Rudra Goel
Lab 08 Report
ECE 2031 L10
25 October 2024

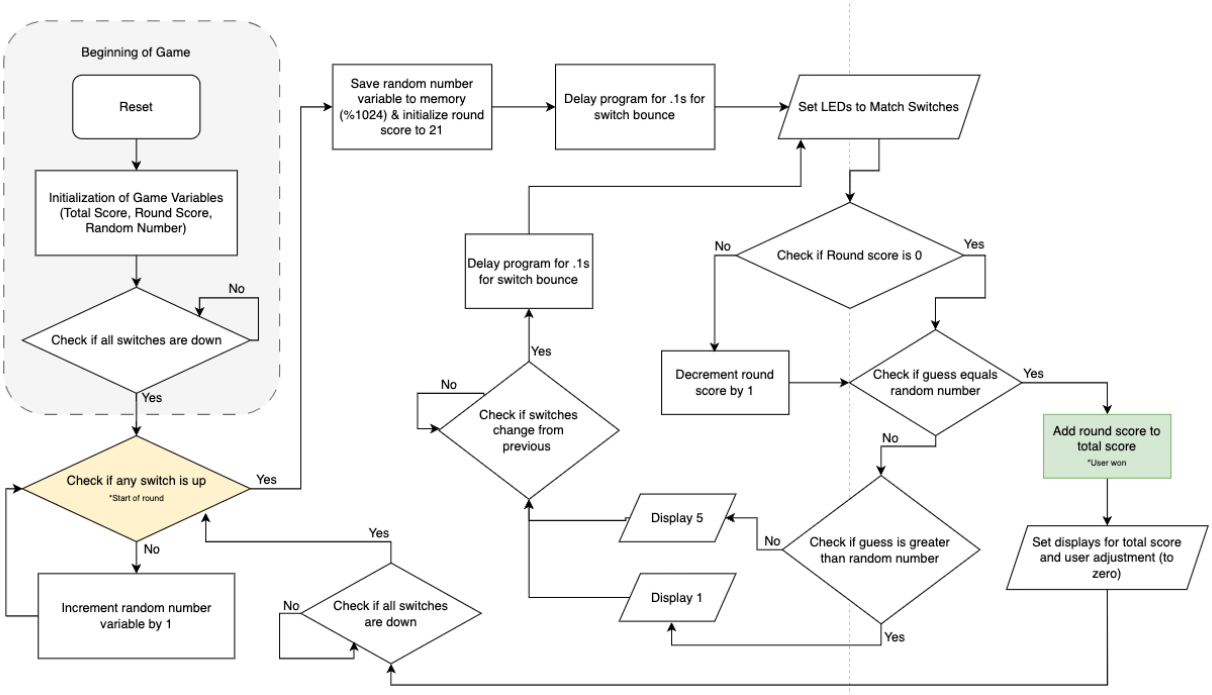

Figure 1. Flow chart of game where user's objective is to guess a random number with 10 switches on the DE-10 board. Users can obtain a score from 20 – 0 where each successive guess reduces their potential score. '1' is displayed if their score is too big and '5' is displayed is their score is too small.

Appendix A
SCOMP Assembly Code To Count The Number Of Set Bits In The Number -3333

```
;Prelab.asm meant to count and store the number of set bits
;in -333
;Author: Rudra Goel
;Date: 10/24/2024

ORG 0
LOADI -1                ;put -1 into AC
SHIFT 11                ;shift AC 11 bits left -> -2024
ADDI -1023
ADDI -262               ;bring AC to -3333

CALL CountSetBits       ; Perform Subroutine
STORE Result1           ; store answer into memory

CALL CountSetBits       ; count bits (set) on that anser to the
previous question
STORE Result2           ; store result

Infinite:
     JUMP Infinite      ;infinite loop

CountSetBits:
     STORE Original_Val ;save intended value
    LOADI 16            ;reinitialize loop counter
    STORE Loop_Counter  ;Store initial value of loop counter
    LOADI 0
    STORE Set_Bits_Count

SubroutineLoop:
     LOAD Loop_Counter  ;load the counter into AC
     JZERO END          ;if we did all 16 iterations, return
     LOAD Original_Val  ;get original val in AC
     AND Mask           ;AC AND 0x0001 ->  if LSB is 1 or zero
     ADD Set_Bits_Count ;add that 0 or 1 to the set_bits_count
     STORE Set_Bits_Count     ;Store updated count

     LOAD Loop_Counter  ;get the loop counter
     ADDI -1                 ;decrement loop counter
     STORE Loop_Counter ;store updated loop counter

     LOAD Original_Val  ;get the val back
     SHIFT -1           ;shift right by one bit to get next LSB
     STORE Original_Val ;restore the new shifted value

     JUMP SubroutineLoop ;loop back to top

END:
```

```
    LOAD Set_Bits_Count ;load # set bits into memory
    RETURN


;Subroutine Variables
Set_Bits_Count:     DW   0
Original_Val:  DW   0
Loop_Counter:  DW   0
Mask:               DW   1

;Result Locations
Result1:       DW   0
Result2:       DW   0
```

