Imperial College London

Physics Department

Microprocessor Project

Constructing a Gaming System with ATmega128 Microcontroller

Quang Nguyen

SCIENTIA IMPERII DECUS ET TUTAMEN

Abstract

Based on the Atmel ATmega128 microcontroller, a gaming system was constructed with graphic and sound. The microcontroller's timers were programmed to handle time-critical functions of the game. Conversely, the microcontroller's external interrupt was utilised to detect the user's input. With DACs (Digital-Analog Converters) and an oscilloscope, graphics were created to complete the game. Ultimately, the gaming system met expectations in graphic and music production. The ATmega128 microcontroller was found to be capable of producing simple gaming applications. Improvements were planned to upgrade the graphic and gameplay of the system.

1 Introduction

Microcontrollers are small computers on single integrated circuits. They are extremely useful in performing specific tasks without consuming much power [1]. With their unique characteristics, microcontrollers are prevalent in modern living. They can be found in most electrical appliances including washing machine, microwave cookers and remote control television [2]. However, in contrast to general-purpose computers, microcontrollers are much more discreet and often hidden away from view.

Microcontrollers were naturally suitable for arcade gaming applications. Arcade games often have repeating gameplay and limited tasks to perform. It was not surprising that arcade game machines adopted these tiny computers soon after their first appearance and remained so till the late 2000s when computer-based systems became popular [3]. As a result, this project was set out to replicate an arcade game machine. With a ATmega128 micro-controller and inexpensive electrical components, a gaming system was constructed.

1.1 The Microcontroller

ATmega128 is a 8-bit CMOS microcontroller with 8 MHz clock speed and 32 general-purpose working registers. The microcontroller provides a perfect platform to make arcade games. Its external interrupts can obtain human inputs while its many output ports can provide peripheral interfaces. In addition, the microcontroller's 4 timers are capable of producing sound signals and time-critical routines.

1.2 The Game

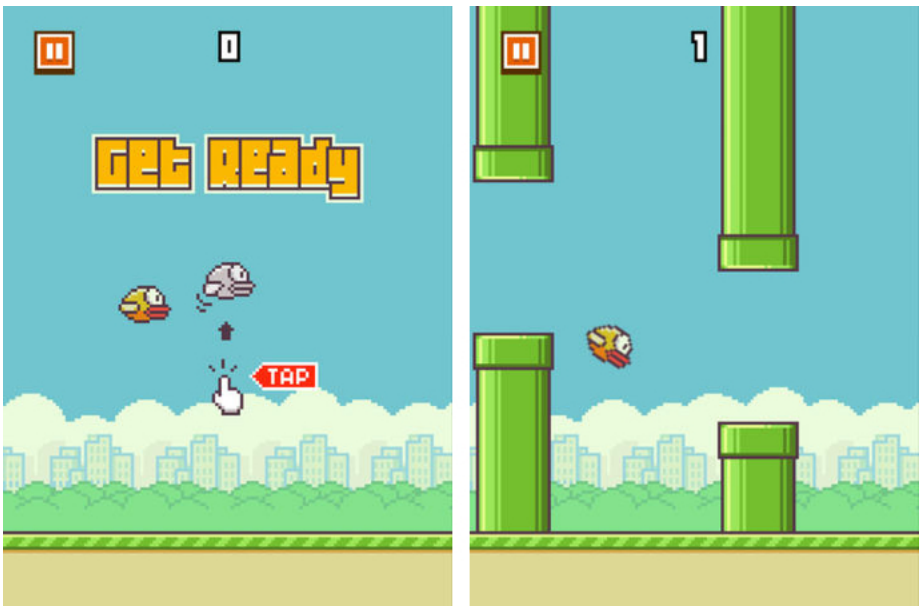

Figure 1: Snap shots of the original Flappy Bird game. The snap shots were taken (left) during the standby screen and (right) during gameplay.

The game chosen for this project was the Flappy Bird game. Developed in 2013 by Dong Nguyen, the Flappy Bird game challenges the players to fly an orange bird through rows of green pipes. Collision with the pipes results in a crash. The Flappy Bird game was hugely successful with over 50 million downloads and hitting the top of 53 countries' app stores [4]. Gameplay was simple with the tubes move towards the bird horizontally. The bird accelerates towards the ground, only being stopped when players press on the screen. Navigation through the tubes earns players points. Ultimately, the players will try to beat their own high score or their friends' achievements. Apart from its simple yet addictive nature, the Flappy Bird game was also chosen based on its suitability for an arcade game machine. The game contains distinct phases with standby, gameplay and game-end.

