SNAKE

Informational Report

Arsalan Mughal

CPE 233

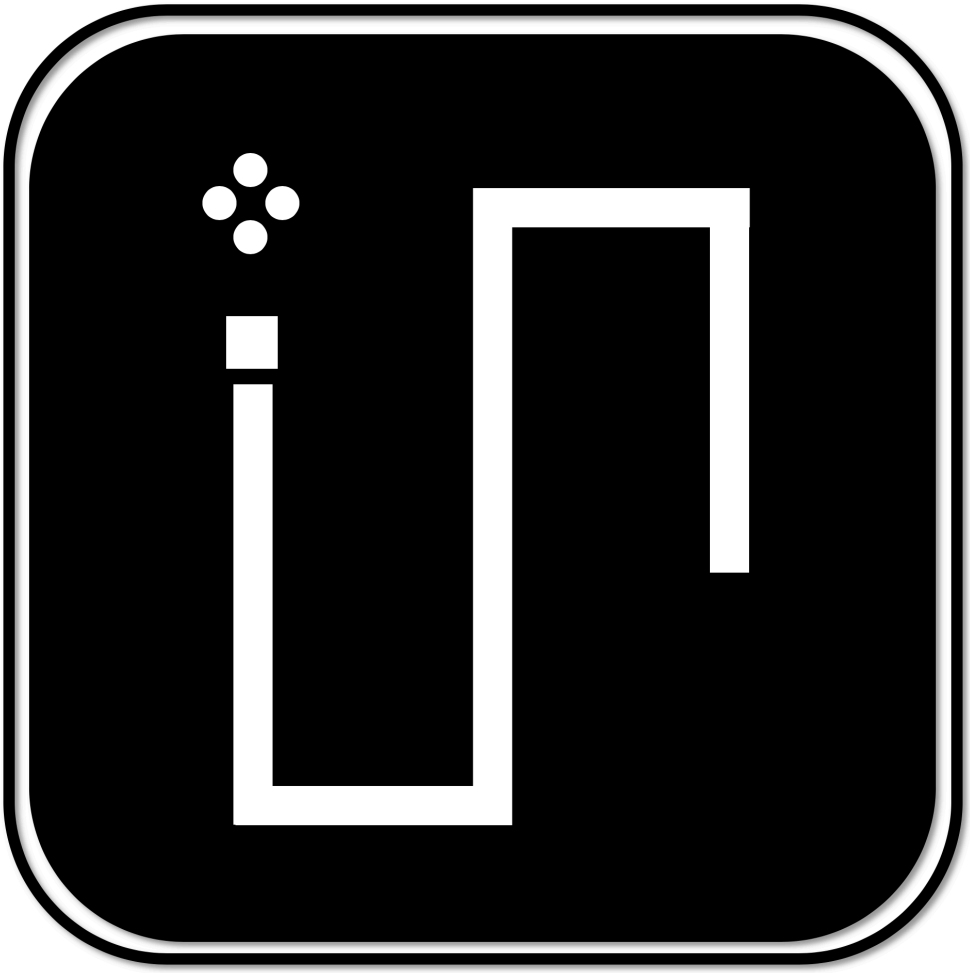


Table of Contents

Introduction

This device is a development of the popular classic game of ‘Snake’ written in RAT Assembly code. The game’s algorithm itself is within the assembly code, and the generated opcode is run on a custom RAT CPU architecture developed and generated in a Xilinix Vivado environment and written in System Verilog.

The goal of the game is to guide the snake on screen to the nearest food element, so it can be eaten, and the snake can grow in length, while also being careful not to hit any walls or the snake itself in the process. When the food is eaten, the snake grows exactly one unit (pixel) in length, before the food is randomly generated and relocated again on screen.

The device runs on a Digilent Basys 3 Board with a 50 MHz Clock. Runtime for execution of a single RAT Assembly code instruction is 40 ns. Peripheral devices include a standard Keyboard and VGA compatible display.

Precision is key for taking in user input quickly, so that delay in the game is minimal. When a key is pressed on the keyboard, an interrupt is generated, quickly storing the user’s input in no more than 3 instructions of assembly code (120 nanoseconds).

