.global \_start // Declare the \_start symbol as globally accessible

//// Define the timer base address as a constant with value 0xFFFEC600 if you execute it in CPUlator

.equ LED\_ADDR, 0xFF200020 // Define the LED address as a constant with value 0xFF200020

.equ LED\_ADDR2, 0xFF200030 // Define the second LED address as a constant with value 0xFF200030

.equ SW\_ADDR, 0xFF200040 // Define the switch address as a constant with value 0xFF200040

.equ TIMERBASE, 0xFFC08000 // Define the timer base address (of DE1SOC ) as a constant with value 0XFFC08000

.equ PUSHBUTTON\_BASE, 0xFF200050 // Define the pushbutton base address as a constant with value 0xFF200050

\_start:

BL WaitForStart // Call WaitForStart subroutine

loop:

MOV R8, #0 // Initialize register R8 to 0

MOV R9, #0 // Initialize register R9 to 0, used for keeping score

LDR R0, =0x3B9ACA00 // Load the value 0x3B9ACA00 into R0 (5 seconds)

LDR R1, =TIMERBASE // Load the timer base address into R1

STR R0, [R1] // Store the value in R0 into the memory location pointed to by R1 (start timer)

LDR R10, =HEXTABLE // Load the address of HEXTABLE into R10

LDR R4, =LED\_ADDR // Load the LED address into R4

Start\_Timer:

MOV R0, #0b11 // Load the value 3 (binary 11) into R0

STR R0, [R1, #8] // Store the value in R0 into the timer control register

BL Clear\_Display // Call Clear\_Display subroutine

BL Display\_Number // Call Display\_Number subroutine

BL Delay // Call Delay subroutine

BL Clear\_Display // Call Clear\_Display subroutine again

BL GetUserInput // Call GetUserInput subroutine

Display\_Number:

PUSH {R6} // Save register R6 on the stack

LDR R3, [R1, #4] // Load the random number from the timer into R3

AND R3, R3, #15 // Mask the last 4 bits of R3

LDR R6, [R10, R3, LSL #2] // Load the corresponding value from HEXTABLE into R6

STR R6, [R4] // Store the value in R6 into the LED address

POP {R6} // Restore register R6

BX LR // Return from subroutine

Clear\_Display:

PUSH {R2, LR} // Save registers R2 and LR on the stack

MOV R2, #0 // Set R2 to 0 (clear value)

STR R2, [R4, #0] // Clear the LED display at offset 0

STR R2, [R4, #4] // Clear the LED display at offset 4

STR R2, [R4, #8] // Clear the LED display at offset 8

STR R2, [R4, #12] // Clear the LED display at offset 12

STR R2, [R4, #16] // Clear the LED display at offset 16

POP {R2, LR} // Restore registers R2 and LR

BX LR // Return from subroutine

Delay:

PUSH {R0, LR} // Save registers R0 and LR on the stack

LDR R0, =2000000000 // Load the delay value into R0

Delay\_Loop:

SUBS R0, R0, #1 // Subtract 1 from R0 and update flags

BNE Delay\_Loop // If R0 is not zero, branch to Delay\_Loop

POP {R0, LR} // Restore registers R0 and LR

BX LR // Return from subroutine

GetUserInput:

PUSH {R7} // Save register R7 on the stack

LDR R1, =SW\_ADDR // Load the switch address into R1

LDR R7, [R1] // Load the value from the switch into R7

AND R7, R7, #0xF // Mask the last 4 bits of R7

CompareNumbers:

CMP R7, R3 // Compare the value in R7 with R3

POP {R7} // Restore register R7

BEQ COUNTER // If equal, branch to COUNTER

BNE Incorrect // If not equal, branch to Incorrect

COUNTER:

ADD R6, R6, #1 // Increment R6

CMP R6, #5 // Compare R6 with 5

BEQ Correct // If equal, branch to Correct

BNE loop // If not equal, branch to loop

Correct:

PUSH {R3, R4, R8, R11, LR} // Save registers R3, R4, R8, R11, and LR on the stack

