BASYSpace Invaders

James Whitney Chandler Gifford
CPE 233 Spring 2016

Introduction

Our project is a space invaders type game displayed on the VGA monitor. The game consists of a player controllable ship that can move left or right at the bottom of the screen via the A and D keys and that can shoot projectiles toward the top of the screen by pressing the spacebar. There are be three purple “asteroid” obstacles in front of the player that projectiles cannot pass through. On the top half of the screen there three stationary enemy “invaders”. If a projectile fired by the player hits an enemy the enemy will be removed from the game.

Hardware Components

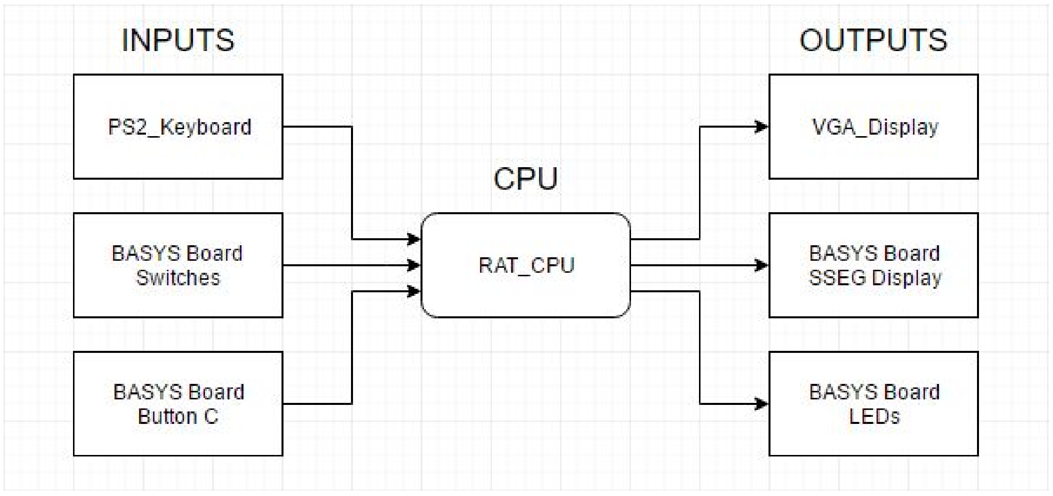

Figure 1. Hardware Component diagram.

In the project, the inputs into the system includes a PS2 Keyboard, as well as the switches and center button on the BASYS Board. For outputs, the system has the VGA Display, the SSEG Display, and the board LEDs.

- The keyboard is used to take input from the player using the **A**,**D**, and **Space** keys, when any key is pressed, the address of that key is displayed in decimal values on the SSEG display.,
- The switches input is output immediately to the corresponding LEDs on the board during the idle loop of the program. The switches purpose is to give debugging feedback as to whether or not the program is able to reach the looping phase of assembly code.
- The center button on the board is used to reset the program.