Before the next display cycle refresh of the game is generated, the game device already knows of the user’s next input and is able to display new information accordingly. Input ranges between acceptable keypresses from W, A, S, D and unacceptable presses from any and all other keypresses (except for the reset/menu screen, which accepts any key to continue).

Operation Manual

Controls:

W - Move the snake Up

A - Move the snake Left

S - Move the snake Down

D - Move the snake Right

Gameplay:

To play the game, first exit the Menu Screen by pressing any key. Once in the game, the snake will automatically start moving to the right. Now you are in control. To move the snake, use ‘W’ to move upwards, ‘S’ to move down, and ‘A’ and ‘D’ to move left and right respectively. The goal is to lead the snake over the food element, shown in green, and grow the snake as long as possible without hitting the snake itself or a bordering wall. To keep it easy to understand the snake’s orientation, the head of the snake is colored red, and the body colored yellow. The border of the game is just determined by the edge of the display.

Score is displayed on the Basys Board’s Seven Segment LED display. The score starts out as 0 and increments by +1 for every food consumed by the snake.

Software Design

Program Function:

The software design and program function start with clearing all registers and the screen of any printed elements. The program sits in an infinite ‘wait’ loop until a key is pressed, triggering the interrupt routine. This routine takes the key pressed and stores it to be used to identify direction the snake should move.

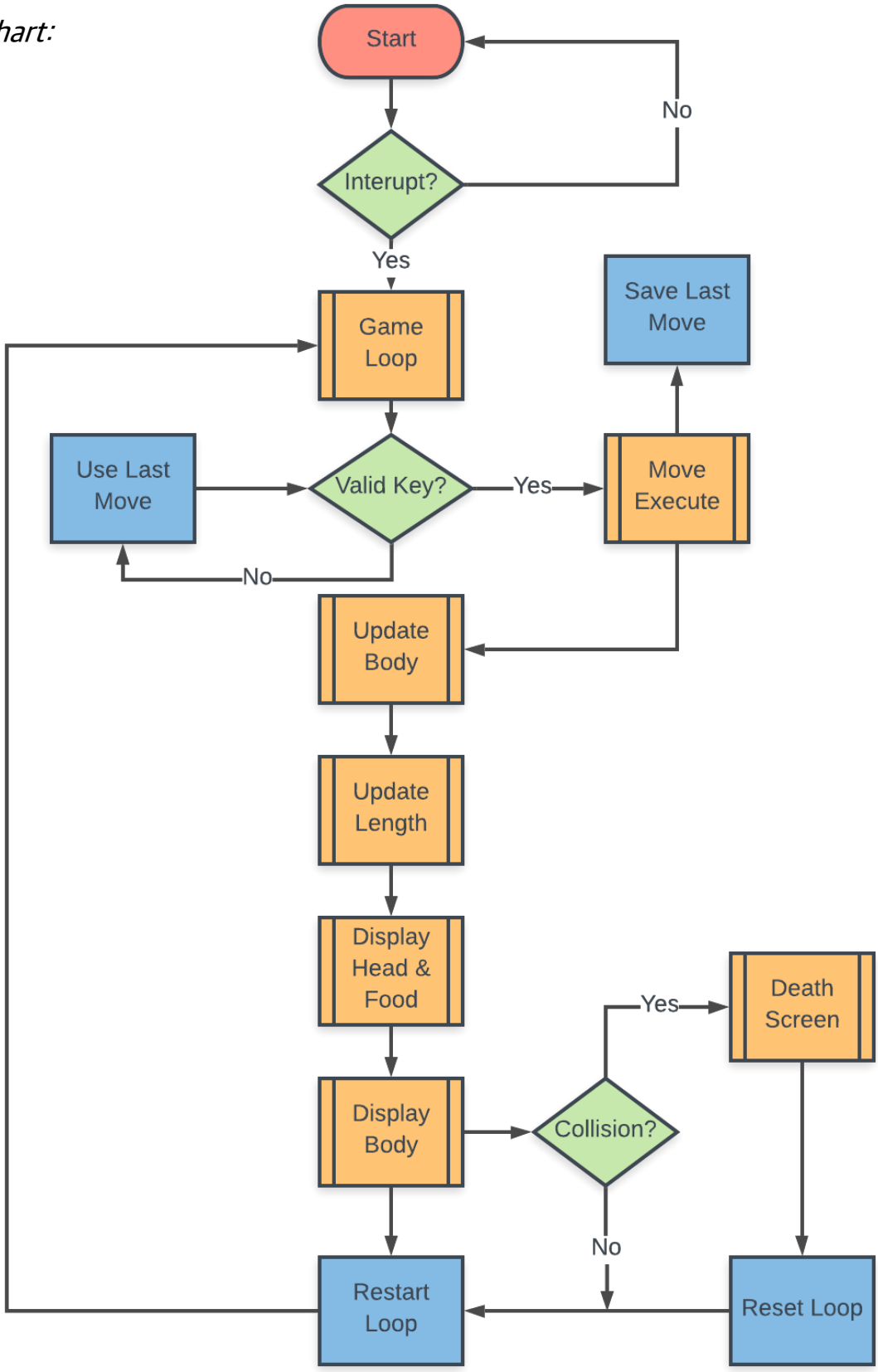
After this is done, the program enters a subroutine to obtain a random (X,Y) location for the food element and enters the game loop. In this loop, the current location of the snake is saved for later use (default is center of the screen). The key pressed by the user is then parsed to check validity, and is compared to preloaded registers, determining what direction the snake should go if the key was valid. If the key was invalid (wrong key or no key pressed) the program defaults the next move to the previous move, keeping the snake moving in its initial direction.

