```
/* Program that counts the longest string of
 * 0's, 1's, and alternating 1's and 0's and
 * displays the results on the 7 segments
 */
            .text
            .global _start

_start:     MOV     R4, #TEST_NUM       // R4 will hold the address of the next data word
            MOV     R5, #0              // R5 will hold length of the string of 1's
            MOV     R6, #0              // R6 will hold length of the string of 0's
            MOV     R7, #0              // R7 will hold length of the string of
            alternating 1's and 0's

COUNT:      LDR     R1, [R4]            // R1 <- next word
            CMP     R1, #0
            BEQ     DISPLAY             // 0 indicates the end of the list
            BL      ONES                // Count longest string of 1's, passes in R1
            CMP     R5, R0              // Result is returned in 0
            MOVLT   R5, R0              // Store greater value in R5
            LDR     R1, [R4]            // R1 <- same word
            BL      ZEROS               // Count longest string of 0's, passes in R1
            CMP     R6, R0              // Result returned in R0
            MOVLT   R6, R0              // Store greater value in R6
            LDR     R1, [R4], #4        // R1 <- same word, R4 moves onto next word
            BL      ALTS                // Count longest string of altenrates, passes
            in R1
            CMP     R7, R0              // Result returned in R0
            MOVLT   R7, R0              // Store greater value in R7
            B       COUNT               // Keep looping until the list is done

// Convert R5, R6, and R7 to decimal and display on the 7 segments
// Display R5 on HEX1-0, R6 on HEX3-2 and R7 on HEX5-4
DISPLAY:    LDR     R8, =0xFF200020     // base address of HEX3-HEX0
            MOV     R0, R5              // display R5 on HEX1-HEX0
            BL      DIVIDE              // ones digit will be in R0; tens digit in R1
            MOV     R9, R1              // save the tens digit
            BL      SEG7_CODE
            MOV     R4, R0              // save bit code
            MOV     R0, R9              // retrieve the tens digit, get bit code
            BL      SEG7_CODE
            LSL     R0, #8
            ORR     R4, R0

            MOV     R0, R6              // Display R6 on HEX3-HEX2
            BL      DIVIDE              // ones digit will be in R0; tens digit in R1
            MOV     R9, R1              // save the tens digit
            BL      SEG7_CODE
            LSL     R0, #16
            ORR     R4, R0              // save bit code
            MOV     R0, R9              // retrieve the tens digit, get bit code
            BL      SEG7_CODE
            LSL     R0, #24
            ORR     R4, R0

            STR     R4, [R8]        // display the numbers from R6 and R5

            LDR     R8, =0xFF200030 // base address of HEX5-HEX4
            MOV     R0, R7              // display R5 on HEX1-HEX0
            BL      DIVIDE              // ones digit will be in R0; tens digit in R1
            MOV     R9, R1              // save the tens digit
            BL      SEG7_CODE
            MOV     R4, R0              // save bit code
            MOV     R0, R9              // retrieve the tens digit, get bit code
            BL      SEG7_CODE
            LSL     R0, #8
            ORR     R4, R0

            STR     R4, [R8]        // display the number from R7

```

```
END:        B       END

/* Subroutine ONES to find longest string of 1's in R1
 * Result is returned in R0
 */
ONES:       PUSH    {R2,LR}             // Store used registers in stack
            MOV     R0, #0              // R0 will hold the result
LOOP:       CMP     R1, #0
            BEQ     END_ONES            // loop until the data contains no more 1's
            LSR     R2, R1, #1          // perform SHIFT, followed by AND
            AND     R1, R1, R2
            ADD     R0, #1              // count the string length so far
            B       LOOP
END_ONES:   POP     {R2,PC}             // Return
// End of subroutine ONES

/* Subroutine ZEROS to find longest string of 0's in R1
 * Result is returned in R0
 * This can be done by complementing R1 and
 * counting the longest string of 1's
 */
ZEROS:      PUSH    {R2,LR}             // Store used registes in stack
            MOV     R2, #ALL_F          // Put string of all 1's into R2
            LDR     R2, [R2]
            EOR     R1, R2              // Complement R1
            BL      ONES                // Count longest string of 1's, passes in R1
            POP     {R2,PC}             // Pop LR(from stack) into PC to return, R0 is
            returned
// End of subroutine ZEROS

/* Subroutine ALTS to find longest alternating string in R1
 * Result is returned in R0
 * This can be done by XOR-ing R1 with an alternating string of 1's and 0's
 * and then counting the longest string of 1's as well as 0's and returning the max
 */
ALTS:       PUSH    {R2,R3,R4,LR}       // Store used registers in stack
            MOV     R4, #ALTERNATES
            LDR     R4, [R4]            // Put string of alternating 1's and 0's into R4
            MOV     R2, R1              // Store the initial value of R1 in R2 to be
            used again later
            EOR     R1, R4              // XOR R1 with alternating 1's and 0's
            BL      ONES                // Count longest string of 1's, passes in R1
            MOV     R3, R0              // Result returned in R0, store in R3 to
            compare later
            EOR     R1, R2, R4          // XOR R2 (initial R1) with alternating 1's and
            0's
            BL      ZEROS               // Count longest string of 0's, passes in R1
            CMP     R0, R3              // Result returned in R0, put greater value in R0
            MOVLT   R0, R3
            POP     {R2,R3,R4,PC}       // Return
// End of subroutine ALTS

/* Subroutine to perform the integer division R0 / 10.
 * Returns quotient in R1 and remainder in R0
 */
DIVIDE:     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)
            MOV     PC, LR

/* Subroutine to convert the digits from 0 to 9 to be shown on a HEX display.
 * Parameters: R0 = the decimal value of the digit to be displayed
 * Returns: R0 = bit pattern to be written to the HEX display
 */
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 00010000001111111110000000001111
157 00010001000111111111001100110010
158 00010010001101000101011001111000
159 10101111010000101000000000111001
160 01110010010011001000100000110001
161 10101001001011101110001110010001
162 11100000110101001011110101000111
163 10001111100010101101101011011000
164 11011111101001111110101001001000
165 11101001100111100001101110010011
166 10100100110011000011000000111011
167 11011010100001111011010011100111
168 */
169
```