```
/* Program that displays a number on HEX0
 * that changes based on the key pressed:
 * 0 - Set to 0
 * 1 - Increment
 * 2 - Decrement
 * 3 - Clear (any key after that will set to 0)
 */
            .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     R0, #0              // R0 will be the counter
MAIN:       LDR     R5, [R7]            // Read KEYs
            CMP     R5, #0
            BEQ     DISPLAY             // Check is no KEY has been pressed
            MOV     R4, R5              // Store KEY value when 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

// Check which key has been pressed and act accordingly
ZERO:       CMP     R4, #0b0001         // Check if KEY0 is pressed
            BNE     INCREMENT
            MOV     R0, #0              // Set counter to 0
            B       DISPLAY
INCREMENT:  CMP     R4, #0b0010         // Check if KEY1 is pressed
            BNE     DECREMENT
            ADD     R0, #1              // Increment counter
            CMP     R0, #10             // Counter goes from 0 to 9, so wrap to 0 if = 10
            MOVEQ   R0, #0
            B       DISPLAY
DECREMENT:  CMP     R4, #0b0100         // Check if KEY2 is pressed
            BNE     CLEAR
            SUBS    R0, #1              // Decrement counter
            MOVMI   R0, #9              // Counter goes from 0 to 9, so wrap to 9 if =-1
            B       DISPLAY
CLEAR:      MOV     R1, #0              // If it gets to here KEY3 is definitely pressed
            STRB    R1, [R6]            // Set HEX to blank
CLEAR_WAIT: LDR     R5, [R7]            // Wait for any KEY to be pressed
            CMP     R5, #0
            BEQ     CLEAR_WAIT
            MOV     R4, #0b0001         // Set the KEY pressed to be "KEY0"
            B       WAIT                // Wait for the KEY to be released

DISPLAY:    LDRB    R1, [R8, +R0]       // Get digit to display
            STRB    R1, [R6]            // Display to HEX0
            B       MAIN                // Program infinitely counts/loops

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

```
/* 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
```

```
/* 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     R9, =0xFFFEC600     // A9 private timer address
            LDR     R4, =50000000       // 0.25 seconds on 200MHz clock
            STR     R4, [R9]
            MOV     R4, #0b011          // Start timer and set it to auto-reload
            STR     R4, [R9, #0x8]
            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     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)

DELAY:      LDR     R4, [R9, #0xC]      // Load timer interrupt flag
            CMP     R4, #0              // Keep on delaying until interrupt flag is 1
            BEQ     DELAY
            STR     R4, [R9, #0xC]      // Reset interrupt flag

            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
```

```
/* This program counts from 00.00 to 59.99 seconds on HEX3-HEX0
 * at a rate of 4Hz, pressing any key will stop/start the conter
 */
            .text
            .global _start

_start:     LDR     R9, =0xFFFEC600     // A9 private timer address
            LDR     R4, =2000000        // 0.01 seconds on 200MHz clock
            STR     R4, [R9]
            MOV     R4, #0b011          // Start timer and set it to auto-reload
            STR     R4, [R9, #0x8]
            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     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)

DELAY:      LDR     R4, [R9, #0xC]      // Load timer interrupt flag
            CMP     R4, #0              // Keep on delaying until interrupt flag is 1
            BEQ     DELAY
            STR     R4, [R9, #0xC]      // Reset interrupt flag

            CMP     R3, #1              // When R3 = 1, increment counter
            BNE     DISPLAY
            ADD     R2, #1
            LDR     R4, =6000           // Load a literal
            CMP     R2, R4              // Wrap around to 0 when R2 > 5999
            BNE     DISPLAY
            MOV     R2, #0

DISPLAY:    MOV     R0, R2              // Separate R2 into its digits
            BL      DIVIDE
            LDRB    R4, [R8, +R0]       // Get pattern for ones digit

            MOV     R0, R1              // Get tens digit
            BL      DIVIDE
            LDRB    R0, [R8, +R0]       // Get pattern for tens digit
            LSL     R0, #8
            ORR     R4, R0

            MOV     R0, R1              // Get hundredth digit
            BL      DIVIDE              // Remainder from divide is thousandth digit
            LDRB    R0, [R8, +R0]       // Get pattern for hundreds digit
            LSL     R0, #16
            ORR     R4, R0
            LDRB    R1, [R8, +R1]       // Get pattern for thousandth digit
            LSL     R1, #24
            ORR     R4, R1

            STR     R4, [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

```