2 High Level Design

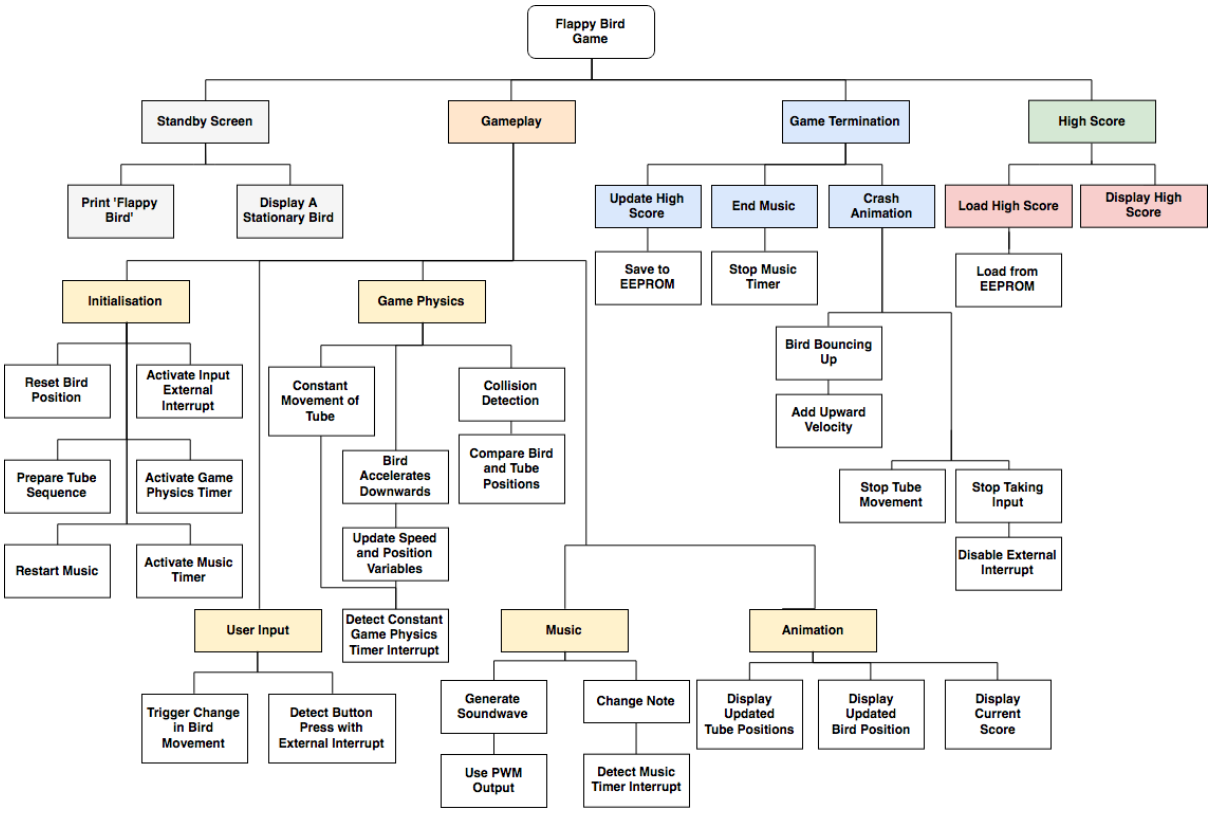

Figure 2: High level design for the Flappy Bird gaming system.

Similar to the original Flappy Bird game, the game created was divided into three distinct phases: standby, gameplay and game-end (as illustrated on Figure 3). The three phases have their own characteristics and thus very different designs and functions. Nevertheless, the current score and high score are displayed at all time in the game.

The standby phase consists of a stationary bird and the 'Flappy Bird' word displayed. This is done to prepare the player for the game and can last indefinitely. During the gameplay, the programme is first initialised. The music is reset to start from the start. The timers and external interrupts are set up accordingly. In addition, the tube positions are prepared for the game. The bird is set to centre and its speed is set to zero. Subsequently, user input, game physics, animation and music work together to produce gameplay. One timer will provide time signals to create physical changes to the game which include the constant movement of the tube as well as the fall of the bird. Another timer will provide a PWM (Pulse-Width-Modulation) signal to create sound signals while an additional timer controls the duration of the music notes. During the game, the bird and the tubes will always be present on screen. Special comparison functions will detect collisions of the bird. Upon crashing, a series of crash animation

will follow. Firstly, the music timer will be disabled to stop all music. Secondly, the bird will be propelled into the air before falling down. Thirdly, all tube movement will be stopped. Last but not least, the current score will be compared with the high score and if that is higher, the high score will be updated.

