

```

1  /* Program that displays a number on HEX0
2  * that changes based on the key pressed:
3  * 0 - Set to 0
4  * 1 - Increment
5  * 2 - Decrement
6  * 3 - Clear (any key after that will set to 0)
7  */
8
9      .text
10     .global _start
11
12 _start:    LDR     R6, =0xFF200020    // HEX3-HEX0 Address
13           LDR     R7, =0xFF200050    // KEY Address
14           MOV     R8, #BIT_CODES     // Address of BIT_CODES array
15           MOV     R0, #0              // R0 will be the counter
16 MAIN:     LDR     R5, [R7]            // Read KEYs
17           CMP     R5, #0
18           BEQ     DISPLAY             // Check is no KEY has been pressed
19           MOV     R4, R5              // Store KEY value when a key has been pressed
20 WAIT:     LDR     R5, [R7]            // Poll KEYs to see if the KEY has been released
21           CMP     R5, #0
22           BNE     WAIT                // Wait for KEY to be released
23
24 // Check which key has been pressed and act accordingly
25 ZERO:     CMP     R4, #0b0001        // Check if KEY0 is pressed
26           BNE     INCREMENT
27           MOV     R0, #0              // Set counter to 0
28           B       DISPLAY
29 INCREMENT: CMP     R4, #0b0010        // Check if KEY1 is pressed
30           BNE     DECREMENT
31           ADD     R0, #1              // Increment counter
32           CMP     R0, #10             // Counter goes from 0 to 9, so wrap to 0 if = 10
33           MOVEQ   R0, #0
34           B       DISPLAY
35 DECREMENT: CMP     R4, #0b0100        // Check if KEY2 is pressed
36           BNE     CLEAR
37           SUBS    R0, #1              // Decrement counter
38           MOVMI   R0, #9              // Counter goes from 0 to 9, so wrap to 9 if =-1
39           B       DISPLAY
40 CLEAR:     MOV     R1, #0              // If it gets to here KEY3 is definitely pressed
41           STRB    R1, [R6]            // Set HEX to blank
42 CLEAR_WAIT: LDR     R5, [R7]            // Wait for any KEY to be pressed
43           CMP     R5, #0
44           BEQ     CLEAR_WAIT
45           MOV     R4, #0b0001        // Set the KEY pressed to be "KEY0"
46           B       WAIT                // Wait for the KEY to be released
47
48 DISPLAY:   LDRB    R1, [R8, +R0]      // Get digit to display
49           STRB    R1, [R6]            // Display to HEX0
50           B       MAIN                // Program infinitely counts/loops
51
52 BIT_CODES: .byte   0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
53           .byte   0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
54           .skip    2                  // pad with 2 bytes to maintain word alignment

```



```

1  /* This program counts from 0 to 99 on HEX1 and HEX0
2  * at a rate of 4Hz, pressing any key will stop/start the conter
3  */
4      .text
5      .global _start
6
7  _start:    LDR        R6, =0xFF200020    // HEX3-HEX0 Address
8            LDR        R7, =0xFF200050    // KEY Address
9            MOV        R8, #BIT_CODES     // Address of BIT_CODES array
10           MOV        R2, #0              // R2 will be the counter
11           MOV        R3, #1              // R3 will determine whether to count or not
12  MAIN:     LDRB        R5, [R7, #0xC]     // Read Edgecapture register
13           CMP        R5, #0
14           BEQ        DO_DELAY            // If Edgecapture is not 0 the a key has been
                                           pressed
15  WAIT:     LDR        R5, [R7]            // Poll KEYs to see if the KEY has been released
16           CMP        R5, #0
17           BNE        WAIT                // Wait for KEY to be released
18           MOV        R5, #0xF            // Reset Edgecapture
19           STR        R5, [R7, #0xC]
20           MOV        R4, #1
21           SUB        R3, R4, R3          // Subtract R3 from 1 to invert it (1 <-> 0)
22
23  DO_DELAY: LDR        R4, =200000000     // Delay counter
24  SUB_LOOP: SUBS       R4, #1
25           BNE        SUB_LOOP
26
27           CMP        R3, #1              // When R3 = 1, increment counter
28           BNE        DISPLAY
29           ADD        R2, #1
30           CMP        R2, #100            // Wrap around to 0 when R2 > 99
31           BNE        DISPLAY
32           MOV        R2, #0
33
34  DISPLAY:  MOV        R0, R2              // Separate R2 into its digits
35           BL         DIVIDE
36           LDRB        R0, [R8, +R0]      // Get pattern for ones digit
37           LDRB        R1, [R8, +R1]      // Get pattern for ones digit
38           LSL        R1, #8
39           ORR        R0, R1              // Put pattern in the same reg as the tens digit
40           STR        R0, [R6]            // Display counter
41           B          MAIN                // Program infinitely counts/loops
42
43
44  /* Subroutine to perform the integer division R0 / 10.
45  * Returns quotient in R1 and remainder in R0
46  */
47  DIVIDE:   PUSH        {R2,LR}
48           MOV        R2, #0
49  CONT:     CMP        R0, #10
50           BLT        DIV_END
51           SUB        R0, #10
52           ADD        R2, #1
53           B          CONT
54  DIV_END:  MOV        R1, R2              // quotient in R1 (remainder in R0)
55           POP        {R2,PC}
56
57  BIT_CODES: .byte      0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
58            .byte      0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
59            .skip      2                  // pad with 2 bytes to maintain word alignment
60

