Andrew Friedman
Lab 8 Report
ECE 2031 CS
12 July 2023

```
1   ; step 2 - pre.asm
2   ; MULTIPLIES 13 BY 20 TWICE AND STORES RESULTS THEN LOOPS INDEFINITELY
3   ; Andrew Friedman
4   ; ECE 2031 CS
5   ; 07/11/2023
6
7   ; init program
8   ORG 0
9     LOADI 13      ; init value
10    CALL Mul20    ; AC *= 20
11    STORE Result1   ; setup output first output
12    CALL Mul20    ; AC *= 20
13    STORE Result2   ; setup output first output
14
15  ; end loop
16  Finish:
17    JUMP Finish   ; inf looper
18
19  ; AC * 20 function
20  Mul20:
21    STORE CURR    ; boot temp variable
22      SHIFT 4     ; *= 16
23      ADD CURR    ; += curr
24      ADD CURR    ; += curr
25      ADD CURR    ; += curr
26      ADD CURR    ; += curr
27      RETURN      ; goto origin
28
29  ; temp & output variables
30  CURR: DW 0      ; Mul20 func temp variable
31  Result1: DW 0   ; AC * 20
32  Result2: DW 0   ; AC * 400
```

Figure 1. SCOMP Assembly Code implementing a subroutine for multiplying the AC by 20. The code executes this operation twice, storing the resulting values each time, before entering an endless loop upon successful execution.