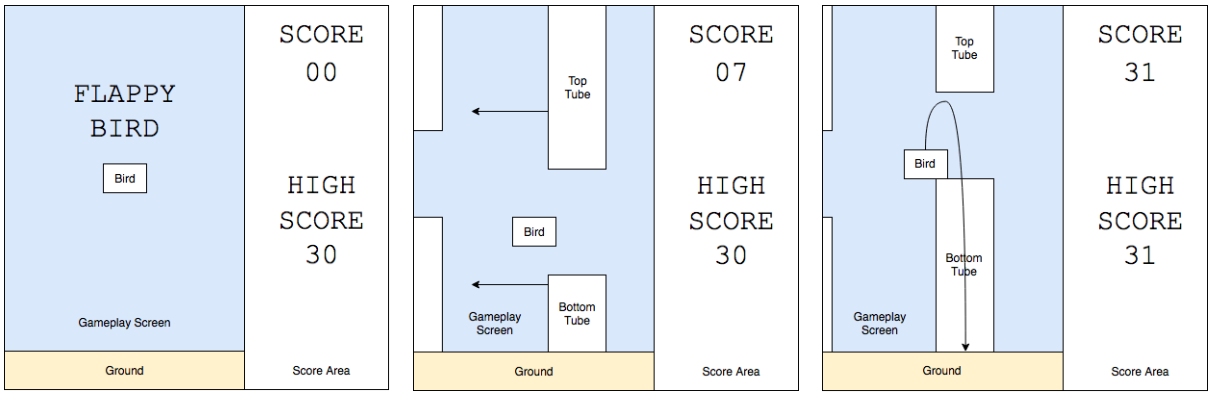

Figure 3: Mock-ups of the game graphics (left) during standby, (centre) during gameplay, (right) during game termination.

3 Software and Hardware Design

3.1 Graphics

A Hitachi Oscilloscope V-555 was employed to provide visual interface for the game. X-Y mode was selected and X-Channel and Y-Channel were made to interface with the microcontroller with the help of DACs (Digital-Analog Converters). The 8-bit TLC7524 DACs help to convert ATmega128 digital output signals to analog forms, which is subsequently amplified by op-ams (operational amplifiers). The analog signals help to navigate the cursor of the oscilloscope around its screen. The DACs and Op-Amps were connected according to Figure 4. As the amplifiers were constructed in inverting mode, the output signal is inverted. Thus the coordinates system is inverted as shown in Figure 5.

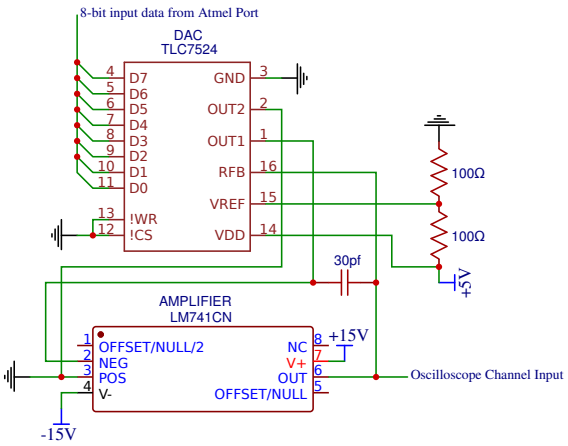

Figure 4: Circuit diagram for the interface between the oscilloscope and ATmega128.

Changes in input signals shift the cursor on the oscilloscope screen. A line can be seen if the cursor moves quickly enough. Creation of a rectangle was easily achieved with the drawing of 4 different straight lines. The rectangle was the basis of many drawings on the game. As a result, 4 unique registers were utilised to facilitate the drawing of rectangles. TempRightX3, TempLeftY4, TempTopLength5, TempBottom6 contain the information about the right, left, top and bottom bounds of the rectangle

