```
                .section .vectors, "ax"
                B       _start              // reset vector
                B       SERVICE_UND         // undefined instruction vector
                B       SERVICE_SVC         // software interrupt vector
                B       SERVICE_ABT_INST    // aborted prefetch vector
                B       SERVICE_ABT_DATA    // aborted data vector
                .word   0                   // unused vector
                B       SERVICE_IRQ         // IRQ interrupt vector
                B       SERVICE_FIQ         // FIQ interrupt vector
                .text
                .global _start
_start:
/* Set up stack pointers for IRQ and SVC processor modes */
                MOV     R1, #0b11010010     // interrupts masked, MODE = IRQ
                MSR     CPSR_c, R1          // change to IRQ mode
                LDR     SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory

                MOV     R1, #0b11010011     // interrupts masked, MODE = SVC
                MSR     CPSR, R1            // change to supervisor mode
                LDR     SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory

                BL      CONFIG_GIC          // configure the ARM generic interrupt controller
                BL      CONFIG_PRIV_TIMER   // configure the private timer
                BL      CONFIG_TIMER        // configure the Interval Timer
                BL      CONFIG_KEYS         // configure the pushbutton KEYs port

/* Enable IRQ interrupts in the ARM processor */
                MOV      R0, #0b01010011    // IRQ unmasked, MODE = SVC
                MSR      CPSR_c, R0

                LDR     R5, =0xFF200000     // LEDR base address
                LDR     R6, =0xFF200020     // HEX3-0 base address
LOOP:
                LDR     R4, COUNT           // global variable
                STR     R4, [R5]            // light up the red lights
                LDR     R4, HEX_CODE        // global variable
                STR     R4, [R6]            // show the time in format SS:DD

                B       LOOP                // Endlessly loop

/* Configure the MPCore private timer to create interrupts every 1/100 seconds */
CONFIG_PRIV_TIMER:
                LDR     R0, =0xFFFEC600     // A9 private timer address
                LDR     R1, =2000000        // 0.01 seconds on 200MHz clock
                STR     R1, [R0]
                MOV     R1, #0b111          // Start timer and set it to auto-reload
                STR     R1, [R0, #0x8]
                BX      LR

/* Configure the Interval Timer to create interrupts at 0.25 second intervals */
CONFIG_TIMER:   LDR     R0, =0xFF202000     // FPGA timer base address
                LDR     R1, RATE
                STR     R1, [R0, #8]        // Set lower bits of timer
                LSR     R1, #16
                STR     R1, [R0, #12]       // Set upper bits of timer
                MOV     R1, #0b0111         // Control register bits
                STR     R1, [R0, #4]        // Start timer, set auto-reload and enable interrupts
                BX      LR

/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
                LDR     R0, =0xFF200050     // KEY address
                MOV     R1, #0xF            // set interrupt mask bits
                STR     R1, [R0, #0x8]      // interrupt mask register (base + 8)
                BX      LR

/* Define the exception service routines */
```

```
SERVICE_IRQ:    PUSH    {R0-R7, LR}
                LDR     R4, =0xFFFEC100     // GIC CPU interface base address
                LDR     R5, [R4, #0x0C]     // read the ICCIAR in the CPU interface

FPGA_IRQ1_HANDLER:
                CMP     R5, #29             // check the interrupt ID
                BEQ     PRIV_INTERRUPT
                CMP     R5, #73
                BEQ     KEY_INTERRUPT
                CMP     R5, #72
                BEQ     CLK_INTERRUPT

UNEXPECTED:     B       UNEXPECTED          // if not recognized, stop here
PRIV_INTERRUPT: BL      PRIV_TIMER_ISR
                B       EXIT_IRQ
KEY_INTERRUPT:  BL      KEY_ISR
                B       EXIT_IRQ
CLK_INTERRUPT:  BL      TIMER_ISR
                B       EXIT_IRQ

EXIT_IRQ:       STR     R5, [R4, #0x10]     // write to the End of Interrupt Register
(ICCEOIR)
                POP     {R0-R7, LR}
                SUBS    PC, LR, #4          // return from exception

/* Check if private timer generated an interrupt and increments time */
PRIV_TIMER_ISR: LDR     R0, =0xFFFEC600     // base address of private timer
                MOV     R1, #1
                STR     R1, [R0, #0xC]      // clear the interrupt
                LDR     R1, TIME            // Increment TIME
                ADD     R1, #1
                LDR     R2, =6000           // Load a literal
                CMP     R1, R2              // Wrap around to 0 when TIME > 5999
                BNE     SET_HEX
                MOV     R1, #0

SET_HEX:        PUSH    {LR}
                LDR     R3, =BIT_CODE
                STR     R1, TIME            // Stores TIME
