```
/* This program counts from 0 to 99 on HEX1 and HEX0
 * at a rate of 4Hz, pressing any key will stop/start the conter
 */
            .text
            .global _start

_start:     LDR     R6, =0xFF200020     // HEX3-HEX0 Address
            LDR     R7, =0xFF200050     // KEY Address
            MOV     R8, #BIT_CODES      // Address of BIT_CODES array
            MOV     R2, #0              // R2 will be the counter
            MOV     R3, #1              // R3 will determine whether to count or not
MAIN:       LDRB    R5, [R7, #0xC]      // Read Edgecapture register
            CMP     R5, #0
            BEQ     DO_DELAY            // If Edgecapture is not 0 the a key has been
            pressed
WAIT:       LDR     R5, [R7]            // Poll KEYs to see if the KEY has been released
            CMP     R5, #0
            BNE     WAIT                // Wait for KEY to be released
            MOV     R5, #0xF            // Reset Edgecapture
            STR     R5, [R7, #0xC]
            MOV     R4, #1
            SUB     R3, R4, R3          // Subtract R3 from 1 to invert it (1 <-> 0)

DO_DELAY:   LDR     R4, =200000000      // Delay counter
SUB_LOOP:   SUBS    R4, #1
            BNE     SUB_LOOP

            CMP     R3, #1              // When R3 = 1, increment counter
            BNE     DISPLAY
            ADD     R2, #1
            CMP     R2, #100            // Wrap around to 0 when R2 > 99
            BNE     DISPLAY
            MOV     R2, #0

DISPLAY:    MOV     R0, R2              // Separate R2 into its digits
            BL      DIVIDE
            LDRB    R0, [R8, +R0]       // Get pattern for ones digit
            LDRB    R1, [R8, +R1]       // Get pattern for ones digit
            LSL     R1, #8
            ORR     R0, R1              // Put pattern in the same reg as the tens digit
            STR     R0, [R6]            // Display counter
            B       MAIN                // Program infinitely counts/loops


/* 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}

BIT_CODES:  .byte   0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
            .byte   0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
            .skip   2               // pad with 2 bytes to maintain word alignment
```