Once a valid key press is received, an update is received to the changing coordinate value for the required move (Up & Down changes Y axis, Left & Right changes X axis value). Then, this valid move is stored to be used as the last valid move in case of an invalid keypress by the user during or at the end of the loop. The body is then updated in a loop according to the length. For future loops once the food is eaten, this body update ensures that the snake doesn’t grow from the center of the screen, but rather actually moves along the screen. The body is displayed in a similar fashion. Growth is checked through a simple compare and adds a pixel to the end of the snake if satisfied.

Collisions are determined through checking against set values for game borders or through comparing the head value to body value locations for wall and self-collisions respectively. Food ‘collision’ is also checked in an equivalent method. Any wall/self-collisions lead to a death screen, then a game restart.

Software Design

Flowchart:



Appendix

Assembly Source Code:

```
;-------------------------------------------------------------------------------
; Final Project: Snake
; Class: CPE 233
; Author: Arsalan Mughal
; Date: 6/6/19
; Description: Game of 'Snake' that displays on a 80x60 display via VGA. Input
;              is obtained from user through PS2 Keyboard. Goal of the game is
;              to guide the snake to food using (WASD keys) without hitting a
;              wall or the snake itself. Score is kept on the Seven Segment LED
;                   display. Game is started with any key on the keyboard.
;
;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
; - Define working registers
;-------------------------------------------------------------------------------
;
; .DEF R_IDX0   r30                ; reg for iteration (for loop index)
; .DEF R_TMP0   r29                ; reg for temp calculations
; .DEF R_TMP1   r28                ; reg for temp calculations
; .DEF R_PTR0   r27                ; reg for pointer operations
; .DEF R_PTR1   r26                ; reg for pointer operations
; .DEF R_KEY    r25                ; reg for key pressed value
; .DEF R_TMPX   r24                ; reg for temp x position
; .DEF R_TMPY   r23                ; reg for temp y position
; .DEF R_GROW   r22                ; reg for snake grow/not-grow
; .DEF R_SNBL   r21                ; reg for snake body length
; .DEF R_SCORE  r20                ; reg for game score
; .DEF R_LOG    r19                ; reg for logging/error reporting
; .DEF R_LEVEL  r18                ; reg for difficulty level
; .DEF R_WAIT   r17                ; reg for wait button
; .DEF R_MENU   r16                 ; reg for menu/wait_loop LEDS

; .DEF R_LINE3  r9                  ; reg for end indep. 'display line' coordinate
; .DEF R_LINE2  r8                  ; reg for start dep. 'display line' coordinate
; .DEF R_LINE1  r7                  ; reg for indep. 'display line' coordinate
; .DEF R_COLOR  r6                  ; reg for color of pixel

; .DEF R_BGCOL  r4                  ; reg for background color
```

```
;-------------------------------------------------------------------------------
; - Define misc constants
;-------------------------------------------------------------------------------
;
.EQU KEY_UP         = 0x1D          ; Keycode w
.EQU KEY_LEFT       = 0x1C          ; Keycode a
.EQU KEY_DOWN       = 0x1B          ; Keycode s
.EQU KEY_RIGHT      = 0x23          ; Keycode d


.EQU LOG_WALL       = 0x01          ; wall collision
.EQU LOG_SELF       = 0x02          ; self collision
.EQU LOG_WIN        = 0x10          ; game won (max len achived)

.EQU GAME_MAX       = 0x50          ; 81 - 1 = 80 (hex 0x50)

.EQU COLOR_HEAD     = 0xE0          ; head color
.EQU COLOR_BODY     = 0xFC          ; body color
.EQU COLOR_FOOD     = 0x1C          ; food color
.EQU COLOR_BRDR     = 0x03          ; border color

.EQU MASK_X_RANGE   = 0x3F          ; limit X boundary to 63 (hex 0x3F)
.EQU MASK_Y_RANGE   = 0x1F          ; limit Y boundary to 31 (hex 0x1F)
.EQU MARGIN_X       = 0x02          ; margin from 0 boundary (within upper)
.EQU MARGIN_Y       = 0x02          ; margin from 0 boundary (within upper)

.EQU HEAD_START_X   = 0x28          ; start in middle (40 = 0x28)
.EQU HEAD_START_Y   = 0x1E          ; start in middle (30 = 0x1E)

.EQU RNDM_REG       = 0x74          ; in port for random number