LDR R11, =WIN // Load the address of WIN into R11

LDRB R3, [R11, #0] // Load the first byte of WIN into R3

ORR R8, R8, R3, LSL #0 // OR the value in R3 into R8

LDRB R3, [R11, #1] // Load the second byte of WIN into R3

ORR R8, R8, R3, LSL #8 // OR the value in R3 into R8 shifted left by 8

LDRB R3, [R11, #2] // Load the third byte of WIN into R3

ORR R8, R8, R3, LSL #16 // OR the value in R3 into R8 shifted left by 16

STR R8, [R4] // Store the value in R8 into the LED address

ADD R9, R9, #1 // Increment the score

BL Delay // Call Delay subroutine

POP {R3, R4, R8, R11, LR} // Restore registers R3, R4, R8, R11, and LR

MOV R6 , #0 // Reset the counter after 1 WIN

B loop // Branch to loop

Incorrect:

PUSH {R3, R4, R8, R11, LR} // Save registers R3, R4, R8, R11, and LR on the stack

MOV R6 , #0 // Reset the counter after LOSE

LDR R11, =LOST // Load the address of LOST into R11

LDRB R3, [R11, #0] // Load the first byte of LOST into R3

ORR R8, R8, R3, LSL #24 // OR the value in R3 into R8 shifted left by 24

LDRB R3, [R11, #1] // Load the second byte of LOST into R3

ORR R8, R8, R3, LSL #16 // OR the value in R3 into R8 shifted left by 16

LDRB R3, [R11, #2] // Load the third byte of LOST into R3

ORR R8, R8, R3, LSL #8 // OR the value in R3 into R8 shifted left by 8

LDRB R3, [R11, #3] // Load the fourth byte of LOST into R3

ORR R8, R8, R3, LSL #0 // OR the value in R3 into R8

LDRB R3, [R11, #4] // Load the fifth byte of LOST into R3

LDR R5, =LED\_ADDR2 // Load the second LED address into R5

STR R3, [R5] // Store the value in R3 into the second LED address

STR R8, [R4] // Store the value in R8 into the LED address

BL Delay // Call Delay subroutine

POP {R3, R8, R4, R11, LR} // Restore registers R3, R8, R4, R11, and LR

B loop // Branch to loop

WaitForStart:

LDR R2, =PUSHBUTTON\_BASE // Load the pushbutton base address into R2

WaitLoop:

LDR R3, [R2] // Load the pushbutton value into R3

TST R3, #0x1 // Test if the first pushbutton is pressed

BEQ WaitLoop // If not pressed, branch to WaitLoop

BX LR // Return from subroutine

END:

B END // Infinite loop

.data

.align 4

WIN:

.byte 0b01010100 // Byte representation of 'W'

.byte 0b00000110 // Byte representation of 'I'

.byte 0b01001111 // Byte representation of 'N'

.align 4

LOST:

.byte 0b00111111 // Byte representation of 'O'

.byte 0b01101101 // Byte representation of 'S'

.byte 0b01111001 // Byte representation of 'E'

.byte 0b01010000 // Byte representation of 'R'

.byte 0b00111000 // Byte representation of 'L'

.align 4

HEXTABLE:

.word 0b00111111 // Byte representation of '0'

.word 0b00000110 // Byte representation of '1'

.word 0b01011011 // Byte representation of '2'

.word 0b01001111 // Byte representation of '3'

.word 0b01100110 // Byte representation of '4'

.word 0b01101101 // Byte representation of '5'

.word 0b01111101 // Byte representation of '6'

.word 0b00000111 // Byte representation of '7'

.word 0b01111111 // Byte representation of '8'

.word 0b01101111 // Byte representation of '9'

.word 0b01110111 // Byte representation of 'A'

.word 0b01111100 // Byte representation of 'B'

.word 0b00111001 // Byte representation of 'C'

.word 0b01011110 // Byte representation of 'D'

.word 0b01111001 // Byte representation of 'E'

.word 0b01110001 // Byte representation of 'F'