آزمایش سوم PIC، کار با اینتراپت (امتیازی)

در این بخش سعی میکنیم بیشتر با اینتراپت آشنا شویم.

کدی که به شما داده شده به طور متناوب هر ۴۰۰۰۰ کلاک یک بار به وقفه خورده و کد اینتراپت (که به ISR یا Interrupt Service Routine شناخته می شود) اجرا میشود و سپس دوباره به کد اصلی برگشته و ادامه می دهد. در این کد تابع CoreTimerIntHandler (از خط ۸۳ الی ۱۵۲) همان ISR است. ISR شامل عملیات مختلفی از جمله استک کردن رجیسترهای مورد نیاز و ... می باشد که اول و آخر ISR نوشته شده است و نیازی نیست آن‌ها را تغییر بدید. خط ۱۱۵ الی ۱۳۲ در واقع کد خالص اینتراپت است که باید آن را پاک کرده و به جای آن کد دلخواه خود را بنویسید. به دو نکته زیر دقت کنید:

- به علت اینکه ممکن است اینتراپت بین هر دو اینستراکشن از کد main رخ دهد، نمیتوانیم در ISR محتوای رجیسترها را خراب کنیم. لذا نیاز است که محتوای رجیسترها در مموری ذخیره (استک) شده و پس از استفاده از آن‌ها به عنوان رجیسترهای temporary در ISR، در انتهای آن محتوای اصلیشان را از مموری به رجیسترها برگردانیم. رجیسترهای t0 و t1 یا $8 و $9 استک شده اند، لذا تنها می‌توانید از این دو رجیستر برای نوشتن کد اینتراپت استفاده کنید. در صورت نیاز می توانید رجیستر های بیشتری را استک کرده و از آن‌ها در کد اینتراپت استفاده کنید.
- به هیچ وجه بخش قرمز رنگ را تغییر ندهید، مگر برای استک کردن در ISR.

```
la $8, TRISB
sw $0, 0($8)

la $8, LATB
sw $0, 0($8)


// display HEL0 on 7-segments

la $8, seg
ori $9, $0, 0x0876   //H
sw $9, 0($8)         //seg1

ori $9, $0, 0x0479   //E
sw $9, 4($8)         //seg2

ori $9, $0, 0x0238   //L
sw $9, 8($8)         //seg3

ori $9, $0, 0x013F   //O
sw $9, 12($8)        //seg4



// do nothing anymore

loop:
    j loop

.end main

/*********************************************************************
 * This is the actual interrupt handler that gets installed
 * in the interrupt vector table. It jumps to the core-timer
 * interrupt handler function.
 *
 * Note: The ".section .vector_0" is not allocatable. Hence to force
 * this section to be allocatable, use the "ax" directive.
 ********************************************************************/
.section .vector_0,code
   j      CoreTimerIntHandler

/*********************************************************************
 * CoreTimerIntHandler()
 * Interrupt handler function for core-timer. The function
 * clears the interrupt flag, shows one digit on the seven segment board and updates the
 * core-timer registers.
 *
 * pre-condition: A jump to ISR is registered in vector table
 * Input: none
 * Output: none
 * Side effect: shows one digit on the seven segment board
 ********************************************************************/
.text
```

ISR

```
.ent CoreTimerIntHandler
CoreTimerIntHandler:
   /* interrupt prologue */
   addiu sp, sp, -20
   sw $1, 0(sp)
   sw t0, 4(sp)
   sw t1, 8(sp)
```