Software Components

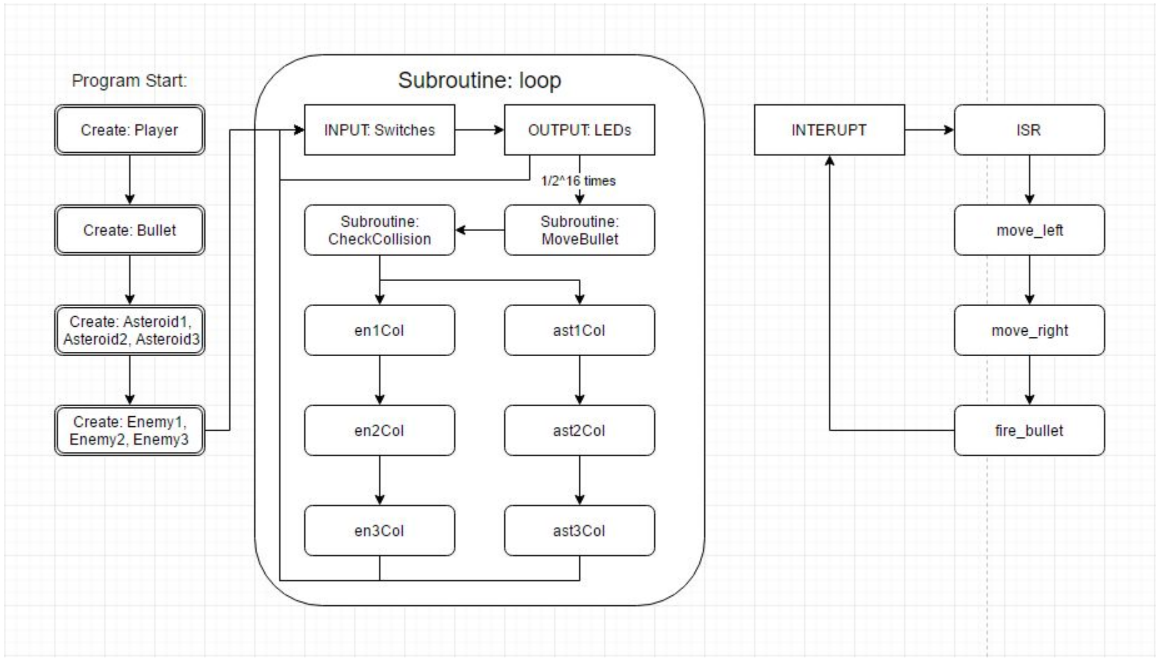

Figure 2. Software Flow Diagram

Our assembly code begins by drawing each of our objects (player/ship, bullet[not enabled], three asteroids, three enemies) then entering an “idle phase” in which the processor takes inputs from the switches and outputs them to the LEDs as shown in figure 1. In order to slow the bullet down to playable speeds, the next two subroutines MoveBullet and CheckCollision are only called once every 2^{16} loops. MoveBullet simply moves the bullet up one block on the screen while checking to see if it has reached the top of the screen, if it has, then it disables the bullet.

CheckCollision is the subroutine that consists of the most logic of any of the subroutines we developed. The subroutine checks for the collision against an object by running either asteroid subroutines ast1Col through 3 or for an enemy by running subroutines en1Col through 3 depending on the Y position of the bullet. After checking collision data it removes an enemy if appropriate then branches back to to the top of loop.

The interrupt subroutine, when triggered runs through and checks whether the **A**, **D**, and **Space** keys are pressed. If **A** is pressed it moves the player left, if **D** is pressed the player is moved to the right, if **Space** is pressed the bullet is moved to a position in front of the player and enabled.

User Operation

Our game is played by moving a spaceship at the bottom of the screen using the **A** and **D** keys and firing bullets using the **Spacebar**. The goal of the game is to hit all of the green aliens at the top of the screen. The purple asteroids act as obstacles and will stop any bullets that hit them.

