```
                .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
/* Set up stack pointers for IRQ and SVC processor modes */
_start:         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

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

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

IDLE:           B       IDLE                // main program simply idles

/* 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, #73             // check the interrupt ID

UNEXPECTED:     BNE     UNEXPECTED          // if not recognized, stop here
                BL      KEY_ISR

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 which key has been pressed and writes accordingly */
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
                LDR     R0, =0xFF200020     // based address of HEX display

CHECK_KEY0:     MOV     R3, #0b0001
                CMP     R3, R1              // Check for KEY0
                BNE     CHECK_KEY1
                MOV     R2, #0b00111111     // '0'
                LDRB    R3, [R0]            // HEX0
                CMP     R2, R3
                BEQ     CLEAR_HEX0          // Check is HEX0 is already '0'
                STRB    R2, [R0]            // Display '0'
                B       END_KEY_ISR
CLEAR_HEX0:     MOV     R2, #0
                STRB    R2, [R0]            // Display blank
                B       END_KEY_ISR
```

```

CHECK_KEY1:     MOV     R3, #0b0010
                CMP     R3, R1                  // Check for KEY1
                BNE     CHECK_KEY2
                MOV     R2, #0b00000110         // '1'
                LDRB    R3, [R0, #1]            // HEX1
                CMP     R2, R3
                BEQ     CLEAR_HEX1              // Check is HEX1 is already '1'
                STRB    R2, [R0, #1]            // Display '1'
                B       END_KEY_ISR
CLEAR_HEX1:     MOV     R2, #0
                STRB    R2, [R0, #1]            // Display blank
                B       END_KEY_ISR

CHECK_KEY2:     MOV     R3, #0b0100
                CMP     R3, R1                  // Check for KEY2
                BNE     IS_KEY3
                MOV     R2, #0b01011011         // '2'
                LDRB    R3, [R0, #2]            // HEX2
                CMP     R2, R3
                BEQ     CLEAR_HEX2              // Check is HEX2 is already '2'
                STRB    R2, [R0, #2]            // Display '2'
                B       END_KEY_ISR
CLEAR_HEX2:     MOV     R2, #0
                STRB    R2, [R0, #2]            // Display blank
                B       END_KEY_ISR

IS_KEY3:        MOV     R2, #0b01001111         // '3'
                LDRB    R3, [R0, #3]            // HEX3
                CMP     R2, R3
                BEQ     CLEAR_HEX3              // Check is HEX3 is already '3'
                STRB    R2, [R0, #3]            // Display '3'
                B       END_KEY_ISR
CLEAR_HEX3:     MOV     R2, #0
                STRB    R2, [R0, #3]            // Display blank
                B       END_KEY_ISR

END_KEY_ISR:    BX      LR                      // Return

                .end

```