.EQU PS2_KEY_CODE   = 0x44          ; keyboard in port
.EQU VGA_HADD       = 0x90          ;
.EQU VGA_LADD       = 0x91          ;
.EQU VGA_COLOR      = 0x92          ;
.EQU loop_count     = 0xAA          ;
.EQU SSEG           = 0x81          ;
.EQU LEDS           = 0x40          ;
.EQU BG_COLOR       = 0x03          ; Background:  blue

.EQU INSIDE_FOR_COUNT  = 0x20   ; 0xFF
.EQU MIDDLE_FOR_COUNT  = 0x20   ; 0xFF
.EQU OUTSIDE_FOR_COUNT = 0x05   ; 0x60
                                      ; Comment value time delay is about 503 mS
```

```
;-------------------------------------------------------------------------------
; - Data Segment
;-------------------------------------------------------------------------------
; - Current DSEG Size: 2 + 40 + 2 + 2 + 1 = 48 bytes
.DSEG
    ; reserve (x,y) coordinates for a snake (head + boday) of len
    ; of 21 (hex 0x15). When player reaches this max length, player wins
    ; the game. memory consumed by snake = 21 * 2 = 42 bytes (hex 0x2A)
    ; following could have been done by ".BYTE 0x15", but data
    ; will be uninitialzed (non zero), which may cause problems
    ;
.ORG    0x00                        ;
SNK_HEAD_X: .DB 0x0                 ; head x coordinate
.ORG    0x01                        ;
SNK_HEAD_Y: .DB 0x0                 ; head y coordinate
.ORG    0x02                        ;
SNK_BODY_X: .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0 ; body x coordinate
            .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0 ;
.ORG    0x52                        ;
SNK_BODY_Y: .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0 ; body y coordinate
            .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
             .DB 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0 ;
.ORG    0xA2                        ;
SNK_FOOD_X: .DB 0x00                ; food x coordinate
.ORG    0xA3                        ;
SNK_FOOD_Y: .DB 0x00                ; food y coordinate
    ; Misc variables
    ;
.ORG   0xA4                        ;
GAME_SCORE: .DB 0x0                 ; game start score is zero
.ORG    0xA5                        ;
LAST_MOVE:  .DB 0x0                 ; last key pressed
.ORG    0xA6                        ;
NEW_MOVE:   .DB 0x0                 ; current key pressed
```

```
;-------------------------------------------------------------------------------
; - Code Segment
;-------------------------------------------------------------------------------
;
.CSEG
.ORG    0x14F

;-------------------------------------------------------------------------------
; main program
;-------------------------------------------------------------------------------
;
        ; First initialize att states and variables
        ;
init:   SEI                         ; enable interrupts

        CALL    reset_state
        CALL    wait_start          ; wait for key to start, this is
                                    ; for game start, later on key
                                    ; evnets will be coming from
                                    ; interrupt handler, it will
                                    ; update r25 reg with new value
        CALL    randomize_food      ;
        ; Enter main loop

game_loop:
        MOV     r27, SNK_HEAD_X     ; save snake head postion for later
        LD      r24, (r27)          ; it will used to adjust body when
        MOV     r26, SNK_HEAD_Y     ; snake moves to new location
        LD      r23, (r26)          ;
        CALL    get_key             ; get user input
        BRN     parse_key           ; try to process key (see if valid)

use_last_move:
        MOV     r27, LAST_MOVE      ; use last value of key pressed
        LD      r25, (r27)          ; from the memory variable

parse_key:
        CMP     r25, KEY_LEFT       ; left key pressed
        BREQ    left                ;
        CMP     r25, KEY_DOWN       ; down key pressed
        BREQ    down                ;
        CMP     r25, KEY_RIGHT      ; right key pressed
        BREQ    right               ;
        CMP     r25, KEY_UP         ; up key pressed
        BRNE    use_last_move       ;
```

```
up:
        MOV     r27, SNK_HEAD_Y     ; load address of variable in pointer
        LD      r29, (r27)          ; load contents into temp register
        SUB     r29, 1              ; decrement y coordinate
        ST      r29, (r27)          ; store updated value back in memory
        BRN     move_done