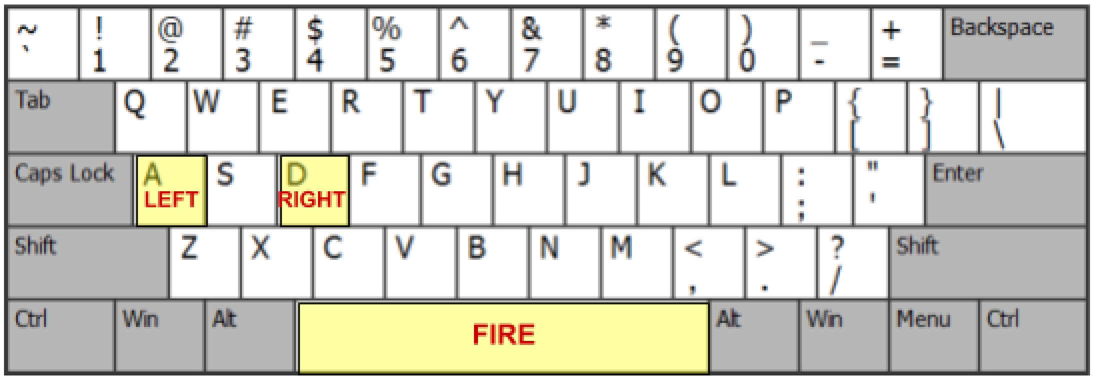

Figure 3: Keyboard Controls

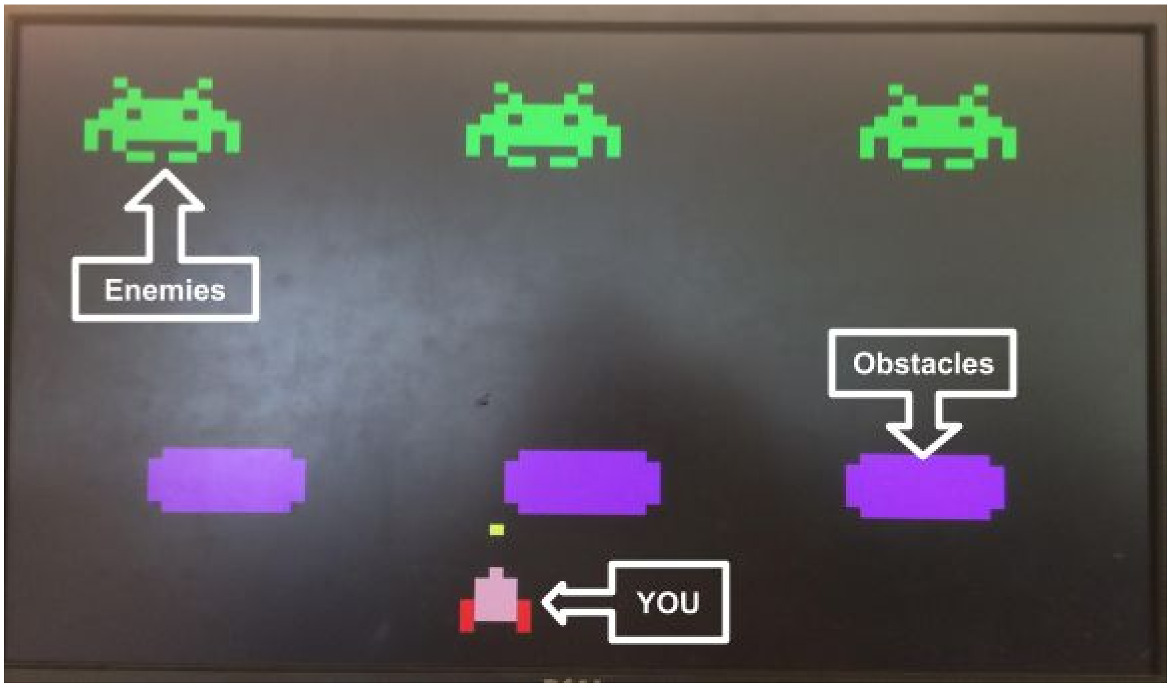

Figure 4: Gameplay screenshot

Conclusion/Future Work

This project turned out to be much more difficult than we anticipated. We spent more than half our time troubleshooting and just trying to get something to display on the screen. This left less time for us to actually implement useful/interesting features in our game. Given more time we would add things like more enemies, moving enemies, enemies that shoot back, a score counter and the ability to fire more than one bullet at a time. I believe all of these goals are fairly easily achievable and had we not had so many problems with our project breaking we likely would have gotten to implement at least some of these parts of our game. In the end, we are just glad we were able to get anything at all to display on the screen and we were actually able to accomplish a lot in the time we had the project in a working state.