```



```

1  /* This program counts from 0 to 99 on HEX1 and HEX0
2  * at a rate of 4Hz, pressing any key will stop/start the conter
3  */
4      .text
5      .global _start
6
7  _start:    LDR        R9, =0xFFFFEC600    // A9 private timer address
8            LDR        R4, =50000000      // 0.25 seconds on 200MHz clock
9            STR        R4, [R9]
10           MOV        R4, #0b011         // Start timer and set it to auto-reload
11           STR        R4, [R9, #0x8]
12           LDR        R6, =0xFF200020     // HEX3-HEX0 Address
13           LDR        R7, =0xFF200050     // KEY Address
14           MOV        R8, #BIT_CODES      // Address of BIT_CODES array
15           MOV        R2, #0              // R2 will be the counter
16           MOV        R3, #1              // R3 will determine whether to count or not
17  MAIN:     LDRB       R5, [R7, #0xC]      // Read Edgecapture register
18           CMP        R5, #0
19           BEQ        DELAY               // If Edgecapture is not 0 the a key has been
20           pressed
21  WAIT:     LDR        R5, [R7]            // Poll KEYs to see if the KEY has been released
22           CMP        R5, #0
23           BNE        WAIT               // Wait for KEY to be released
24           MOV        R5, #0xF           // Reset Edgecapture
25           STR        R5, [R7, #0xC]
26           MOV        R4, #1
27           SUB        R3, R4, R3          // Subtract R3 from 1 to invert it (1 <-> 0)
28  DELAY:    LDR        R4, [R9, #0xC]      // Load timer interrupt flag
29           CMP        R4, #0              // Keep on delaying until interrupt flag is 1
30           BEQ        DELAY
31           STR        R4, [R9, #0xC]      // Reset interrupt flag
32
33           CMP        R3, #1              // When R3 = 1, increment counter
34           BNE        DISPLAY
35           ADD        R2, #1
36           CMP        R2, #100            // Wrap around to 0 when R2 > 99
37           BNE        DISPLAY
38           MOV        R2, #0
39
40  DISPLAY:  MOV        R0, R2              // Separate R2 into its digits
41           BL         DIVIDE
42           LDRB       R0, [R8, +R0]        // Get pattern for ones digit
43           LDRB       R1, [R8, +R1]        // Get pattern for ones digit
44           LSL        R1, #8
45           ORR        R0, R1              // Put pattern in the same reg as the tens digit
46           STR        R0, [R6]            // Display counter
47           B         MAIN                 // Program infinitely counts/loops
48
49
50  /* Subroutine to perform the integer division R0 / 10.
51  * Returns quotient in R1 and remainder in R0
52  */
53  DIVIDE:   PUSH        {R2,LR}
54           MOV        R2, #0
55  CONT:     CMP        R0, #10
56           BLT        DIV_END
57           SUB        R0, #10
58           ADD        R2, #1
59           B         CONT
60  DIV_END:  MOV        R1, R2              // quotient in R1 (remainder in R0)
61           POP        {R2,PC}
62
63  BIT_CODES: .byte     0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
64            .byte     0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
65            .skip     2                  // pad with 2 bytes to maintain word alignment
66