left:
        MOV     r27, SNK_HEAD_X     ; load address of variable in pointer
        LD      r29, (r27)          ; load contents into temp register
        SUB     r29, 1              ; decrement x coordinate
        ST      r29, (r27)          ; store updated value back in memory
        BRN     move_done

right:
        MOV     r27, SNK_HEAD_X     ; load address of variable in pointer
        LD      r29, (r27)          ; load contents into temp register
        ADD     r29, 1              ; increment x coordinate
        ST      r29, (r27)          ; store updated value back in memory
        BRN     move_done

down:
        MOV     r27, SNK_HEAD_Y     ; load address of variable in pointer
        LD      r29, (r27)          ; load contents into temp register
        ADD     r29, 1              ; increment y coordinate
        ST      r29, (r27)          ; store updated value back in memory
        BRN     move_done

move_done:
        MOV     r27, LAST_MOVE      ; save current/last key into variable
        ST      r25, (r27)          ; load it in reg from memory

        MOV     r27, SNK_BODY_X     ; load address of snake body x
        MOV     r26, SNK_BODY_Y     ; load address of snake body y
        MOV     r30, r21            ; load snake body length into index

update_body:
        CMP     r30, 0              ; If body length 0 then done
        BREQ    update_length       ;
        LD      r29, (r27)          ; load contents of body x into temp
        LD      r28, (r26)          ; load contents of body y into temp
        ST      r24, (r27)          ; store tmp x cell x into new x
        ST      r23, (r26)          ; store tmp x cell x into new y
        MOV     r24, r29            ; swap tmp x with older x
        MOV     r23, r28            ; swap tmp y with older y
```

```
        SUB     r30, 1              ; decrement index
        ADD     r27, 1              ; increment x pointer
        ADD     r26, 1              ; increment y pointer
        BRN     update_body         ; keep looping until done

update_length:
        CMP     r22, 1              ; snake should grow
        BRNE    display_stuff       ; if not jump to display_stuff
        ST      r24, (r27)          ; store tmp x cell x into new x
        ST      r23, (r26)          ; store tmp x cell x into new y
        ADD     r21, 1              ; increase snake body length
        MOV     r22, 0              ; disable grow_snake_flag
        MOV     r19, LOG_WIN        ; pre-load win value, just incase...
        CMP     r21, GAME_MAX       ; if max reached game over
        BREQ    game_over_win       ; exit/restart game

display_stuff:
        CALL clear_screen
        MOV     r27, SNK_HEAD_X     ; load address of snake head x
        LD      r8, (r27)           ; head x
        MOV     r26, SNK_HEAD_Y     ; load address of snake head y
        LD      r7, (r26)           ; head y
        MOV     r6, COLOR_HEAD      ; head color
        call    draw_dot            ; draw dot

        MOV     r27, SNK_FOOD_X     ; load address of snake food x
        LD      r8, (r27)           ; food x
        MOV     r26, SNK_FOOD_Y     ; load address of snake food y
        LD      r7, (r26)           ; food y
        MOV     r6, COLOR_FOOD      ; food color
        call    draw_dot            ; draw dot

        MOV     r30, r21            ; load snake body length into index
        MOV     r27, SNK_BODY_X     ; load address of snake body x
        MOV     r26, SNK_BODY_Y     ; load address of snake body y
        MOV     r6, COLOR_BODY      ; body color

display_body:
        CMP     r30, 0              ; If body length 0 then done
        BREQ    wall_collision      ;
        LD      r8, (r27)           ; body x
        LD      r7, (r26)           ; body y
        call    draw_dot            ; draw dot
        SUB     r30, 1              ; decrement index
        ADD     r27, 1              ; increment x pointer
```

```
        ADD     r26, 1              ; increment y pointer
        BRN     display_body        ; keep looping until done

wall_collision:
        MOV     r19, LOG_WALL       ; pre-load error, just incase...
        MOV     r27, SNK_HEAD_X     ; load address of snake head x
        LD      r24, (r27)          ; head x
        CMP     r24, 80             ; oops! too far left
        BREQ    game_over_wall      ; exit/restart game