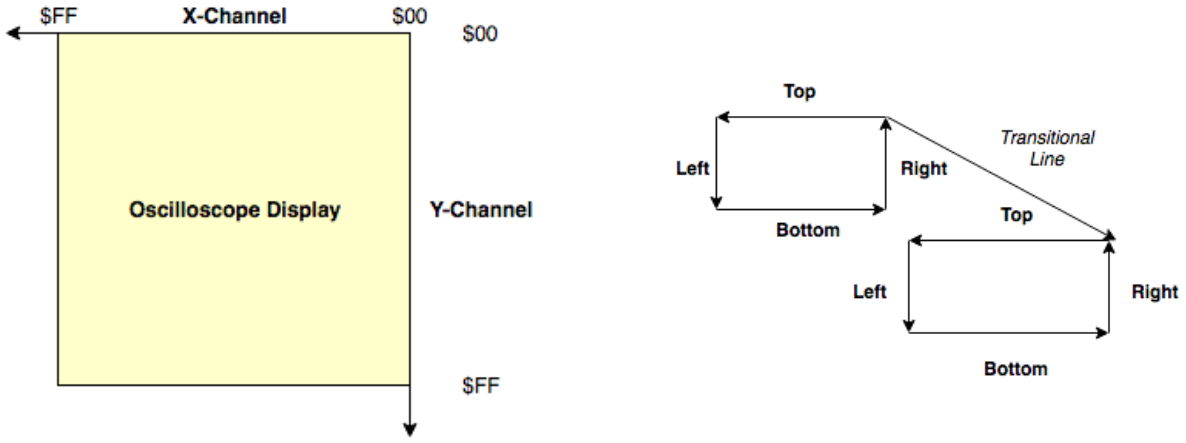

Figure 5: Illustrations of the game graphical interface: (left) coordinate system for the oscilloscope cursor, (right) drawing of rectangles on the display requiring transitional line.

respectively. With the 4 registers set, routine TubeOut can be called to output the rectangle on the screen (detailed in Listing 1).

Listing 1: Implementation in Assembly for drawing of a rectangle on the oscilloscope screen.

```
TubeOut:
    cp TempRightX3, TempLeftY4
    brsh tubedone

    mov TempReg, TempRightX3

    out PORTD, TempTopLength5
    out PORTC, TempReg

    cbi PortB, 2          ; Enable Brightness
    rcall BigDel

Top:
    inc TempReg
    out PORTD, TempTopLength5
    out PORTC, TempReg

    cp TempReg, TempLeftY4
    brne Top

    mov TempReg, TempTopLength5

Left:
    inc TempReg
    out PORTD, TempReg
    cp TempReg, TempBottom6
    brne Left

    mov TempReg, TempLeftY4

Bottom:
    dec TempReg
    out PORTC, TempReg
    cp TempReg, TempRightX3
    brne Bottom

    mov TempReg, TempBottom6

Right:
    dec TempReg
    out PORTD, TempReg

```

```
    cp TempReg, TempTopLength5
    brne Right

    sbi PortB, 2          ; Disable Brightness
    rcall BigDel
    ret
```

However, since any movement of the cursor will be observed on the screen by the player, transitional lines (shown in Figure 5) will be also be seen. Viewing of the transitional lines is undesirable. As such, the graphic was cleaned up with the use of the oscilloscope's blanking input. A high voltage blanking input will dim the the cursor's brightness. In contrast, a low voltage blanking input will increase the light intensity. As a result, upon ending of rectangle drawing, the blanking input voltage is raised high before reducing to zero when another rectangle is produced. The transitional lines will be dimmer and thus not visible to the eyes at correct intensity settings. Pin 2 of PortB was used to output blanking signal. The pin was connected to a non-inverting amplifier to amplify the voltage (shown in Figure 6). Changes to brightness were achieved with the commands in Listing 2.

Listing 2: Assembly commands to change the brightness of light on the oscilloscope screen.

```
    cbi PortB, 2          ; Enable Brightness
    sbi PortB, 2          ; Disable Brightness
```

The larger the voltage, the dimmer transitional lines. With the maximum input for the blanking input being 50V and the output of the pin ranging from 0-5V, the resistor values were set for a gain factor of 10. However, the maximum voltage possible with the amplifier was found to be 15V with V+ input being the limiting factor. Nevertheless, +15V was determined to be sufficient to dim the transitional lines. Taking into account of the amplifier's slew rate, a delay has to be inserted to wait for the amplifier voltage to settle. The delay, BigDel shown in Listing 3 with a waiting period of 10 cycles was tested to be the best compromise between dimming transitional lines and producing bright rectangles.

