1.)

	Status						IntCtl	Cause	INTCON	Ebase	OFF009	OFF014
	BEV	IPL	UM	ERL	EXL	IE	VS	IV	MVEC			
After Power On	1	0000000	1	1	1	0	0000	1	0	800000000000	0x9CE3	0x157AE
At beginning of main	0	0000000	0	0	0	0	1	1	0	9D000000000	0x200	0x200
No defined reset value		u	u	u	u	u						

e) The ERL bit in Status will appear to have a POR value of 1 when the manual says it doesn't have a POR value. Attempt to explain the discrepancy.

The ERL is being set to 1 because we are doing a power on reset which is one of the things that sets the ERL bit.

f) Based on your "After Power On" values above, what core interrupt mode (not offset mode (computed or variable) or operating mode (user / kernel / debug) or single vs multi-vector mode) is the **processor core** in immediately after power on? What bit values justify this conclusion?

The core interrupt mode is Interrupt Compatibility Mode since the IntCtl.VS value is 0.

g) Based on your "After Power On" values above, is the processor in Kernel or User mode immediately after power on? What bit values justify this conclusion?

The processor is in Kernel mode since the ERL and the EXL bits are both 1 since only one of them needs to be a 1 to be in Kernel mode. When the ERL bit is set the processor is running in Kernel mode and interrupts are disabled.

h) What core interrupt mode is the **processor core** in at the beginning of main after crt.o runs? What bit values justify this conclusion?

The core interrupt mode is EIC mode since the IntCtl.VS bit is not 0.

i) Is the processor in Kernel or User mode immediately after crt.o runs? What bit values justify this conclusion?

The processor is running in Kernel mode since the UM value is 0. Had it been a 1 with the EXL and ERL bits being 0 it would have been in user mode.

j) What is the processor priority at the beginning of main? Why is this important to know?

The processor priority at the beginning of main is 0 since the IPL is set to 0. It is important to know your priority starting point so that you can properly implement interrupts because a process will only be interrupted when the priority value is higher than the IPL.

k) What, beyond what has already been done by crt.o, will need to be done in main to enable interrupts to run in single vector mode?

First, we need to enable interrupts by setting the IE bit to 1. And since the INTCON MVEC bit is defaulted to 0 for single vector mode.

l) What, beyond what has already been done by crt.o, will need to be done in main to enable interrupts to run in multi-vector mode?

First, we need to enable interrupts by setting the IE bit to 1. And set the INTCON MVEC bit to 1 for multi-vector mode.

m) What was responsible for changing the values of OFF009 and OFF0014?

Offsets are held in the EIC and the EIC provides the Offset to the MIPS core which adds it to Ebase.

3.)

Tmr2 count 1st time = 83

Every time after = 28

The first time after reset we had nothing in the cache after running the routine the first time the routine and data is in the cache which saves time.

4.)

OFF009 = 0x00000200

Ebase = 0x9D000000

EBase + OFFx = 9D000200

RDPGPR SP, SP

It has no label

5.)

9D00_0294

_DefaultInterrupt

All other OFFx = 0x294

The _DefaultInterrupt will cause a debug breakpoint and reset the device.

The programmer could forget to provide a specific handler so that the _DefaultInterrupt is used.

6.)

Tmr2 at interrupt vector breakpoint = 17

The time is less because we only went to the entry point of the ISR so we didn't do all the prologue stuff.

Second time = 13

Again the first time the information is not in cache but the second tie it is which saves times.

7.)

The opening curly brace is set at the beginning of the prologue. The closing curly brace is set at the start of the epilogue.

8.)

V1 and V0 and FP registers are saved.

9.)

Tmr2 first run = 124

Eret = 124 – 86 = 38

Tmr2 second run = 35

Eret = 50 – 35 = 15

10.)

3.)

Tmr2 count 1st time = 71

Tmr2 count 2nd time = 23

6.)

Tmr2 count 1st time = 17

Tmr2 count 2nd time = 13

8.)

None of the general purpose registers are being saved since we are using shadow set registers.

It is going faster because we do not need to save the general purpose registers since we are using shadow set registers. It may go up a step or two for step 6 because we need to put the information into the shadow set registers.

12.)

OFF014 = 0x294

OFF009 = 0x200

13.)

I expect the tmr2 tmr3 tmr2 tmr2 tmr3 tmr2 tmr2 tmr3.