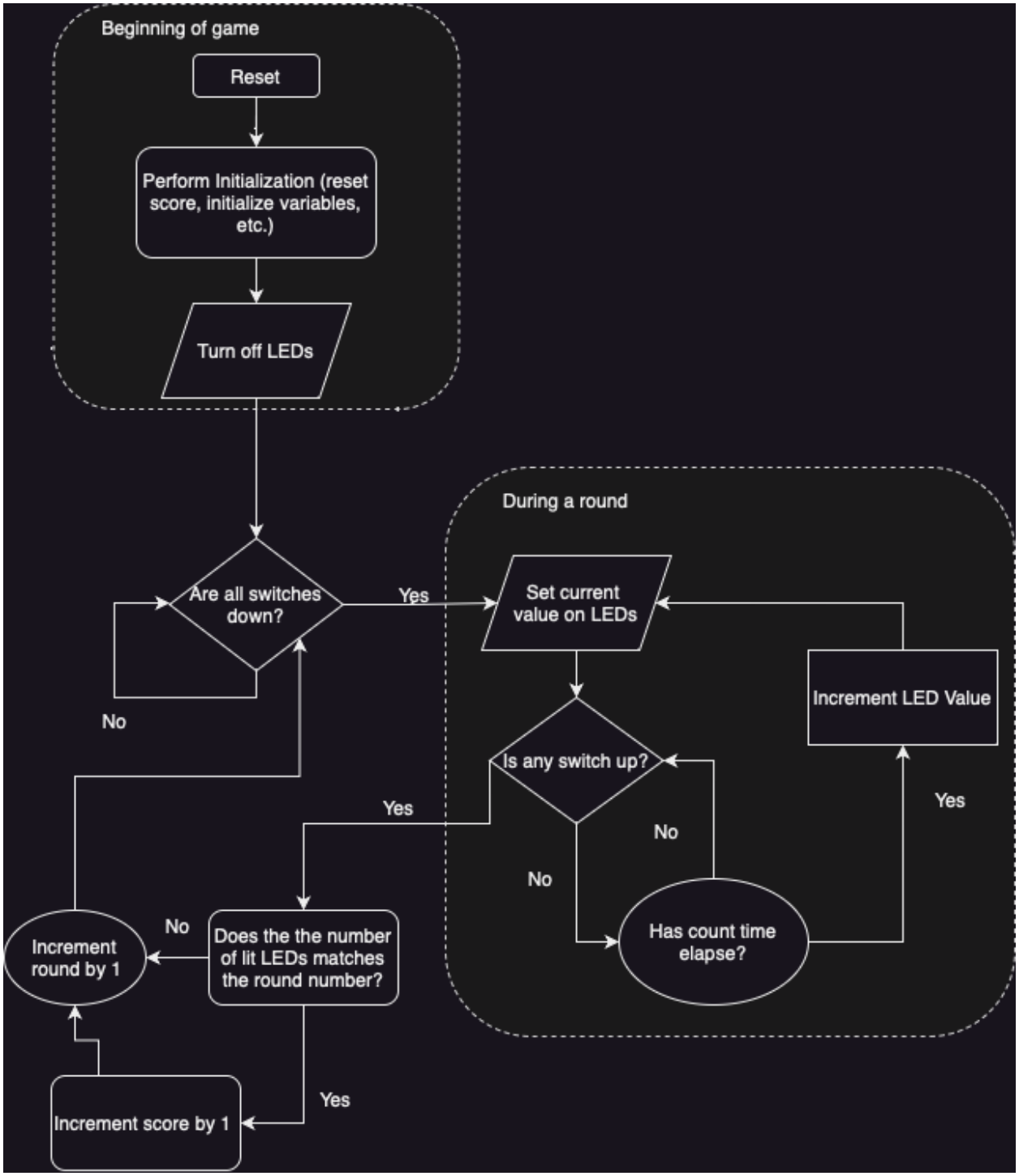

Figure 2. Digitally-produced flowchart demonstrating the game program flow for the SCOMP system. The flowchart visually presents each step in the game's logic, the conditions required for transitions between game states, and the user interactions necessary for the game's progress.

APPENDIX A
ASSEMBLY CODE IMPLEMENTING A MOVING LIGHT FROM SWITCH INPUTS

```
; IODemo.asm
; Produces a "bouncing" animation on the LEDs.
; The LED pattern is initialized with the switch state.
; Andrew Friedman
; ECE 2031 CS
; 07/11/2023

ORG 0

        ; Get and store the switch values
        IN     Switches
    STORE  PatternInit          ; save initial switch values
        OUT    LEDs
        STORE  Pattern

Left:
        ; Slow down the loop so humans can watch it.
        CALL   Delay

        ; Check if the left place is 1 and if so, switch direction
        CALL   SwitchFlip               ; check for changes in switches state
        LOAD   Pattern
        AND    Bit9                     ; bit mask
        JPOS   Right                    ; bit9 is 1; go right

        LOAD   Pattern
        SHIFT  1
        STORE  Pattern
        OUT    LEDs

        JUMP   Left

Right:
        ; Slow down the loop so humans can watch it.
        CALL   Delay

        ; Check if the right place is 1 and if so, switch direction
        CALL   SwitchFlip               ; check for channges in switches state
        LOAD   Pattern
        AND    Bit0                     ; bit mask
        JPOS   Left                     ; bit0 is 1; go left

        LOAD   Pattern
        SHIFT  -1
```

```
        STORE  Pattern
        OUT    LEDs

        JUMP   Right

; To make things happen on a human timescale, the timer is
; used to delay for half a second.
Delay:
        OUT    Timer
WaitingLoop:
        IN     Timer
        ADDI   -5
        JNEG   WaitingLoop
        RETURN

SwitchFlip:
        IN     Switches          ; Get current state of switches
        XOR    PatternInit       ; Compare current and initial switch state
        JZERO  Break             ; If 0, switch states are the same and break
        IN     Switches          ; read switches again
        STORE  PatternInit       ; initial state = new state (for later comparisons)
        STORE  Pattern           ; current LED pattern = new state
        OUT    LEDs              ; update LED state
Break: RETURN

; Variables
Pattern:   DW 0
PatternInit:   DW &B0000000000

; Useful values
Bit0:      DW &B0000000001
Bit9:      DW &B1000000000

; IO address constants
Switches:  EQU 000
LEDs:      EQU 001
Timer:     EQU 002
Hex0:      EQU 004
Hex1:      EQU 005
```