Appendix B
SCOMP Assembly Code For Bouncing LED Program Only If Two Or Less Switches Active

```
; IODemo.asm
; Produces a "bouncing" animation on the LEDs.
; The LED pattern is initialized with the switch state.
;Author: Kevin Johnson & Rudra Goel
;Date: 10/24/2024

ORG 0
Start:
     ; Get and store the switch values
     IN     Switches
     OUT    LEDs
     STORE  Pattern
     CALL CountSetBits   ;determine number of set bits

     ADDI -2; if num of set bits is 2, AC will be 0

     ;if more than two switches high, it will jump to start
     JPOS Start

     ;if only two switches active, reread that switch and enter
     IN Switches

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

     ; Check if the left place is 1 and if so, switch direction
     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
     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

;subroutine for counting the number of set bits in the AC
CountSetBits:
     STORE Original_Val  ;save intended value
     LOADI 16            ;reinitialize loop counter
     STORE Loop_Counter  ;Store initial value of loop counter
          LOADI 0
    STORE Set_Bits_Count

SubroutineLoop:
     LOAD Loop_Counter   ;load the counter into AC
     JZERO END           ;if we did all 16 iterations, return

     LOAD Original_Val   ;get original val in AC
     AND Mask            ;AC AND 0x0001 if LSB is 1 or zero
     ADD Set_Bits_Count  ;add that 0 or 1 to the set_bits_count
     STORE Set_Bits_Count     ;Store updated count

     LOAD Loop_Counter   ;get the loop counter
     ADDI -1                  ;decrement loop counter
     STORE Loop_Counter  ;store updated loop counter

     LOAD Original_Val   ;get the val back
     SHIFT -1            ;shift right by one bit to get next LSB
     STORE Original_Val  ;restore the new shifted value

     JUMP SubroutineLoop ;loop back to top

END:
     LOAD Set_Bits_Count ;load number of set bits into memory
     RETURN
```

```
; Variables
Pattern:   DW 0

; 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

;Count Set Bits Subroutine Variables
Set_Bits_Count:     DW   0
Original_Val:       DW   0
Loop_Counter:       DW   0
Mask:               DW   1
```

Appendix C
SCOMP Assembly Code Implementing Random Number Guessing Game On DE-10 Board

```
;Assembly code for implementing guessing game. Uses delay
;subroutine declared below
;Author: Rudra Goel
;Date: 10/24/2024

ORG 0
Start:
     ;reset total score to zero
     LOADI 0
     STORE TotalScore
     OUT Hex0       ;display total score on 7seg display
     OUT Hex1       ;display user guess adjuster display to 0
     STORE TargetNumber
     STORE RandomNumber


CheckIfSwitchesAreDown:  IN Switches
                         JPOS CheckIfSwitchesAreDown

                         CALL Delay     ;Delay for switch bounce

CheckIfSwitchesAreUp:
                         LOAD RandomNumber
                         ADDI 1
                         STORE RandomNumber
                         IN Switches
                         JZERO CheckIfSwitchesAreUp

     LOAD RandomNumber
     AND RandomMask      ; modulo random number 1024
     STORE TargetNumber  ; store the target

     LOADI 21
     STORE RoundScore    ; initialize the round score to 21

     CALL Delay          ;delay for switch bounce

SetLEDsToSwitches:  IN Switches    ;read switches
                    OUT LEDs       ;output the signals to LEDs
                    STORE Guess    ;save switch in Guess

     LOAD RoundScore
     ;if score was zero, then bypass decrementing the score
     JZERO CompareToGuess

     ADDI -1
     STORE RoundScore    ;decrement the score
```

```
CompareToGuess:
               LOAD Guess
               SUB TargetNumber    ;AC <= Guess - TargetNumber
               ; Guess is Random number so user wins round

               JZERO RoundWon
               JPOS GuessTooHigh   ; Guess > TargetNumber
               JNEG GuessTooLow    ; Guess < TargetNumber

RoundWon:
     LOAD RoundScore
     ADD TotalScore
     ;add round score to total score and save in mem
     STORE TotalScore
     OUT Hex0            ; update display of total score

     LOADI 0
     ;reset the display of telling user their guess to 0
     OUT Hex1
;jump to wait till switches are all down to start of next round
     JUMP CheckIfSwitchesAreDown

GuessTooHigh:
     LOADI 1
     OUT Hex1  ;output 1 - indicate the users guess was too high
     JUMP CheckIfSwitchesChange

GuessTooLow:
     LOADI 5
     OUT Hex1  ;output -1 indicate the users guess was too low

CheckIfSwitchesChange:
     IN Switches    ;read in the current switch value
     SUB Guess ; do CurrentSwitch - Orginal Guess
     ; if the switch value is the same, recheck until it is not
     JZERO CheckIfSwitchesChange

     JUMP SetLEDsToSwitches   ;jump back to round start


;Subroutine to delay by .1 seconds based on a 10Hz timer
Delay:
     OUT    Timer
WaitingLoop:
     IN     Timer
     ADDI   -1
```

```
    JNEG   WaitingLoop
    RETURN


;Variables global to game
TotalScore:         DW   0
RandomNumber:  DW   0
TargetNumber:  DW   0
RandomMask:         DW   &B1111111111
RoundScore:         DW   0
Guess:              DW   0


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