Appendix

```
;-------------------------------------
;   BASYSpace Invaders: An attempt to make space invaders with the RAT CPU
;      Authors: James Whitney, Chandler Gifford
;      Date:    6/2/2016
;
;   Based on code from:
;
;   Keyboard and Bufferless VGA Demo Assembly Program
;   Authors: Bridget Benson and Ryan Rumsey
;   Date: 5/4/16
;-------------------------------------

;------------------
;- Port Definitions
;------------------
.EQU  X_POS_EN_ID = 0xA1   ;VGA Controller port X_POS_EN
.EQU  Y_POS_ID = 0xA2   ;VGA Controller port Y_POS
.EQU  RGB_DATA_ID = 0xA3  ;VGA Controller port RGB_DATA_IN
.EQU  OBJ_ADDR_ID = 0xA4   ;VGA Controller port OBJ_ADDR
.EQU  SSEG_ID     = 0x80  ;Seven Segment Display
.EQU  LEDS_ID     = 0x40
.EQU  SWITCHES_ID = 0x20
.EQU  BUTTONS_ID  = 0x24
.EQU  PS2_KEY_CODE_ID = 0x30
.EQU  PS2_CONTROL_ID = 0x32

;------------------
;- Bit Masks
;------------------
.EQU  EN_MASK     = 0x80  ;Enable bit is in MSB position of X_POS_EN
.EQU  DIS_MASK    = 0x7F  ;Disable X_POS_EN MSB

;------------------------------------------------------------------
; Various Keyboard Definitions
;------------------------------------------------------------------
.EQU KEY_UP     = 0xF0        ; key release data
.EQU int_flag   = 0x01        ; interrupt hello from keyboard
.EQU UP       = 0x1D         ; 'w'
.EQU LEFT     = 0x1C         ; 'a'
.EQU RIGHT    = 0x23         ; 'd'
```

```
.EQU DOWN     = 0x1B         ; 's'
.EQU SPACE    = 0x29         ; spacebar
;-------------------------------------------------------------

;------------------
;- VGA Boundaries
;------------------
.EQU  MAX_X     = 0xCB  ;Maximum X position
.EQU  MAX_Y     = 0x3B   ;Maximum Y position
;------------------
;- Object Memory
;------------------
.EQU  OBJ0_MEM = 0x00   ;Stack address for Ship info
.EQU  OBJ1_MEM = 0x03   ;Stack address for Bullet info
.EQU  OBJ2_MEM = 0x06   ;Stack address for Asteroid1 info
.EQU  OBJ3_MEM = 0x09   ;Stack address for Asteroid2 info
.EQU  OBJ4_MEM = 0x0C   ;Stack address for Asteroid3 info
.EQU  OBJ5_MEM = 0x0F   ;Stack address for Enemy 1 info
.EQU  OBJ6_MEM = 0x12   ;Stack address for Enemy 2 info
.EQU  OBJ7_MEM = 0x15   ;Stack address for Enemy 3 info


;----------------------
;- Register Definitions
;----------------------
.DEF  R_X_POS_EN   = r0
.DEF  R_Y_POS      = r1
.DEF  R_RGB_DATA   = r2
.DEF  R_OBJ_ADDR   = r3
.DEF  R_ARGUMENT   = r31


.CSEG
.ORG 0x01

; Draw on screen
init:    ;Enable Ship
         MOV   R_X_POS_EN, 0x28
         OR    R_X_POS_EN, EN_MASK
         MOV   R_Y_POS,    0x33
         MOV   R_OBJ_ADDR, 0x01
         CALL  update_obj
         MOV   R_ARGUMENT, OBJ0_MEM ;Set up r31 with mem address
         CALL  set_obj_data   ;Store r0-2 into stack at OBJ0_MEM

         ;Enable Bullet
         MOV   R_X_POS_EN, 0x00
         MOV   R_Y_POS,    0x00
         MOV   R_OBJ_ADDR, 0x02
         CALL  update_obj
         MOV   R_ARGUMENT, OBJ1_MEM ;Set up r31 with mem address
         CALL  set_obj_data   ;Store r0-2 into stack at OBJ1_MEM

         ;Enable Asteroid 1
         MOV   R_X_POS_EN, 0x0A
         OR    R_X_POS_EN, EN_MASK
         MOV   R_Y_POS,    0x28
         MOV   R_OBJ_ADDR, 0x03
         CALL  update_obj
         MOV   R_ARGUMENT, OBJ2_MEM ;Set up r31 with mem address
```

```
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ2_MEM

        ;Enable Asteroid 2
        MOV   R_X_POS_EN, 0x23
        OR    R_X_POS_EN, EN_MASK
