, 178         ;(3) ;Load the value for delay
   out TCNT0, mpr       ;(4)
   ldi mpr, 0b00000111 ;(5) ;prescalar = 1024, mode = normal
   out TCCR0, mpr       ;(6)
   ldi counter, 200     ;(7) ;Set counter 200*.005=1sec.
LOOP:
   cpi counter, 0       ;Repeat until interrupted
   brne LOOP

Reload_counter:
   push mpr            ;Save mpr
   ldi mpr, 178         ;(8) ;Load the value for delay
   out TCNT0, mpr       ;(9)
   dec counter          ;(10);Decrement counter

   pop mpr             ;Restore mpr
```

reti

4. Consider the AVR code segment shown below that performs initialization and receive routines for USART1 to continuously receives data from a transmitter. Write and explain the instructions in lines (1-16) necessary to make this code work properly. More specifically,

   a Fill in lines (1-4) with the necessary code to initialize the stack pointer.

   b Fill in lines (5-6) with the necessary code to configure RXD1 (Port D, pin 2).

   c Fill in lines (7-10) with the necessary code to set the Baud rate at 9,600 bps using double data rate.

   d Fill in lines (11-12) with the necessary code to enable the receiver and the Receive Complete interrupt.

   e Fill in lines (13-14) to set the frame format to be asynchronous with 8 data bits, 2 stop bits, and even parity.

   f Fill in lines (15-16) to receive data and store it into Buffer.

   Assume the system clock frequency of 16 MHz.

   Answer:

```
.include "m128def.inc"
.def mpr = r16
.ORG $0000
   RJMP initUSART1;
.ORG $003C
   RCALL RECEIVE_DATA;
   RETI
.ORG $0046

initUSART1:
   ;Initialize Stack Pointer
   ldi mpr, low(RAMEND) ;(1) ;Low Byte of End SRAM Address
   out SPL, mpr         ;(2) ;Write byte to SPL
   ldi mpr, high(RAMEND);(3) ;High Byte of End SRAM Address
   out SPH, mpr         ;(4) ;Write byte to SPH

   ;Configure Port D, pin 2
   ldi mpr, (0<<PD2)    ;(5) ;Set RXD1 to be input
   out DDRD, mpr        ;(6)

   ;Set baud rate at 9600bps, double data rate
   ldi mpr, (1<<U2X1)   ;(7) ;Set double data rate on USART1
   sts UCSR1A, mpr      ;(8) ;UCSR1A in extended I/O space
   ldi mpr, 207         ;(9) ;UBRR = 207
   sts UBRR1L, mpr      ;(10);UBRR1L in extended I/O space

   ;Enable receiver and RX Complete interrupt
   ldi mpr, (1<<RXEN1)|(1<<RXCIE1) ;(11) ;Rec Enable and Rec Inrpt Enable
   sts UCSR1B, mpr     ;(12) ;UCSR1B in extended I/O space
```

```
    ;Set frame format: 8 data, 2 stop bits, asynchronous, even parity
    ldi mpr, (1<<USBS1)|(1<<UPM11)(1<<UCSZ11)|(1<<UCSZ10);(13)
    sts UCSR1C, mpr      ;(14)

    sei

MAIN:
    LDI XH, high(Buffer)
    LDI XL, low(Buffer)
LOOP:
    RJMP LOOP ; Wait for data
    ...
RECEIVE_DATA:
    push mpr ; Save mpr
    ;Receive data and store in buffer
    ldi mpr, UDR1  ;(15) mpr = pointer to recieved data
    st X+, mpr     ;(16) X = *mpr, X++

    pop mpr ; Restore mpr
    ret ;Return
Buffer:
    .BYTE 100 ;100 byte buffer
```