Name: Shahin Saeidi

Student ID: 251324723

ECE9047_Lab1

The code is written in assembly language and its purpose is to display different digits on a 7-segment display according to the value stored in register r2.

```
.global _start @where the program will begin execution


.text @part of the text segment, which contains executable code
```

The program begins execution at the label _start and starts by loading the values 11, 0, and 1 into registers r4, r0, and r1 respectively. It then loads the value 0x3f, which represents the digit 0 on a 7-segment display, into register r3.

```
_start: @beginning of the code

        mov r4, #11

        mov r0, #0

        mov r1, #1

        ldr r3, =0x3f @This loads the value 0x3f (which represents the digit 0 on

                      @a 7-segment display) into register r3

        ldr r12, =0xff200020
```

The code then enters a loop that compares the value in register r4 to the value 1. If they are equal, it branches (jumps) back to the label _start. Otherwise, it continues to the next instruction. Inside the loop, the code adds the values in registers r0 and r1 and stores the result in register r2. It then moves the value in register r1 to register r0 and the value in register r2 to register r1.

```
loop:

        cmp r4, #1

        beq _start

        @This compares the value in register r4 to the value 1. If they are equal,

        @it branches (jumps) to the label _start. Otherwise,

        @it continues to the next instruction.

        add r2, r1, r0

        mov r0, r1
        mov r1, r2
```

The code then enters a series of conditional statements that compare the value in register r2 to different constants. If the value in register r2 is greater than the constant, it branches to the corresponding "then" label and loads the appropriate value into register r3 to display the correct digit on the 7-segment display.

```
if: cmp r2, #1
        bhi elif1
then1: ldr r3, =0x3f @display the digit 0 on a 7-segment display
        b store


elif1: cmp r2, #2


        bhi elif2


then2: ldr r3, =0x6 @display the digit 1 on a 7-segment display
        b store


elif2: cmp r2, #3


        bhi elif3
then3: ldr r3, =0x5b @display the digit 2 on a 7-segment display
        b store


elif3: cmp r2, #5


        bhi elif4
then4: ldr r3, =0x4f @display the digit 3 on a 7-segment display
        b store


elif4: cmp r2, #8
```

```
            bhi elif5
then5: ldr r3, =0x6d @display the digit 5 on a 7-segment display
            b store

elif5: cmp r2, #13

            bhi elif6
then6: ldr r3, =0x7f @display the digit 8 on a 7-segment display
            b store

elif6: cmp r2, #21

            bhi elif7
then7: ldr r3, =0x64f @display the digit 13 on a 7-segment display
            b store

elif7: cmp r2, #34

            bhi elif8
then8: ldr r3, =0x5b06 @display the digit 21 on a 7-segment display
            b store

elif8: cmp r2, #55

            bhi else
then9: ldr r3, =0x4f66 @display the digit 34 on a 7-segment display
            b store
```

```
else: ldr r3, =0x6d6d @display the digit 55 on a 7-segment display
```

After loading the appropriate value into register r3, the code branches to the label "store", which stores the value in register r3 into the memory address specified by the constant 0xff200020. It then subtracts 1 from the value in register r4 and calls the subroutine "delay" using the branch with link (bl) instruction. The delay subroutine implements a pause in the program.

```
store: str r3, [r12]

        sub r4, #1

        bl delay @Branches to the "delay" subroutine, which is responsible for

                 @implementing the pause

        b loop @Branches back to the beginning of the "loop" subroutine to

               @continue execution



adress_timer_inval: .word time_inval
```

The delay subroutine starts by pushing the values in registers r4, r5, r8, and r9 onto the stack. It then loads the value 0xff202000 into register r2 and the value stored in the constant "time_inval" into register r9. It loads the value stored in the memory address pointed to by r9 into register r3 and stores it in the memory address pointed to by r2 plus 8. It then shifts the value in register r3 16 bits to the right and stores the result back into r3. The subroutine then stores the value of 4 into the memory address pointed to by r2 plus 4.

```
delay:

        push {r4,r5,r8,r9}

        ldr r2, =0xff202000

        ldr r9, adress_timer_inval

        ldr r3, [r9]

        str r3 , [r2 , #8]

        mov r3 , r3 , lsr #16 @Shifts the value in register r3 16 bits to the right

                              @and stores the result back into r3

        str r3 , [r2 , #12] @Stores the value of r3 into the memory address pointed to

                            @by register r2 plus 12

        mov r3 , #4

        str r3 , [r2 , #4]
```

The subroutine then enters a loop that stores the value in register r4 into the memory address pointed to by r2 plus 16, loads the value from that memory address into r4, loads the value from the memory address pointed to by r2 plus 20 into r5, adds the values in registers r4 and r5 left-shifted by 16 bits, and compares the result to 0. If the result is not equal to 0, the loop continues. Otherwise, the subroutine pops the values from the stack into registers r4, r5, r8, and r9 and branches back to the return address stored in the link register to return from the subroutine.

```
loop2:
        str r4 , [r2 , #16]
        ldr r4 , [r2 , #16]
        ldr r5 , [r2 , #20]
        add r4, r5, lsl #16
        cmp r4 , #0
        bne loop2
        pop {r4,r5,r8,r9}
        bx lr @Branches to the return address stored in the link register to return
              @from the subroutine
```

Finally, the program ends with a data section that defines the constant "time_inval" as the value 100000000.

```
.data
time_inval: .word 100000000
.end @Ends the program
```

Overall, the program implements a simple algorithm to display different digits on a 7-segment display using conditional statements and a delay subroutine.