It appears this way because tmr3 is being interrupted by tmr2.

16.)

Tmr2 1st time = 67

Tmr2 2nd time = 76

Tmr2 3rd time = 28

Tmr2 4th time = 52

Tmr2 5th time = 28

Tmr2 6th time = 28

Tmr2 7th time = 28

Tmr2 8th time = 28

Tmr2 9th time = 28

Tmr2 10th time = 28

a.)

Because we have already ran ISR for timer 3 which uses the same stack as timer 2 ISR so in the prologue we get cache hits that saves some time.

b.)

They are taking longer because it is happening during the timer 3 ISR which no cannot be interrupted by the timer 2 ISR and timer 3 ISR is taking longer because of the while loop.

c.)

Because the difference between the service routines is about 24 since one is executed once every 1000 and one is twice every 1024.

17.)

Tmr2 interrupt should be nesting in Tmr3.

I made a watch for the SP so that I could see when it changes to account for the nesting.

18.)

a.) Without RIPL you do not know what the priority is in single vector mode.

b.) The prologue saves the register information for you and in C local variables create their own version each time so complies with the need to be re-entrant.

c.) Since the var is static it will not be creating its own version of that variable so it is not re-entrant. This could cause it to mistake which interrupt occurred.

```
/*******************************************************************************
 * Author: Chris Thomas
 * Date: 10/18/2022
 * Project: Lab 3 - Interrupts in C
 * Updates: N/A
 ******************************************************************************/
// PIC32MZ2048ECG144, EFM144 or or EFG144 based HMZ144 board Configuration Bit Settings
// DEVCFG2
#if defined(__32MZ2048EFG144__) || defined(__32MZ2048EFM144__)
#pragma config FPLLIDIV = DIV_4        // System PLL Input Divider (4x Divider) for 24MHz clock (Rev C1 board w EFG) 24MHz/4 = 6MHz
                                       // also 24MHz clock rev C board w EFM (weird - went back to C. rev D also is EFM but with Osc)
#pragma config UPLLFSEL = FREQ_24MHZ    // USB PLL Input Frequency Selection (USB PLL input is 24 MHz)
#else
```

```
#pragma config FPLLIDIV = DIV_2         // System PLL Input Divider (2x Divider) for 12 MHz crystal (Rev B and C boards w ECG) 12MHz/2 = 6MHz

#pragma config UPLLEN = OFF             // USB PLL Enable (USB PLL is disabled)

#endif

#pragma config FPLLRNG = RANGE_5_10_MHZ // System PLL Input Range (5-10 MHz Input)

#pragma config FPLLICLK = PLL_POSC      // System PLL Input Clock Selection (POSC is input to the System PLL)

#pragma config FPLLMULT = MUL_112       // System PLL Multiplier (PLL Multiply by 112) 6MHz * 112 = 672MHz

#pragma config FPLLODIV = DIV_8         // System PLL Output Clock Divider (8x Divider) 672MHz / 2 = 84MHz


// DEVCFG1

#pragma config FNOSC = SPLL             // Oscillator Selection Bits (Primary Osc (HS,EC))

#pragma config FSOSCEN = OFF            // Secondary Oscillator Enable (Disable SOSC)

#if defined(__32MZ2048EFG144__)

#pragma config POSCMOD = EC             // Primary Oscillator Configuration EC - External clock osc

                                // Rev C1 board w EFG uses an Oscillator (Rev D boards too))

#else

#pragma config POSCMOD = HS             // Primary Oscillator Configuration HS - Crystal osc

                                // Rev B and C (w ECG or EFM) use Crystals

#endif

#pragma config FCKSM = CSDCMD           // Clock Switching and Monitor Selection (Clock Switch Disabled, FSCM Disabled)