        CMP     r24, 0              ; oops! too far right
        BREQ    game_over_wall      ; exit/restart game
        MOV     r26, SNK_HEAD_Y     ; load address of snake head y
        LD      r23, (r26)          ; head y
        CMP     r23, 60             ; oops! too far down
        BREQ    game_over_wall      ; exit/restart game
        CMP     r23, 0              ; oops! too far up
        BREQ    game_over_wall      ; exit/restart game

        MOV     r19, LOG_SELF       ; pre-load error, just incase...
        MOV     r30, r21            ; load snake body length into index
        MOV     r27, SNK_BODY_X     ; load address of snake body x
        MOV     r26, SNK_BODY_Y     ; load address of snake body y
self_collision:
        CMP     r30, 0              ; If body length 0 then done
        BREQ    food_collision      ;
        BREQ    restart_loop        ;
        LD      r29, (r27)          ; body x
        LD      r28, (r26)          ; body y
        CMP     r29, r24            ; body x match head x, check y now
        BRNE    collision_loop      ; keep looping until done
        CMP     r28, r23            ; y matched? game over!
        BREQ    game_over_self      ; exit/restart game
collision_loop:
        SUB     r30, 1              ; decrement index
        ADD     r27, 1              ; increment x pointer
        ADD     r26, 1              ; increment y pointer
        BRN     self_collision      ;

food_collision:
        MOV     r27, SNK_FOOD_X     ; load address of snake food x
        LD      r29, (r27)          ; food x
        MOV     r26, SNK_FOOD_Y     ; load address of snake food y
        LD      r28, (r26)          ; food y
        CMP     r29, r24            ; food x match head x, check y now
        BRNE    restart_loop        ;
```

```
        CMP     r28, r23            ; y matched
        BRNE    restart_loop        ;
        ADD     r20, 1              ; we found food! increment score
        call    display_score       ; display new updated  score
        call    randomize_food      ; add new food to new location
        ADD     r22, 1              ; set grow flag to 1

restart_loop:
        call    sleep_delay
        BRN     game_loop

;-------------------------------------------------------------------------------
; Interrupts service routine
;-------------------------------------------------------------------------------
;
randomize_food:
        MOV     r27, SNK_FOOD_X     ; load address of snake food x
        MOV     r26, SNK_FOOD_Y     ; load address of snake food y
        IN      r29, RNDM_REG       ; read random number
        AND     r29, MASK_X_RANGE   ; limit random x value within range
        ADD     r29, MARGIN_X       ; just incase random # is 0
        IN      r28, RNDM_REG       ; read random number
        AND     r28, MASK_Y_RANGE   ; limit random y value within range
        ADD     r28, MARGIN_Y       ; just incase random # is 0
        ST      r29, (r27)          ; save new food x position
        ST      r28, (r26)          ; save new food y position
        RET

;-------------------------------------------------------------------------------
; reset all state variables
;-------------------------------------------------------------------------------
;
reset_state:
             MOV     r17, 0              ; set wait button state to zero
        MOV     r18, 1              ; set difficulty level to zero
        MOV     r19, 0              ; get error reporting to zero
        MOV     r20, 0              ; set game score to zero
        MOV     r21, 0              ; set snake body length to zero
        MOV     r22, 0              ; set snake grow flag to zero
             OUT          r20, SSEG                  ; output 0 score to board
        MOV     r23, HEAD_START_Y   ; head start position y
        MOV     r24, HEAD_START_X   ; head start position x
        MOV     r25, 0              ; set key input to zero
        MOV     r27, SNK_HEAD_X     ; load address of variable in pointer
        MOV     r29, HEAD_START_X   ; load contents into temp register
```

```
        ST      r29, (r27)          ; store updated value back in memory
        MOV     r27, SNK_HEAD_Y     ; load address of variable in pointer
        MOV     r29, HEAD_START_Y   ; load contents into temp register
        ST      r29, (r27)          ; store updated value back in memory
        MOV     r27, LAST_MOVE      ; load address of variable in pointer