        MOV   R_Y_POS,    0x28
        MOV   R_OBJ_ADDR, 0x04
        CALL  update_obj
        MOV   R_ARGUMENT, OBJ3_MEM ;Set up r31 with mem address
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ3_MEM

        ;Enable Asteroid 3
        MOV   R_X_POS_EN, 0x3B
        OR    R_X_POS_EN, EN_MASK
        MOV   R_Y_POS,    0x28
        MOV   R_OBJ_ADDR, 0x05
        CALL  update_obj
        MOV   R_ARGUMENT, OBJ4_MEM ;Set up r31 with mem address
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ4_MEM

        ;Enable enemy 1
        MOV   R_X_POS_EN, 0x05
        OR    R_X_POS_EN, EN_MASK
        MOV   R_Y_POS,    0x05
        MOV   R_OBJ_ADDR, 0x06
        CALL  update_obj
        MOV   R_ARGUMENT, OBJ5_MEM ;Set up r31 with mem address
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ5_MEM

        ;Enable enemy 2
        MOV   R_X_POS_EN, 0x20
        OR    R_X_POS_EN, EN_MASK
        MOV   R_Y_POS,    0x05
        MOV   R_OBJ_ADDR, 0x07
        CALL  update_obj
        MOV   R_ARGUMENT, OBJ6_MEM ;Set up r31 with mem address
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ6_MEM

        ;Enable enemy 3
        MOV   R_X_POS_EN, 0x3C
        OR    R_X_POS_EN, EN_MASK
        MOV   R_Y_POS,    0x05
        MOV   R_OBJ_ADDR, 0x08
        CALL  update_obj
        MOV   R_ARGUMENT, OBJ7_MEM ;Set up r31 with mem address
        CALL  set_obj_data   ;Store r0-2 into stack at OBJ7_MEM

main:   MOV      R_ARGUMENT, OBJ0_MEM  ;select to move the ship
        MOV      R_OBJ_ADDR, 0x01
        CALL    get_obj_data
        SEI

;---------------------------
; Main loop, repeats continuously with a delay for moving bullet
;----------------------------
loop:   IN    r20, SWITCHES_ID  ;just to test switches
        OUT   r20, LEDS_ID      ;just to test LEDS
        ;delay
        ADD   r25, 0x01
        CMP   r25, 0x00
```

```
         BRNE  skip
         ADD   r26, 0x01
         CMP   r26, 0x00
         BRNE  skip
         CALL  moveBullet ;call move bullet routine
         CALL  checkCollide ;all collision check routine
   skip: BRN   loop           ;hang out here waiting for keyboard interrupts

;-------------------------
; Fire subroutine spawns bullet at tip of ship
;-------------------------
fire:    ADD   R_X_POS_EN, 0x02 ;set position to tip of ship
         SUB   R_Y_POS, 0x01
         MOV   R_OBJ_ADDR, 0x02 ;change to control bullet
         CALL  update_obj
         MOV   R_ARGUMENT, OBJ1_MEM
         CALL  set_obj_data   ;save bullet pos
         MOV   R_OBJ_ADDR, 0x01 ;set back to ship
         MOV   R_ARGUMENT, OBJ0_MEM
         CALL  get_obj_data
         RET

;---------------------------
; Moves the bullet up and disables it when it reaches the top
;--------------------------
moveBullet: MOV   R_OBJ_ADDR, 0x02 ;get bullet
         MOV   R_ARGUMENT, OBJ1_MEM
         CALL  get_obj_data
         CMP   R_Y_POS,0x00    ; see if you can move
         BREQ  hideBullet
         SUB   R_Y_POS, 0x01 ;move up
         CALL  update_obj
         CALL  set_obj_data   ;Store bullet data
BulDone: MOV   R_OBJ_ADDR, 0x01 ;set back to ship
         MOV   R_ARGUMENT, OBJ0_MEM
         CALL  get_obj_data
         RET

hideBullet: AND   R_X_POS_EN, DIS_MASK
         CALL  update_obj
         CALL  set_obj_data
         BRN   BulDone


;--------------------------------
; Checks if the bullet collides with enemy or asteroid
; Removes bullet if there is a hit and removes