#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled)

#pragma config FDMTEN = OFF             // Deadman Timer Enable (Deadman Timer is disabled)

#pragma config DMTINTV = WIN_127_128    // Default DMT Count Window Interval (Window/Interval value is 127/128 counter value)

#pragma config DMTCNT = DMT31           // Max Deadman Timer count = 2^31


// DEVCFG0

#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)

#pragma config ICESEL = ICS_PGx2        // ICD/ICE is on PGEC2/PGED2 pins (not default)


#include <xc.h>
```

```
#include <sys/attribs.h>
int tmr2var;
int tmr3var;

int main()
{
    int i;
    T2CONSET = 0;
    INTCONSET = _INTCON_MVEC_MASK; // set MVEC bit
    IFS0CLR = _IFS0_T2IF_MASK; // Clear the timer interrupt status flag
    IEC0SET = _IEC0_T2IE_MASK;
    IPC2CLR = _IPC2_T2IP_MASK | _IPC2_T2IS_MASK;
    IPC2SET = (4 << _IPC2_T2IP_POSITION) & _IPC2_T2IP_MASK;
    IFS0CLR = _IFS0_T3IF_MASK; // Clear the timer interrupt status flag
    IEC0SET = _IEC0_T3IE_MASK;
    IPC3CLR = _IPC3_T3IP_MASK | _IPC3_T3IS_MASK;
    IPC3SET = (3 << _IPC3_T3IP_POSITION) & _IPC3_T3IP_MASK;
    SYSKEY = 0; // Ensure lock
    SYSKEY = 0xAA996655; // Write Key 1
    SYSKEY = 0x556699AA; // Write Key 2
    PB3DIV = _PB3DIV_ON_MASK | 0 & _PB3DIV_PBDIV_MASK; // 0 = div by 1, 1 = div by 2, 2 = div by 3 etc up to 128
    SYSKEY = 0; // Re lock
    T2CON = ((0 << _T2CON_TCKPS_POSITION) & _T2CON_TCKPS_MASK);
    PRECON = (1 & _PRECON_PFMWS_MASK) | ((2 << _PRECON_PREFEN_POSITION) & _PRECON_PREFEN_MASK);
    PR2 = 511;
    PR3 = 999;
    //PRISS = (7 << _PRISS_PRI7SS_POSITION) & _PRISS_PRI7SS_MASK;
    asm("ei");
    T2CONSET = _T2CON_ON_MASK; // Turn Timer 2 and Timer 3 on.
    T3CONSET = _T2CON_ON_MASK; // Do not rearrange these lines so timers will sync
    IFS0SET = _IFS0_T3IF_MASK; //artificially trigger a T3 int
```

```
   for(i = 0; i < 2; i++) // run the following twice so it runs from cache the second time
   {
      TMR2 = 0;
      TMR3 = 5; // Needed to get TMR2 and 3 in sync
   }
   while(1);
}
void __ISR_AT_VECTOR(_TIMER_2_VECTOR, IPL4SOFT) MyTimer2Handler(void)
{
   tmr2var++;
   IFS0CLR = _IFS0_T2IF_MASK;  //clear flag
}
void __ISR_AT_VECTOR(_TIMER_3_VECTOR, IPL3SOFT) MyTimer3Handler(void)
{
   tmr3var++;
   while(TMR3 < 40);
   IFS0CLR = _IFS0_T3IF_MASK;  //clear flag
}
/*******************************************************************************
 * Author: Chris Thomas
 * Date: 10/18/2022
 * Project: Lab 3 - Interrupts in C
 * Updates: N/A
 *******************************************************************************/
// PIC32MZ2048ECG144, EFM144 or or EFG144 based HMZ144 board Configuration Bit Settings
// DEVCFG2
#if defined(__32MZ2048EFG144__) || defined(__32MZ2048EFM144__)
#pragma config FPLLIDIV = DIV_4        // System PLL Input Divider (4x Divider) for 24MHz clock (Rev C1 board w EFG) 24MHz/4 = 6MHz
                                       // also 24MHz clock rev C board w EFM (weird - went back to C. rev D also is EFM but with Osc)
```

```
#pragma config UPLLFSEL = FREQ_24MHZ    // USB PLL Input Frequency Selection (USB PLL input is 24 MHz)

#else

#pragma config FPLLIDIV = DIV_2         // System PLL Input Divider (2x Divider) for 12 MHz crystal (Rev B and C boards w ECG) 12MHz/2 = 6MHz

#pragma config UPLLEN = OFF             // USB PLL Enable (USB PLL is disabled)

#endif

