

```

1  /* Program that counts consecutive 1's and holds the length of the longest string */
2
3      .text                                // executable code follows
4      .global _start
5  _start:  MOV     R4, #TEST_NUM           // R4 will hold the address of the next data word
6          MOV     R5, #0                  // R5 will hold length of the string of ones
7  MAIN:    LDR     R1, [R4], #4            // R1 <- next word
8          CMP     R1, #0
9          BEQ     END                     // 0 indicates the end of the list
10         BL      ONES                    // Count longest string of 1's
11         CMP     R5, R0                  // Result is returned in 0
12         MOVLT   R5, R0                  // Store greater value in R5
13         B       MAIN                    // Keep looping until the list is done
14
15  END:     B       END
16
17  // Subroutine ONES to find longest string of ones in R1
18  // Result is returned in R0
19  ONES:    MOV     R0, #0                  // R0 will hold the result
20  LOOP:    CMP     R1, #0
21          BEQ     END_ONES                // loop until the data contains no more 1's
22          LSR     R2, R1, #1              // perform SHIFT, followed by AND
23          AND     R1, R1, R2
24          ADD     R0, #1                  // count the string length so far
25          B       LOOP
26  END_ONES: MOV     PC, LR                 // Return
27  // End of subroutine ONES
28
29  TEST_NUM: .word    0x103fe00f, 0x111ff332, 0x12345678
30            .word    0xaf428039, 0x724c8831, 0xa92ee391
31            .word    0xe0d4bd47, 0x8f8adad8, 0xdfa7ea48
32            .word    0xe99e1b93, 0xa4cc303b, 0xda87b4e7
33            .word    0
34
35            .end
36

```



```

1  /* Program that counts the longest string of
2     0's, 1's, and alternating 1's and 0's */
3     .text
4     .global _start
5
6  _start:    MOV     R4, #TEST_NUM        // R4 will hold the address of the next data word
7            MOV     R5, #0              // R5 will hold length of the string of 1's
8            MOV     R6, #0              // R6 will hold length of the string of 0's
9            MOV     R7, #0              // R7 will hold length of the string of
10           alternating 1's and 0's
11 MAIN:      LDR     R1, [R4]            // R1 <- next word
12            CMP     R1, #0              // 0 indicates the end of the list
13            BEQ     END                // Count longest string of 1's, passes in R1
14            BL      ONES                // Result is returned in R0
15            MOVLT   R5, R0              // Store greater value in R5
16            LDR     R1, [R4]            // R1 <- same word
17            BL      ZEROS                // Count longest string of 0's, passes in R1
18            CMP     R6, R0              // Result returned in R0
19            MOVLT   R6, R0              // Store greater value in R6
20            LDR     R1, [R4], #4         // R1 <- same word, R4 moves onto next word
21            BL      ALTS                // Count longest string of alternates, passes
22           in R1
23            CMP     R7, R0              // Result returned in R0
24            MOVLT   R7, R0              // Store greater value in R7
25            B       MAIN                // Keep looping until the list is done
26
27 END:        B       END
28
29 // Subroutine ONES to find longest string of 1's in R1
30 // Result is returned in R0
31 ONES:      PUSH    {R2,LR}             // Store used registers in stack
32            MOV     R0, #0              // R0 will hold the result
33 LOOP:      CMP     R1, #0              // loop until the data contains no more 1's
34            BEQ     END_ONES            // perform SHIFT, followed by AND
35            LSR     R2, R1, #1          //
36            AND     R1, R1, R2          //
37            ADD     R0, #1              // count the string length so far
38            B       LOOP
39 END_ONES:  POP     {R2,PC}             // Return
40 // End of subroutine ONES
41
42 // Subroutine ZEROS to find longest string of 0's in R1
43 // Result is returned in R0
44 // This can be done by complementing R1 and
45 // counting the longest string of 1's
46 ZEROS:     PUSH    {R2,LR}             // Store used registers in stack
47            MOV     R2, #ALL_F          // Put string of all 1's into R2
48            LDR     R2, [R2]
49            EOR     R1, R2              // Complement R1
50            BL      ONES                // Count longest string of 1's, passes in R1
51            POP     {R2,PC}             // Pop LR(from stack) into PC to return, R0 is
52           returned
53 // End of subroutine ZEROS
54
55 // Subroutine ALTS to find longest alternating string in R1
56 // Result is returned in R0
57 // This can be done by XOR-ing R1 with an alternating string of 1's and 0's
58 // and then counting the longest string of 1's as well as 0's and returning the max
59 ALTS:      PUSH    {R2,R3,R4,LR}       // Store used registers in stack
60            MOV     R4, #ALTERNATES
61            LDR     R4, [R4]            // Put string of alternating 1's and 0's into R4
62            MOV     R2, R1              // Store the initial value of R1 in R2 to be
63           used again later
64            EOR     R1, R4              // XOR R1 with alternating 1's and 0's
65            BL      ONES                // Count longest string of 1's, passes in R1
66            MOV     R3, R0              // Result returned in R0, store in R3 to
67           compare later
68            EOR     R1, R2, R4          // XOR R2 (initial R1) with alternating 1's and