; enemy if an enemy is hit
;---------------------------------
checkCollide:  MOV      R_OBJ_ADDR, 0x02 ;get bullet
            MOV   R_ARGUMENT, OBJ1_MEM
            CALL  get_obj_data

            CMP   R_Y_POS, 0x0D ;y pos lines up with enemies
            BREQ  en1Col
            CMP   R_Y_POS, 0x2D ;y pos lines up with astroids
            BREQ  ast1Col
            BRN   noCollide ;otherwise there is no collisiokn
```

```
en1Col:  LD    R11, 0x0F ;load x pos of enem 1
         CMP   R_X_POS_EN, R11 ;left of left astroid
         BRCS  en2Col
         ADD   R11, 0x0B
         CMP   R_X_POS_EN, R11
         BRCC  en2Col
         MOV   R_X_POS_EN, 0x00 ;move bullet
         CALL  set_obj_data
         MOV   R_OBJ_ADDR, 0x06
         MOV   R_ARGUMENT, OBJ5_MEM
         CALL  get_obj_data
         AND   R_X_POS_EN, DIS_MASK
         CALL  update_obj
         CALL  set_obj_data
         CALL  noCollide

en2Col:  LD    R11, 0x12 ;load x pos of enemy 2
         CMP   R_X_POS_EN, R11 ;left of left astroid
         BRCS  en3Col
         ADD   R11, 0x0B
         CMP   R_X_POS_EN, R11
         BRCC  en3Col
         MOV   R_X_POS_EN, 0x00 ;move bullet
         CALL  set_obj_data
         MOV   R_OBJ_ADDR, 0x07
         MOV   R_ARGUMENT, OBJ6_MEM
         CALL  get_obj_data
         AND   R_X_POS_EN, DIS_MASK
         CALL  update_obj
         CALL  set_obj_data
         CALL  noCollide

en3Col:  LD    R11, 0x15 ;load x pos of enemy 3
         CMP   R_X_POS_EN, R11 ;left of left astroid
         BRCS  ast1Col
         ADD   R11, 0x0B
         CMP   R_X_POS_EN, R11
         BRCC  ast1Col
         MOV   R_X_POS_EN, 0x00 ;move bullet
         CALL  set_obj_data
         MOV   R_OBJ_ADDR, 0x08
         MOV   R_ARGUMENT, OBJ7_MEM
         CALL  get_obj_data
         AND   R_X_POS_EN, DIS_MASK
         CALL  update_obj
         CALL  set_obj_data


ast1Col: LD    R11, 0x06 ;load x pos of enem 1
         CMP   R_X_POS_EN, R11 ;left of left astroid
         BRCS  ast2Col
         ADD   R11, 0x0B
         CMP   R_X_POS_EN, R11
         BRCC  ast2Col
         MOV   R_X_POS_EN, 0x00 ;move bullet
         CALL  set_obj_data
         CALL  noCollide

ast2Col: LD    R11, 0x09 ;load x pos of enemy 2
         CMP   R_X_POS_EN, R11 ;left of left astroid
```

```
            BRCS  ast3Col
            ADD   R11, 0x0B
            CMP   R_X_POS_EN, R11
            BRCC  ast3Col
            MOV   R_X_POS_EN, 0x00 ;move bullet
            CALL  set_obj_data
            CALL  noCollide

   ast3Col: LD    R11, 0x0C ;load x pos of enemy 3
            CMP   R_X_POS_EN, R11 ;left of left astroid
            BRCS  noCollide
            ADD   R11, 0x0B
            CMP   R_X_POS_EN, R11
            BRCC  noCollide
            MOV   R_X_POS_EN, 0x00 ;move bullet
            CALL  set_obj_data


   noCollide:  MOV   R_OBJ_ADDR, 0x01 ;set back to ship
            MOV   R_ARGUMENT, OBJ0_MEM
            CALL  get_obj_data
            RET


;------------------------------------------------------------
;- These subroutines add and/or subtract '1' from the given
;- X value, depending on the direction the object was
;- told to go.
;-
;- Tweaked Registers: possibly r0, r1 (X and Y positions)
;------------------------------------------------------------
sub_x:   CMP   R_X_POS_EN ,0x80    ; see if you can move
         BREQ  done1
         SUB   R_X_POS_EN,0x01    ; move if you can
done1:   RET

add_x:   CMP   R_X_POS_EN, MAX_X    ; see if you can move