#pragma config FPLLRNG = RANGE_5_10_MHZ // System PLL Input Range (5-10 MHz Input)

#pragma config FPLLICLK = PLL_POSC      // System PLL Input Clock Selection (POSC is input to the System PLL)

#pragma config FPLLMULT = MUL_112       // System PLL Multiplier (PLL Multiply by 112) 6MHz * 112 = 672MHz

#pragma config FPLLODIV = DIV_8         // System PLL Output Clock Divider (8x Divider) 672MHz / 2 = 84MHz


// DEVCFG1

#pragma config FNOSC = SPLL             // Oscillator Selection Bits (Primary Osc (HS,EC))

#pragma config FSOSCEN = OFF            // Secondary Oscillator Enable (Disable SOSC)

#if defined(__32MZ2048EFG144__)

#pragma config POSCMOD = EC             // Primary Oscillator Configuration EC - External clock osc

                                // Rev C1 board w EFG uses an Oscillator (Rev D boards too))

#else

#pragma config POSCMOD = HS             // Primary Oscillator Configuration HS - Crystal osc

                                // Rev B and C (w ECG or EFM) use Crystals

#endif

#pragma config FCKSM = CSDCMD           // Clock Switching and Monitor Selection (Clock Switch Disabled, FSCM Disabled)

#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled)

#pragma config FDMTEN = OFF             // Deadman Timer Enable (Deadman Timer is disabled)

#pragma config DMTINTV = WIN_127_128    // Default DMT Count Window Interval (Window/Interval value is 127/128 counter value)

#pragma config DMTCNT = DMT31           // Max Deadman Timer count = 2^31


// DEVCFG0

#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)

#pragma config ICESEL = ICS_PGx2        // ICD/ICE is on PGEC2/PGED2 pins (not default)
```

```
#include <xc.h>
#include <sys/attribs.h>
int tmr2var;
int tmr3var;

int main()
{
    int i;
    T2CONSET = 0;
    //INTCONSET = _INTCON_MVEC_MASK; // set MVEC bit
    IFS0CLR = _IFS0_T2IF_MASK; // Clear the timer interrupt status flag
    IEC0SET = _IEC0_T2IE_MASK;
    IPC2CLR = _IPC2_T2IP_MASK | _IPC2_T2IS_MASK;
    IPC2SET = (4 << _IPC2_T2IP_POSITION) & _IPC2_T2IP_MASK;
    IFS0CLR = _IFS0_T3IF_MASK; // Clear the timer interrupt status flag
    IEC0SET = _IEC0_T3IE_MASK;
    IPC3CLR = _IPC3_T3IP_MASK | _IPC3_T3IS_MASK;
    IPC3SET = (3 << _IPC3_T3IP_POSITION) & _IPC3_T3IP_MASK;
    SYSKEY = 0; // Ensure lock
    SYSKEY = 0xAA996655; // Write Key 1
    SYSKEY = 0x556699AA; // Write Key 2
    PB3DIV = _PB3DIV_ON_MASK | 0 & _PB3DIV_PBDIV_MASK; // 0 = div by 1, 1 = div by 2, 2 = div by 3 etc up to 128
    SYSKEY = 0; // Re lock
    T2CON = ((0 << _T2CON_TCKPS_POSITION) & _T2CON_TCKPS_MASK);
    PRECON = (1 & _PRECON_PFMWS_MASK) | ((2 << _PRECON_PREFEN_POSITION) & _PRECON_PREFEN_MASK);
    PR2 = 511;
    PR3 = 999;
    //PRISS = (7 << _PRISS_PRI7SS_POSITION) & _PRISS_PRI7SS_MASK;
    asm("ei");
    T2CONSET = _T2CON_ON_MASK; // Turn Timer 2 and Timer 3 on.
```

```
    T3CONSET = _T2CON_ON_MASK; // Do not rearrange these lines so timers will sync
    IFS0SET = _IFS0_T3IF_MASK; //artificially trigger a T3 int
    for(i = 0; i < 2; i++) // run the following twice so it runs from cache the second time
    {
        TMR2 = 0;
        TMR3 = 5; // Needed to get TMR2 and 3 in sync
    }
    while(1);
}

void __ISR_AT_VECTOR(0, RIPL) MyTimerHandler(void)
{
    if (IFS0 & _IFS0_T2IF_MASK)
    {
        tmr2var++;
        IFS0CLR = _IFS0_T2IF_MASK;  //clear flag
    }
    else
    {
        tmr3var++;
        IFS0CLR = _IFS0_T3IF_MASK;  //clear flag
    }
}
```