```

```

65         0's
66         BL      ZEROS          // Count longest string of 0's, passes in R1
67         CMP     R0, R3        // Result returned in R0, put greater value in R0
68         MOVLTL  R0, R3
69         POP     {R2,R3,R4,PC} // Return
70 // End of subroutine ALTS
71
72 TEST_NUM: .word    0x103fe00f, 0x111ff332, 0x12345678
73           .word    0xaf428039, 0x724c8831, 0xa92ee391
74           .word    0xe0d4bd47, 0x8f8adad8, 0xdfa7ea48
75           .word    0xe99e1b93, 0xa4cc303b, 0xda87b4e7
76           .word    0
77 ALL_F:    .word    0xffffffff
78 ALTERNATES: .word  0xaaaaaaaa
79           .end
80
81 /* VALUES in binary:
82 00010000001111111110000000001111
83 00010001000111111111001100110010
84 00010010001101000101011001111000
85 10101111010000101000000000111001
86 01110010010011001000100000110001
87 10101001001011101110001110010001
88 11100000110101001011110101000111
89 10001111100010101101101011011000
90 11011111101001111110101001001000
91 11101001100111100001101110010011
92 10100100110011000011000000111011
93 11011010100001111011010011100111
94 */
95

```

```

1  /* Program that counts the longest string of
2  * 0's, 1's, and alternating 1's and 0's and
3  * displays the results on the 7 segments
4  */
5      .text
6      .global _start
7
8  _start:    MOV     R4, #TEST_NUM        // R4 will hold the address of the next data word
9            MOV     R5, #0              // R5 will hold length of the string of 1's
10           MOV     R6, #0              // R6 will hold length of the string of 0's
11           MOV     R7, #0              // R7 will hold length of the string of
alternating 1's and 0's
12
13  COUNT:    LDR     R1, [R4]            // R1 <- next word
14           CMP     R1, #0
15           BEQ     DISPLAY            // 0 indicates the end of the list
16           BL      ONES               // Count longest string of 1's, passes in R1
17           CMP     R5, R0             // Result is returned in 0
18           MOVLT   R5, R0             // Store greater value in R5
19           LDR     R1, [R4]            // R1 <- same word
20           BL      ZEROS              // Count longest string of 0's, passes in R1
21           CMP     R6, R0             // Result returned in R0
22           MOVLT   R6, R0             // Store greater value in R6
23           LDR     R1, [R4], #4        // R1 <- same word, R4 moves onto next word
24           BL      ALTS               // Count longest string of alternates, passes
in R1
25           CMP     R7, R0             // Result returned in R0
26           MOVLT   R7, R0             // Store greater value in R7
27           B       COUNT              // Keep looping until the list is done
28
29  // Convert R5, R6, and R7 to decimal and display on the 7 segments
30  // Display R5 on HEX1-0, R6 on HEX3-2 and R7 on HEX5-4
31  DISPLAY:  LDR     R8, =0xFF200020    // base address of HEX3-HEX0
32           MOV     R0, R5              // display R5 on HEX1-HEX0
33           BL      DIVIDE              // ones digit will be in R0; tens digit in R1
34           MOV     R9, R1              // save the tens digit
35           BL      SEG7_CODE
36           MOV     R4, R0              // save bit code
37           MOV     R0, R9              // retrieve the tens digit, get bit code
38           BL      SEG7_CODE
39           LSL     R0, #8
40           ORR     R4, R0
41
42           MOV     R0, R6              // Display R6 on HEX3-HEX2
43           BL      DIVIDE              // ones digit will be in R0; tens digit in R1
44           MOV     R9, R1              // save the tens digit
45           BL      SEG7_CODE
46           LSL     R0, #16
47           ORR     R4, R0              // save bit code
48           MOV     R0, R9              // retrieve the tens digit, get bit code
49           BL      SEG7_CODE
50           LSL     R0, #24
51           ORR     R4, R0
52
53           STR     R4, [R8]            // display the numbers from R6 and R5
54
55           LDR     R8, =0xFF200030    // base address of HEX5-HEX4
56           MOV     R0, R7              // display R5 on HEX1-HEX0
57           BL      DIVIDE              // ones digit will be in R0; tens digit in R1
58           MOV     R9, R1              // save the tens digit
59           BL      SEG7_CODE
60           MOV     R4, R0              // save bit code
61           MOV     R0, R9              // retrieve the tens digit, get bit code
62           BL      SEG7_CODE
63           LSL     R0, #8
64           ORR     R4, R0
65
66           STR     R4, [R8]            // display the number from R7
67