         BREQ  done3
         ADD   R_X_POS_EN,0x01    ; move if you can
done3:   RET


;------------------------------------
; Subroutine get_obj_data
; Loads object data (X_POS, Y_POS, and color)
; from the stack based on address in r4
;
; R_ARGUMENT (r31) - Stack address
;------------------------------------
get_obj_data:
         LD    R_X_POS_EN, (r31)
         ADD   R_ARGUMENT, 0x01
         LD    R_Y_POS,    (r31)
         ADD   R_ARGUMENT, 0x01
         LD    R_RGB_DATA, (r31)
         SUB   R_ARGUMENT, 0x02
         RET

;------------------------------------
; Subroutine set_obj_data
; Stores object data onto the stack based on address in r4
; Uses 3 memory words
;
; R_ARGUMENT (r31) - Stack address
;------------------------------------
set_obj_data:
         ST    R_X_POS_EN, (r31)
         ADD   R_ARGUMENT, 0x01
         ST    R_Y_POS,    (r31)
         ADD   R_ARGUMENT, 0x01
         ST    R_RGB_DATA, (r31)
         SUB   R_ARGUMENT, 0x02
         RET

;------------------------------------
; Subroutine update_obj
;
; r0 - X_POS_EN
; r1 - Y_POS
; r2 - RGB_DATA
; r3 - OBJ_ADDR

;------------------------------------
update_obj:
         MOV      r4, R_OBJ_ADDR       ;r4 is temp address
         OUT      r0, X_POS_EN_ID
         OUT      r1, Y_POS_ID
         OUT      r2, RGB_DATA_ID
         OUT      r4, OBJ_ADDR_ID
         MOV      r4, 0
         OUT      r4, OBJ_ADDR_ID
         RET

;---------------------------------------------------------------
; Interrup Service Routine - Handles Interrupts from keyboard
;---------------------------------------------------------------
; handles interupts from the keyboard and calls the appropriate
; subroutine based on which key was pressed
;
; Tweaked Registers; r6, r15
;---------------------------------------------------------------
ISR:      CMP   r15, int_flag        ; check key-up flag
          BRNE  continue
          MOV   r15, 0x00            ; clean key-up flag
          BRN   reset_ps2_register

continue: IN    r6, PS2_KEY_CODE_ID     ; get keycode data
          OUT  r6, SSEG_ID

move_left:
          CMP   r6, LEFT
          BRNE  move_right
          CALL  sub_x                ; verify move
          CALL  update_obj             ; draw object
          CALL  set_obj_data
          BRN   reset_ps2_register

move_right:
          CMP   r6, RIGHT
          BRNE  fire_bullet
          CALL  add_x                 ; verify move
          CALL  update_obj              ; draw object

          CALL  set_obj_data
          BRN   reset_ps2_register

fire_bullet:
      CMP r6, SPACE
      BRNE key_up_check
      CALL fire
      BRN reset_ps2_register

key_up_check:
          CMP   r6,KEY_UP             ; look for key-up code
          BRNE  reset_ps2_register    ; branch if not found

set_skip_flag:
          ADD   r15, 0x01             ; indicate key-up found

reset_ps2_register:                   ; reset PS2 register
          MOV    r6, 0x01
          OUT    r6, PS2_CONTROL_ID
          MOV    r6, 0x00
          OUT    r6, PS2_CONTROL_ID
        RETIE
;--------------------------------------------------------------------

;----------------------------------------------------------------------
; interrupt vector
;----------------------------------------------------------------------
.CSEG
.ORG 0x3FF
           BRN   ISR
;----------------------------------------------------------------------
```