        MOV     r29, KEY_RIGHT      ; load contents into temp register
        ST      r29, (r27)          ; store updated value back in memory
        MOV         r4, 0x1C                ; set background color to green
             CALL    clear_screen        ; clear screen
        RET

;-------------------------------------------------------------------------------
; clear screen
;-------------------------------------------------------------------------------
clear_screen:
             MOV          r4, 0x03
             CALL   draw_background
        RET
;-------------------------------------------------------------------------------
; display score on LED
;-------------------------------------------------------------------------------
display_score:
             OUT          r20, SSEG
        RET
;-------------------------------------------------------------------------------
; get key from keyboard
;-------------------------------------------------------------------------------
get_key:
        MOV     r27, NEW_MOVE       ; use new value of key pressed
        LD      r25, (r27)          ; load it into r25 register
        RET
;-------------------------------------------------------------------------------
; wait for game start
;-------------------------------------------------------------------------------
wait_start:
wait_loop:
        CMP     r17, 1              ; did user press button
             MOV          r16, 0x01                ; light LED to signify wait_loop
             OUT     r16, LEDS               ;
        BRNE    wait_loop           ; wait until user ready
             MOV          r16, 0x00                ; turn off LED
             OUT          r16, LEDS
        MOV     r17, 0              ; reset button state
        RET
```

```
;-------------------------------------------------------------------------------
; game over (you win !!!)
;-------------------------------------------------------------------------------

; reg r19 has value of error code, to display on LED

game_over_win:
        OUT         r19, SSEG
             CALL   clear_screen
             CALL   red_screen
             MOV          r18, 0x08
             CALL   Delay
             BRN     init             ; restart game
        RET

;-------------------------------------------------------------------------------
; game over (you hit wall)
;-------------------------------------------------------------------------------

; reg r19 has value of error code, to display on LED

game_over_wall:
        OUT         r19, SSEG
             CALL   clear_screen
             CALL   red_screen
             MOV          r18, 0x08
             CALL   Delay
             BRN     init             ; restart game
        RET

;-------------------------------------------------------------------------------
; game over (you hit self)
;-------------------------------------------------------------------------------

; reg r19 has value of error code, to display on LED

game_over_self:
        OUT         r19, SSEG
             CALL   clear_screen
             CALL   red_screen
             MOV          r18, 0x08
             CALL   Delay
             BRN     init             ; restart game
        RET
```

```
;-------------------------------------------------------------------------------
; sleep for some time
;-------------------------------------------------------------------------------
; reg r18 is used for tuning difficulty level

sleep_delay:

             SUB          R18, 0x01
             MOV          R1, OUTSIDE_FOR_COUNT     ;set outside for loop count
outside_one:
             SUB    R1, 0x01
        MOV  R2, MIDDLE_FOR_COUNT      ;set middle for loop count
middle_one:
             SUB    R2, 0x01
        MOV  R3, INSIDE_FOR_COUNT      ;set inside for loop count
inside_one:
             SUB    R3, 0x01
        BRNE inside_one
             OR           R2, 0x00           ;load flags for middle for counter
             BRNE   middle_one
             OR           R1, 0x00           ;load flags for outsde for counter
value
             BRNE   outside_one

             CMP          R18, 0x00
             BRNE   sleep_delay
        RET


;-------------------------------------------------------------------------------
; Delay
;-------------------------------------------------------------------------------

Delay:
             SUB          R18, 0x01
