```
.global _start

// Roulette Game for DE1-SoC board
// How it works: pick a LED with switches, watch them spin, press KEY0 to stop
// If your LED is the one that stops, you win and see "YES", otherwise "END"
// The final LED number shows up on HEX3

_start:
    // First, let's see which switch the user picked
    LDR R1, =0xFF200040        // Switch base address
    LDR R2, [R1]               // Read the switches
    MOV R3, R2                 // Keep a copy of user's bet

    // Set up the LEDs
    LDR R4, =0xFF200000        // LED base address
    MOV R5, #1                 // Start with just LED0 on

    // We'll need to check the buttons too
    LDR R10, =0xFF200050       // Button base address

spin_loop:
    STR R5, [R4]               // Light up current LED
    BL delay                   // Wait a bit so we can see the spin
    LSL R5, R5, #1             // Move to next LED (shift left)
    CMP R5, #1024              // Did we go past LED9?
    BNE skip_reset
    MOV R5, #1                 // Back to LED0 if we did
skip_reset:
    LDR R11, [R10]             // Check if any button is pressed
    TST R11, #1                // Is KEY0 pressed?
    BEQ spin_loop              // Nope, keep spinning

    // OK, user stopped the spin - save where we landed
    MOV R7, R5                 // Final LED position

    // Did they win? Check if their switch matches the LED
    TST R3, R7                 // Bitwise AND to see if they match
    BEQ show_end               // No match = they lost
    B show_win                 // Match = they won!

// Player won - show YES and which LED it was
show_win:
    BL display_yes             // Put "YES" on the display
```

```
    BL show_led_number          // Show the winning LED number
    B done

// Player lost - show END and which LED it was
show_end:
    BL display_end              // Put "END" on the display
    BL show_led_number          // Show the LED number anyway
    B done

// Just a simple delay to make the spinning visible
delay:
    MOV R8, #0x8000             // Some big number to count down
loop_delay:
    SUBS R8, R8, #1             // Count down
    BNE loop_delay              // Keep going until zero
    BX LR                       // Return

// Display "YES" on HEX2, HEX1, HEX0
// Had to look up the 7-segment codes: Y=0x6E, E=0x79, S=0x6D
display_yes:
    LDR R0, =0xFF200020         // HEX display address
    MOV R1, #0x6E               // Y pattern
    LSL R1, R1, #8              // Shift for next digit
    ORR R1, R1, #0x79           // Add E pattern
    LSL R1, R1, #8              // Shift again
    ORR R1, R1, #0x6D           // Add S pattern
    STR R1, [R0]                // Send it to the display
    BX LR

// Display "END" on HEX2, HEX1, HEX0
// 7-segment codes: E=0x79, N=0x37, D=0x5E
display_end:
    LDR R0, =0xFF200020         // HEX display address
    MOV R1, #0x79               // E pattern
    LSL R1, R1, #8              // Make room for N
    ORR R1, R1, #0x37           // Add N pattern
    LSL R1, R1, #8              // Make room for D
    ORR R1, R1, #0x5E           // Add D pattern
    STR R1, [R0]                // Write to display
    BX LR

// Figure out which LED number (0-9) and show it on HEX3
show_led_number:
    MOV R6, #0                  // Start counting from 0
```

```
    MOV R12, R7                 // Copy the LED value

count_loop:
    TST R12, #1                 // Is the rightmost bit set?
    BEQ shift_right             // No, keep looking
    B display_led_index         // Yes, found our LED number
shift_right:
    LSR R12, R12, #1            // Shift right to check next bit
    ADD R6, R6, #1              // Increment our counter
    CMP R6, #10                 // Shouldn't go past 9
    BLT count_loop              // Keep looking
    B display_led_index

display_led_index:
    LDR R0, =digit_table        // Get our lookup table
    LDR R1, [R0, R6, LSL #2]    // Get the 7-segment pattern for this digit

    LDR R2, =0xFF200020         // HEX display base
    LDR R3, [R2]                // Read what's currently there
    BIC R3, R3, #(0xFF << 24)   // Clear out HEX3 (top 8 bits)
    ORR R3, R3, R1, LSL #24     // Put our digit in HEX3
    STR R3, [R2]                // Update the display
    BX LR

// 7-segment patterns for digits 0-9
// These control which segments light up to make each number
.align 2
digit_table:
    .word 0x3F      // 0
    .word 0x06      // 1
    .word 0x5B      // 2
    .word 0x4F      // 3
    .word 0x66      // 4
    .word 0x6D      // 5
    .word 0x7D      // 6
    .word 0x07      // 7
    .word 0x7F      // 8
    .word 0x6F      // 9

// That's it - just loop forever here
done:
    B done          // Infinite loop to end the program
```