→ stacking t0 and t1

```

   mfc0 t1, $14          /* read EPC register (program counter at last exeption) */
   sw t1, 12(sp)         /* save EPC on stack */

   mfc0 t1, $12          /* read STATUS register */
   sw t1, 16(sp)         /* save STATUS on stack */

   mfc0        t0, $13          /* read CAUSE register */
   srl         t0, t0,0xA       /* align RIPL (Requested Interrupt Priority Level) to bit 0 */
   ins         t1, t0, 10, 6    /* insert RIPL to IPL(Interrupt Priority Level) field (copy and replace from 6 LSBs of t0 to t1 starting at bit 10 of t1, t1<15:10> = t0<5:0>) */
   ins         t1, zero, 1, 4     /* t1<4:1> = zero<3:0> */
   mtc0        t1, $12          /* write STATUS register  (status<15:10> = cause<15:10>, status<4:1> = 0)  */


   /*update core-timer routine for next interrupt. */
   ori     t0, zero, TIMER_PERIOD
   mtc0    $0, $9   // count register ($9) in coprocessor0 = 0
   mtc0    t0, $11  // compare register ($11) in coprocessor0 = TIMER_PERIOD

   /* clear interrupt flag */
   addiu   t1, zero, 1
   la      t0, IFS0CLR
   sw      t1, 0(t0)    // IFS0<0> = 0

   ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////



















   ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

   /* interrupt epilogue */
   di                     /* disable interrupts */

   lw t1, 12(sp)         /* restore EPC from stack */
   mtc0 t1, $14

   lw t1, 16(sp)         /* restore STATUS from stack */
   mtc0 t1, $12

   lw $1, 0(sp)
   lw t0, 4(sp)
   lw t1, 8(sp)
   addiu sp, sp, 20

   ei                     /* enable interrupts */
   /* return from interrupt */
   eret
.end CoreTimerIntHandler

 /*********************************************************************
 * InterruptSetup()
 *
 * cause<23> = 1
 * ei
 * count = 0
 * compare = TIMER_PERIOD
 * IFS0CLR<0> = 0
 * IPC0<4:2> = CT_INT_PRIOR_3
 * IEC0<0> = CT_INT_ON
 ********************************************************************/
.ent InterruptSetup
InterruptSetup:
    /* function prologue - save registers used in this function
     * on stack and adjust stack-pointer
     */
   addiu   sp, sp, -8
   sw      s0, 0(sp)
   sw      s1, 4(sp)

   //INTEnableSystemMultiVectoredInt:
   mfc0    s0, $13
   lui     s1, 0x0080
   or      s0, s0, s1
   mtc0    s0, $13  // bit <23> (IV bit = Interrupt Vector) in the cause register ($13) in coprocessor0 is set
   ei      /* enable system-wide interrupts */

   //OpenCoreTimer:
   ori     s0, zero, TIMER_PERIOD  // s0 = TIMER_PERIOD
   mtc0    $0, $9   // count register ($9) in coprocessor0 = 0
   mtc0    s0, $11  // compare register ($11) in coprocessor0 = TIMER_PERIOD

   //mCTClearIntFlag:
   addiu   s1, zero, 1
   la      s0, IFS0CLR
   sw      s1, 0(s0)    // bit <0> in the Interrupt Flag Status register is cleared

   //mCTSetIntPriority:
   addiu   s1, zero, (7 << 2)
   la      s0, IPC0CLR
   sw      s1, 0(s0)
   ori     s0, zero, CT_INT_PRIOR_3 // s0 = CT_INT_PRIOR_3
   sll     s1, s0, 2
   la      s0, IPC0SET
   sw      s1, 0(s0)    // bits <4:2> in the Interrupt Priority Control register (of the core timer interrupt), which are the core timer interrupt priority bits = CT_INT_PRIOR_3

   //mCTIntEnable:
   ori     s1, zero, CT_INT_ON  // s1 = CT_INT_ON
   la      s0, IEC0SET
   sw      s1, 0(s0)    // bit <0> in the Interrupt Enable Control register (of the core timer interrupt), which is the core timer interrupt enable bit = CT_INT_ON

   /* function epilogue - restore registers used in this function
    * from stack and adjust stack-pointer
    */
   lw      s1, 4(sp)
   lw      s0, 0(sp)
   addiu   sp, sp, 8

   /* return to caller */
   jr      ra
.end InterruptSetup
```

بخش ششم: با استفاده از اینتراپت، یک LED در بورد button-led که در آزمایش اول از آن استفاده کردید را توسط اینتراپت به طور متناوب روشن و خاموش کنید، به طوری که سرعت چشمک زدن آن ثابت و با چشم واضحاً قابل تشخیص باشد (مثلا یک بار در ثانیه).

بخش هفتم: کدی بنویسید که مقادیر وارد شده در ۱۲ بیت اول ۴ متغیر seg (مانند helo که در بخش main کد داده شده بود) به طور چشمک‌زن روی Seg-7ها نمایش داده شود. به طوری که سرعت چشمک زدن آن ثابت و با چشم واضحاً قابل تشخیص باشد (مثلا یک بار در ثانیه).

بخش هشتم: کدی بنویسید که مقادیر وارد شده در ۱۲ بیت اول ۱۰ متغیر seg روی Seg-7ها به ترتیب از چپ به راست بر روی چهار Seg-7 نمایش داده شود به شکلی که در هر لحظه چهار رقم آن نمایش داده شده و سپس به سمت چپ حرکت کند. این نمایش باید بصورت نامتناهی انجام شود.

بخش نهم: کدی بنویسید که کی‌پد را در ISR خوانده و عدد متناظر با کلید زده شده را در یکی از Seg-7ها نمایش دهد. تبدیل سطر و ستون یک کلید به معادل عددی آن می‌تواند هم بصورت محاسباتی و هم با بکارگیری یک LUT صورت پذیرد.

بخش دهم: کدی بنویسید که علاوه بر نشان دادن عدد ورودی در Seg-7ها، قابلیت کم و زیاد کردن روشنایی رقم‌های نمایش داده شده را با زدن کلیدهای پرانتز ")" و "(" با حداقل ده درجه داشته باشد.