APPENDIX B
ASSEMBLY CODE IMPLEMENTING A GAME TO ‘CATCH’ LIGHTS BASED ON ROUND NUMBER

```
; step 5.asm
; ASSEMBLY FOR LIGHT CHASING GAME
; Andrew Friedman
; ECE 2031 CS
; 07/11/2023

ORG 0
Init:
        LOADI 0                         ; Load immediate value 0 into the accumulator
        STORE Initial                   ; Store accumulator value in the Initial variable
        STORE LEDDisp                   ; Initialize LEDDisp to 0
        OUT LEDs                        ; Output accumulator value to LEDs
        LOADI 0                         ; Load immediate value 0 into the accumulator
        STORE Score                     ; Initialize Score to 0
        OUT Hex0                        ; Output accumulator value to Hex0
        LOADI 1                         ; Load immediate value 1 into the accumulator
        STORE Round                     ; Initialize Round to 1
        OUT    Hex1                     ; Output accumulator value to Hex1

SwitchHold:
        IN Switches                     ; Input from switches
        OR Initial                      ; Or the input value with Initial
        JZERO Game                      ; If the result is zero, jump to Game
        JUMP SwitchHold                 ; If not, continue polling SwitchHold

Game:
        LOAD LEDDisp                    ; Load LEDDisp into the accumulator
        OUT LEDs                        ; Output accumulator value to LEDs
        Call Delay                      ; Call the Delay subroutine
        IN Switches                     ; Input from switches
        JZERO NextLED                   ; If the input is zero, jump to NextLED

        LOAD LEDDisp                    ; Load LEDDisp into the accumulator
        Call Count1s                    ; Call the Count1s subroutine
        SUB Round                       ; Subtract Round from accumulator
        JPOS Next                       ; If the result is positive, jump to Next
        JNEG Next                       ; If the result is negative, jump to Next

        Win:
                LOAD Score              ; Load Score into the accumulator
                ADDI 1                  ; Increment the accumulator
                STORE Score             ; Store accumulator value in the Score variable
        Next:
                LOAD Round          ; Load Round into the accumulator
```

```
                ADDI 1                  ; Increment the accumulator
                STORE Round             ; Store accumulator value in the Round variable
                OUT Hex1                ; Output accumulator value to Hex1
                LOAD Score              ; Load Score into the accumulator
                OUT Hex0                ; Output accumulator value to Hex0
                LOAD Initial            ; Load Initial into the accumulator
                STORE LEDDisp           ; Store accumulator value in the LEDDisp variable
                JUMP SwitchHold         ; Jump back to SwitchHold

NextLED:
        LOAD LEDDisp                    ; Load LEDDisp into the accumulator
        ADDI &B0000000001               ; Increment the accumulator
        STORE LEDDisp                   ; Store accumulator value in the LEDDisp variable
        JUMP Game                       ; Jump back to Game


; Delay func
Delay:
        OUT Timer
WaitingLoop:
        IN Timer
        ADDI -7
        JNEG WaitingLoop
        RETURN

; Count1s - Subroutine to count the number
; of 1s in the accumulator.  Result is
; returned in accumulator.
; Does not work if MSb is 1.
Count1s:
  STORE  C1Val    ; Save AC for later
  LOADI  0        ; Reset count variable
  STORE  C1Count
  LOAD   C1Val
  JNEG   C1Ret    ; Return if MSb is 1
C1Loop:
  AND    C1One    ; Mask LSb
  ADD    C1Count  ; Add to count
  STORE  C1Count  ; Store new count
  LOAD   C1Val
  SHIFT  -1       ; Shift to the right
  STORE  C1Val
  JPOS   C1Loop   ; Loop until 0
C1Ret:
```

```
  LOAD   C1Count  ; The value to return
  RETURN
C1One:   DW 1
C1Val:   DW 0
C1Count: DW 0

; Game variables
Round: DW 0
Score: DW 0
Initial: DW &B0000000000
LEDDisp: DW &B0000000000

; Constant values for IO
Switches: EQU 000
LEDs: EQU 001
Timer: EQU 002
Hex0: EQU 004
Hex1: EQU 005
```