```

```

68  END:          B          END
69
70  /* Subroutine ONES to find longest string of 1's in R1
71  * Result is returned in R0
72  */
73  ONES:          PUSH      {R2,LR}          // Store used registers in stack
74                MOV       R0, #0          // R0 will hold the result
75  LOOP:          CMP       R1, #0
76                BEQ        END_ONES        // loop until the data contains no more 1's
77                LSR        R2, R1, #1      // perform SHIFT, followed by AND
78                AND        R1, R1, R2
79                ADD        R0, #1          // count the string length so far
80                B          LOOP
81  END_ONES:      POP        {R2,PC}         // Return
82  // End of subroutine ONES
83
84  /* Subroutine ZEROS to find longest string of 0's in R1
85  * Result is returned in R0
86  * This can be done by complementing R1 and
87  * counting the longest string of 1's
88  */
89  ZEROS:          PUSH      {R2,LR}          // Store used registers in stack
90                MOV       R2, #ALL_F      // Put string of all 1's into R2
91                LDR        R2, [R2]
92                EOR        R1, R2          // Complement R1
93                BL         ONES            // Count longest string of 1's, passes in R1
94                POP        {R2,PC}         // Pop LR(from stack) into PC to return, R0 is
// returned
95  // End of subroutine ZEROS
96
97  /* Subroutine ALTS to find longest alternating string in R1
98  * Result is returned in R0
99  * This can be done by XOR-ing R1 with an alternating string of 1's and 0's
100 * and then counting the longest string of 1's as well as 0's and returning the max
101 */
102  ALTS:           PUSH      {R2,R3,R4,LR}    // Store used registers in stack
103                MOV       R4, #ALTERNATES
104                LDR        R4, [R4]          // Put string of alternating 1's and 0's into R4
105                MOV       R2, R1            // Store the initial value of R1 in R2 to be
used again later
106                EOR        R1, R4          // XOR R1 with alternating 1's and 0's
107                BL         ONES            // Count longest string of 1's, passes in R1
108                MOV       R3, R0            // Result returned in R0, store in R3 to
compare later
109                EOR        R1, R2, R4      // XOR R2 (initial R1) with alternating 1's and
0's
110                BL         ZEROS           // Count longest string of 0's, passes in R1
111                CMP        R0, R3          // Result returned in R0, put greater value in R0
112                MOVLTP     R0, R3
113                POP        {R2,R3,R4,PC}    // Return
114  // End of subroutine ALTS
115
116  /* Subroutine to perform the integer division R0 / 10.
117  * Returns quotient in R1 and remainder in R0
118  */
119  DIVIDE:         MOV       R2, #0
120  CONT:           CMP       R0, #10
121                BLT        DIV_END
122                SUB        R0, #10
123                ADD        R2, #1
124                B          CONT
125  DIV_END:        MOV       R1, R2          // quotient in R1 (remainder in R0)
126                MOV       PC, LR
127
128  /* Subroutine to convert the digits from 0 to 9 to be shown on a HEX display.
129  * Parameters: R0 = the decimal value of the digit to be displayed
130  * Returns: R0 = bit pattern to be written to the HEX display
131  */
132  SEG7_CODE:      MOV       R1, #BIT_CODES

```

```

133         ADD      R1, R0          // index into the BIT_CODES "array"
134         LDRB     R0, [R1]        // load the bit pattern (to be returned)
135         MOV      PC, LR
136
137     // Data
138     TEST_NUM:    .word    0x103fe00f, 0x111ff332, 0x12345678
139                  .word    0xaf428039, 0x724c8831, 0xa92ee391
140                  .word    0xe0d4bd47, 0x8f8adad8, 0xdfa7ea48
141                  .word    0xe99e1b93, 0xa4cc303b, 0xda87b4e7
142                  .word    0
143
144     ALL_F:       .word    0xffffffff
145     ALTERNATES: .word    0xaaaaaaaa
146
147     BIT_CODES:   .byte    0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
148                  .byte    0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
149                  .skip    2          // pad with 2 bytes to maintain word alignment
150
151     DIGITS:      .space    6          // Allocate space for 6 digits
152
153                 .end
154
155     /* VALUES in binary:
156     000100000011111111110000000001111
157     00010001000111111111001100110010
158     00010010001101000101011001111000
159     1010111101000010100000000111001
160     01110010010011001000100000110001
161     10101001001011101110001110010001
162     11100000110101001011110101000111
163     10001111100010101101101011011000
164     11011111101001111110101001001000
165     11101001100111100001101110010011
166     10100100110011000011000000111011
167     11011010100001111011010011100111
168     */
169

```