             MOV          R1, 0xFF     ;set outside for loop count
outside_loop:
             SUB    R1, 0x01
        MOV  R2, 0xFF     ;set middle for loop count
middle_loop:
             SUB    R2, 0x01
        MOV  R3, 0x60     ;set inside for loop count
inside_loop:
             SUB    R3, 0x01
        BRNE inside_loop
```

```
            OR          R2, 0x00          ;load flags for middle for counter
            BRNE   middle_loop
            OR          R1, 0x00          ;load flags for outsde for counter
value
            BRNE   outside_loop

            CMP         R18, 0x00
            BRNE    Delay
        RET

;-----------------------------------------------------------------------------
; Red Screen :(
;-----------------------------------------------------------------------------

red_screen:
            MOV    r4, 0xE0
            CALL   draw_background




;-----------------------------------------------------------------------------
;-  Subroutine: draw_horizontal_line
;-
;-  Draws a horizontal line from (r8,r7) to (r9,r7) using color in r6
;-
;-  Parameters:
;-   r8  = starting x-coordinate
;-   r7  = y-coordinate
;-   r9  = ending x-coordinate
;-   r6  = color used for line
;-
;- Tweaked registers: r8,r9
;-----------------------------------------------------------------------------
draw_horizontal_line:
        ADD    r9,0x01             ; go from r8 to r15 inclusive

draw_horiz1:
        CALL   draw_dot            ;
        ADD    r8,0x01             ;
        CMP    r8,r9               ;
        BRNE   draw_horiz1         ;
        RET
```

```
;-------------------------------------------------------------------------------
;-  Subroutine: draw_vertical_line
;-
;-  Draws a horizontal line from (r8,r7) to (r8,r9) using color in r6
;-
;-  Parameters:
;-   r8  = x-coordinate
;-   r7  = starting y-coordinate
;-   r9  = ending y-coordinate
;-   r6  = color used for line
;-
;- Tweaked registers: r7,r9
;-------------------------------------------------------------------------------
draw_vertical_line:
        ADD    r9,0x01             ;

draw_vert1:
        CALL   draw_dot            ;
        ADD    r7,0x01             ;
        CMP    r7,R9               ;
        BRNE   draw_vert1          ;
        RET

;-------------------------------------------------------------------------------
;-  Subroutine: draw_background
;-
;-  Fills the 80x60 grid with one color using successive calls to
;-  draw_horizontal_line subroutine.
;-
;-  Tweaked registers: r10,r7,r8,r9
;-------------------------------------------------------------------------------
draw_background:
        MOV    r6,r4                    ; get color
        MOV    r10,0x00             ; r10 keeps track of rows
start:  MOV    r7,r10               ; load current row count
        MOV    r8,0x00              ; restart x coordinates
        MOV    r9,0x4F              ; set to total number of columns

        CALL   draw_horizontal_line
        ADD    r10,0x01             ; increment row count
        CMP    r10,0x3C             ; see if more rows to draw
        BRNE   start                ; branch to draw more rows
        RET
```

```
;-------------------------------------------------------------------------------
;- Subrountine: draw_dot
;-
;- This subroutine draws a dot on the display the given coordinates:
;-
;- (X,Y) = (r8,r7)  with a color stored in r6

;-------------------------------------------------------------------------------
draw_dot:
           OUT   r8,VGA_LADD        ; write bot 8 address bits to register
           OUT   r7,VGA_HADD        ; write top 5 address bits to register
           OUT   r6,VGA_COLOR       ; write color data to frame buffer
           RET

;-------------------------------------------------------------------------------
; interrupts service routine
;-------------------------------------------------------------------------------
;
my_isr:
        MOV     r17, 1              ; set flag for button pressed
        IN      r29, PS2_KEY_CODE   ; read key from keyboard
        MOV     r27, NEW_MOVE       ; use new value of key pressed
        ST      r29, (r27)          ; save in memory
        RETIE
;

;-------------------------------------------------------------------------------
;
; interrupt vector
;-------------------------------------------------------------------------------
;
.CSEG
.ORG 0x3FF
BRN my_isr

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
```