Microprocessor Lab --- Interrupts

Use Interrupts to Perform Simultaneous Tasks

1. Equipment

- M68HC12EVB
- Oscilloscope
- Personal Computer

2. Introduction

Interrupts are a very useful tool which allows the processor to perform multiple tasks simultaneously. An interrupt is typically generated by hardware such as a UART, timer, or GPIO pin. When an interrupt is triggered, the main program is halted and the processor examines the Interrupt Vector Table to find the memory location of the Interrupt Service Routine. Each interrupt has a specific location in the IVT. Once the location of the ISR is determined, the processor will jump to that location. An ISR is a lot like a regular subroutine, except that context switching is implemented by the processor when entering and leaving the ISR. Context switching consists of pushing all the registers onto the stack, and popping them once the ISR is complete. This provides you with a "virtual processor" to run the ISR without causing any changes to registers used by the main program. Remember that execution of the ISR can happen at any time, so take appropriate steps to ensure that unexpected behavior does not occur. Also be aware that you cannot use registers to communicate between the ISR and main program.

3. Procedure

1. Run the following program, which will toggle the pins in port T and echo characters to the serial terminal simultaneously.

```
REGBS       EQU   $0000    ; DP256 register bank base address
PTT:        EQU   REGBS+$240 ;portT data register
DDRT:       EQU   REGBS+$242 ;portT direction register
TIOS:       EQU   REGBS+$40 ;timer input/output select
TSCR:       EQU   REGBS+$46 ;timer system control register
TMSK1:      EQU   REGBS+$4C ;timer interrupt mask 1
TMSK2:      EQU   REGBS+$4D ;timer interrupt mask 2
TFLG1:      EQU   REGBS+$4E ;timer flags 1
TC7:        EQU   REGBS+$5E ;timer capture/compare register 7
SCOSR       EQU   $00CC
SCODR       EQU   $00cf
```

These lines define the memory locations of hardware we will be using in this program. None of this functionality has been covered yet, so you do not need to be very concerned with exactly what is going on.

```
            org   $4000 ;main program
```

Note that MON12 requires that an ISR be located in the address space between $4000 and $7FFF. Our program is very short, so we can be sure the ISR defined below will be in this space.

```
            movw  #tc7_isr,$3FE0 ;setup vector table, timer channel 7
```

Here we are placing the address of the ISR into the interrupt vector table. In our system, we are using MON12's RAM Interrupt Vector Table. Note that typically the IVT is located in ROM along with your program. The timer channel 7 interrupt vector is located at $FFE0 in the HC12 ROM IVT. In practice, the IVT would be defined the same was as a constant array in program memory.

```
            movb   #$FF,DDRT
```

This line sets all pins in port T to output. We will cover this in a later lab.

```
            movb   #$80,TSCR ;enable timer, generate 25ms periodic interrupt
            clr    TMSK1
            movb   #$80,TIOS
            movb   #$2B,TMSK2
            ldd    #25000 ;25 ms delay
            std    TC7
```

This block of code sets up the timer module to generate a periodic interrupt every 25ms. Again, we have not covered this functionality, so consider the timer module a "black box" where you are only concerned with what it does, not how it works. We will cover this in a later lab.

```
            bset   TMSK1,#$80; enable interrupt
            cli
```

The first opcode sets the bit which enables the interrupt for timer channel 7. The second opcode globally enables interrupts.

```
getchar     BRCLR  SCOSR, #$20, * ;wait for RDRF bit to be set
            LDAA   SCODR          ;read char in SCODR into A
            BRCLR  SCOSR, #$80, *
            staa   SCODR
            bra    getchar
```

This infinite loop is the main program that will echo characters sent to the microcontroller by Miniide's serial terminal. This is done to illustrate that multiple tasks can be accomplished simultaneously. It works by waiting for the UART to receive a character, storing that character in A, waiting to be sure the UART is done transmitting, then transmitting that character by writing A to the data register. While all of this is happening, the pins in port T are being toggled every 25ms by the ISR.

```
tc7_isr     bclr   TFLG1,$7F ;clear c7f flag

            ldab   PTT ;toggle port t
            eorb   #$FF
            stab   PTT
            rti
```

The first thing the ISR must do is clear the interrupt flag. If this is not done, the ISR will keep executing and crash the program. Note that we do not have to take any steps to determine the cause of the timer channel 7 interrupt because we have only enabled one way to trigger the interrupt. The rest of the code uses the exclusive or operator to toggle all of the bits in the port T register. Remember to use rti instead of rts for interrupt service routines.

2. With the program running, attach the oscilloscope between ground and any of the pins in port T. You should see a square wave with a period of 50ms. Type some characters to the serial terminal and observe that they are being echoed back by the microcontroller.

3. Modify the program so it will run for a period of 10 seconds then exit. You should do this by incrementing or decrementing a word variable in the ISR, then breaking out of the getchar loop once the time has elapsed. Most of the time in the getchar process is spent in the

first line, which loops waiting for a character to arrive. Checking of the counter variable should be implemented inside of this loop. Note that 25ms is 1/40 of a second.

4. Lab report

Include your program and execution results, sketch the waveform you obtained from port T.

```
REGBS        EQU   $0000    ; DP256 register bank base address
PTT:         EQU   REGBS+$240 ;portT data register
DDRT:        EQU   REGBS+$242 ;portT direction register
TIOS:        EQU   REGBS+$40 ;timer input/output select
TSCR:        EQU   REGBS+$46 ;timer system control register
TMSK1:       EQU   REGBS+$4C ;timer interrupt mask 1
TMSK2:       EQU   REGBS+$4D ;timer interrupt mask 2
TFLG1:       EQU   REGBS+$4E ;timer flags 1
TC7:         EQU   REGBS+$5E ;timer capture/compare register 7
SCOSR        EQU   $00CC
SCODR        EQU   $00cf

             org   $4000 ;main program
             movw  #tc7_isr,$3FE0 ;setup vector table, timer channel 7

             movb  #$FF,DDRT

             movb  #$80,TSCR ;enable timer, generate 25ms periodic interrupt
             clr   TMSK1
             movb  #$80,TIOS
             movb  #$2B,TMSK2
             ldd   #25000 ;25 ms delay
             std   TC7

             bset  TMSK1,#$80; enable interrupt
             cli


getchar      BRCLR SCOSR, #$20, * ;wait for RDRF bit to be set
             LDAA  SCODR          ;read char in SCODR into A
             BRCLR SCOSR, #$80, *
             staa  SCODR
             bra   getchar

tc7_isr      bclr  TFLG1,$7F ;clear c7f flag

             ldab  PTT ;toggle port t
             eorb  #$FF
             stab  PTT
             rti
```