```



```

1  /* This program counts from 00.00 to 59.99 seconds on HEX3-HEX0
2  * at a rate of 4Hz, pressing any key will stop/start the conter
3  */
4      .text
5      .global _start
6
7  _start:    LDR        R9, =0xFFFFEC600    // A9 private timer address
8            LDR        R4, =2000000        // 0.01 seconds on 200MHz clock
9            STR        R4, [R9]
10           MOV        R4, #0b011          // Start timer and set it to auto-reload
11           STR        R4, [R9, #0x8]
12           LDR        R6, =0xFF200020      // HEX3-HEX0 Address
13           LDR        R7, =0xFF200050      // KEY Address
14           MOV        R8, #BIT_CODES       // Address of BIT_CODES array
15           MOV        R2, #0               // R2 will be the counter
16           MOV        R3, #1               // R3 will determine whether to count or not
17  MAIN:     LDRB       R5, [R7, #0xC]       // Read Edgecapture register
18           CMP        R5, #0
19           BEQ        DELAY                // If Edgecapture is not 0 the a key has been
20           pressed
21  WAIT:     LDR        R5, [R7]             // Poll KEYs to see if the KEY has been released
22           CMP        R5, #0
23           BNE        WAIT                 // Wait for KEY to be released
24           MOV        R5, #0xF             // Reset Edgecapture
25           STR        R5, [R7, #0xC]
26           MOV        R4, #1
27           SUB        R3, R4, R3           // Subtract R3 from 1 to invert it (1 <-> 0)
28  DELAY:    LDR        R4, [R9, #0xC]       // Load timer interrupt flag
29           CMP        R4, #0               // Keep on delaying until interrupt flag is 1
30           BEQ        DELAY
31           STR        R4, [R9, #0xC]       // Reset interrupt flag
32
33           CMP        R3, #1               // When R3 = 1, increment counter
34           BNE        DISPLAY
35           ADD        R2, #1
36           LDR        R4, =6000            // Load a literal
37           CMP        R2, R4               // Wrap around to 0 when R2 > 5999
38           BNE        DISPLAY
39           MOV        R2, #0
40
41  DISPLAY:  MOV        R0, R2               // Separate R2 into its digits
42           BL         DIVIDE
43           LDRB       R4, [R8, +R0]        // Get pattern for ones digit
44
45           MOV        R0, R1               // Get tens digit
46           BL         DIVIDE
47           LDRB       R0, [R8, +R0]        // Get pattern for tens digit
48           LSL        R0, #8
49           ORR        R4, R0
50
51           MOV        R0, R1               // Get hundredth digit
52           BL         DIVIDE               // Remainder from divide is thousandth digit
53           LDRB       R0, [R8, +R0]        // Get pattern for hundreds digit
54           LSL        R0, #16
55           ORR        R4, R0
56           LDRB       R1, [R8, +R1]        // Get pattern for thousandth digit
57           LSL        R1, #24
58           ORR        R4, R1
59
60           STR        R4, [R6]             // Display counter
61           B          MAIN                 // Program infinitely counts/loops
62
63
64  /* Subroutine to perform the integer division R0 / 10.
65  * Returns quotient in R1 and remainder in R0
66  */
67  DIVIDE:   PUSH       {R2,LR}
68           MOV        R2, #0

```

```

69  CONT:      CMP      R0, #10
70              BLT      DIV_END
71              SUB      R0, #10
72              ADD      R2, #1
73              B        CONT
74  DIV_END:    MOV      R1, R2          // quotient in R1 (remainder in R0)
75              POP      {R2,PC}
76
77  BIT_CODES:  .byte    0b00111111, 0b000000110, 0b01011011, 0b01001111, 0b01100110
78              .byte    0b01101101, 0b01111101, 0b000000111, 0b01111111, 0b01100111
79              .skip    2              // pad with 2 bytes to maintain word alignment
80

```