                MOV     R0, R1              // Separate R1 into its digits
                BL      DIVIDE
                LDRB    R0, [R3, +R0]       // Get pattern for ones digit
                MOV     R2, R0

                MOV     R0, R1              // Get tens digit
                BL      DIVIDE
                LDRB    R0, [R3, +R0]       // Get pattern for tens digit
                LSL     R0, #8
                ORR     R2, R0

                MOV     R0, R1              // Get hundredth digit
                BL      DIVIDE              // Remainder from divide is thousandth digit
                LDRB    R0, [R3, +R0]       // Get pattern for hundreds digit
                LSL     R0, #16
                ORR     R2, R0
                LDRB    R1, [R3, +R1]       // Get pattern for thousandth digit
                LSL     R1, #24
                ORR     R2, R1

                STR     R2, HEX_CODE        // Set HEX_CODE for 7 segments

END_PRIV_TIMER_ISR:
                POP     {PC}                // Return

/* Check if FPGA timer generated an interrupt and adds RUN to COUNT */
TIMER_ISR:      LDR     R0, =0xFF202000     // base address of FPGA timer
                MOV     R2, #0
                STR     R2, [R0]            // clear the interrupt
                LDR     R0, RUN             // Load RUN toggle
                LDR     R1, COUNT           // Load counter
                ADD     R1, R0              // Increment counter by RUN
                STR     R1, COUNT           // Store incremented counter
END_TIMER_ISR:  BX      LR                  // Return

/* Check if a key has been pressed and toggles RUN */
KEY_ISR:        LDR     R0, =0xFF200050     // base address of pushbutton KEY port
                LDR     R1, [R0, #0xC]      // read edge capture register
                MOV     R2, #0xF
                STR     R2, [R0, #0xC]      // clear the interrupt

CHECK_KEY0:     MOV     R3, #0b0001
                CMP     R3, R1              // Check for KEY0
                BNE     CHECK_KEY1
                LDR     R0, RUN             // LOAD RUN toggle
                EOR     R0, #1              // Toggle RUN
                STR     R0, RUN             // Set RUN in memory
                B       END_KEY_ISR

CHECK_KEY1:     MOV     R3, #0b0010
                CMP     R3, R1              // Check for KEY1
                BNE     CHECK_KEY2
                LDR     R0, =0xFF202000     // Address of FPGA timer
                MOV     R1, #0b1000
                STR     R1, [R0, #4]        // Stop timer
                LDR     R1, RATE
                LSR     R1, #1              // Double the rate (HALF the timeout)
                CMP     R1, #0xFF               // Limit maximum rate
                BGT     SET_TIMER
                MOV     R1, #0xFF
                B       SET_TIMER

CHECK_KEY2:     MOV     R3, #0b0100
                CMP     R3, R1              // Check for KEY2
                BNE     END_KEY_ISR
                LDR     R0, =0xFF202000     // Address of FPGA timer
                MOV     R1, #0b1000
                STR     R1, [R0, #4]        // Stop timer
                LDR     R1, RATE
                LSL     R1, #1              // Half the rate (DOUBLE the timeout)
                CMP     R1, #0              // Limit minimum rate
                BNE     SET_TIMER
                MOV     R1, #0xA0000000

SET_TIMER:      STR     R1, RATE            // Store the rate
                STR     R1, [R0, #8]        // Set lower bits of timer
                LSR     R1, #16
                STR     R1, [R0, #12]       // Set upper bits of timer
                MOV     R1, #0b0111         // Control register bits
                STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                interrupts

END_KEY_ISR:    BX      LR                  // Return

/* Subroutine to perform the integer division R0 / 10.
 * Returns quotient in R1 and remainder in R0
 */
DIVIDE:         PUSH    {R2,LR}
                MOV     R2, #0
CONT:           CMP     R0, #10
                BLT     DIV_END
                SUB     R0, #10
                ADD     R2, #1
                B       CONT
DIV_END:        MOV     R1, R2              // quotient in R1 (remainder in R0)
                POP     {R2,PC}

/* Global variables */
                .global COUNT
```

```
COUNT:          .word   0x0                 // used by timer
                .global RUN                 // used by pushbutton KEYs
RUN:            .word   0x1                 // initial value to increment COUNT
RATE:           .word   25000000
TIME:           .word   0x0                 // used for real-time clock
                .global HEX_CODE
HEX_CODE:       .word   0x0                 // used for 7-segment displays
BIT_CODE:       .byte   0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
                .byte   0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
                .skip   2                   // pad with 2 bytes to maintain word
                alignment

                .end
```