Listing 3: Delay routine BigDel to wait for amplifier voltage to settle.

```
BigDEL:
        ldi ZH, HIGH(10)
        ldi ZL, LOW (10)
CountBigDel:
        sbiw ZL, 1
        brne CountBigDel
        ret
```

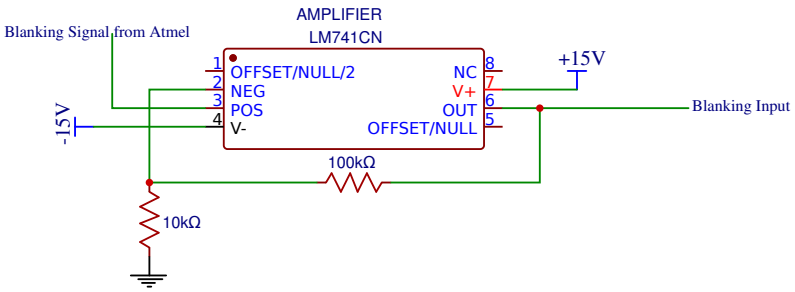

Figure 6: Circuit diagram for blanking input from the ATmega128 to the oscilloscope.

The ground, the screen frame and the bird are made up of rectangle drawings. Each tube row contains tubes at the top and the bottom of the screen (shown in Figure 7). At all time, a maximum of two tube rows will appear simultaneously on the screen. As such, the parameters of 2 tubes are saved in the SRAM under the name of Tubex1, Tubey1, Tubex2 and Tubey2. Tubexn contains information on the horizontal positions of the tube rows while Tubeyn contains information on the height of the openings of the tubes. During the gameplay, if the tubes appear to have crossed the screen boundaries, routines in Tube (as shown in Listint 4) will modify TempRightX3, and TempLeftY4 in order to produce the impression of the tube coming out from the right boundary and disappearing to the left boundary.

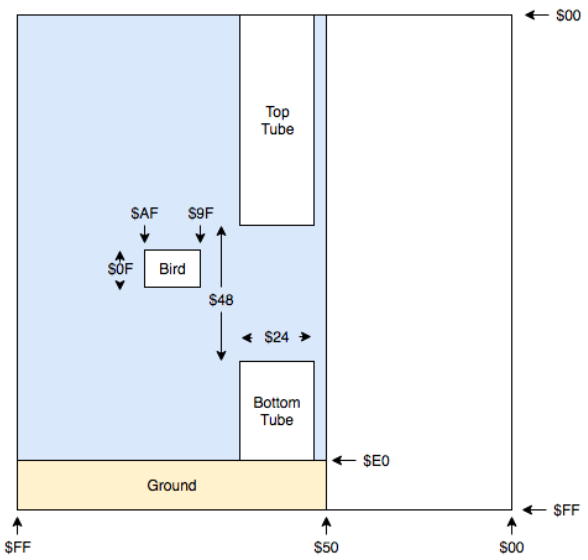

Figure 7: Important dimensions of the game display, bird and tubes.

Listing 4: Checking and modification routines for the tubes to animate their motion.

```
Tube:
    ldi TempReg, $24          ;
    lds TempRightX3, Tubex1      ; Extract Tube1 X-Coordinates
    lds TempLeftY4, Tubex1       ;
    add TempLeftY4, TempReg      ; Calculate Tube1 Left

    cpi TempLeftY4, $50          ;
    brlo compare1           ; If TubeLeft exceeds the right of display area, go to
    compare1
    rjmp notcompare1          ; If TubeLeft does not exceed the right of display area, go
    to notcompare1

compare1:
    cpi TempRightX3, $50         ;
    brlo tube2              ; If TubeRight does not exceed the left of display area, go to
     tube2
    ldi TempLeftY4, $FF          ; If TubeRight exceed the left of display area, set
    TubeLeft to left display boundary

notcompare1:
    cpi TempRightX3, $50         ;
    brsh tube1out           ; If TubeRight does exceed the right of display area, go to
    tube1out
    ldi TempRightX3, $50         ; If TubeRight exceed the right of display area, set
    TubeRight to right display boundary

tube1out:
    lds TempTopLength5, Tubey1     ; Extract Tube1 Height
    ldi TempBottom6, $E0         ; Calculate Bottom Tube1 Height
    rcall TubeOut             ; Display Bottom Tube1

    ldi TempTopLength5, $00      ;
    lds TempBottom6, Tubey1      ;
    subi TempBottom6, $48        ; Calculate Top Tube1 Height
    rcall TubeOut             ; Display Top Tube1

tube2:
    ldi TempReg, $24          ;
    lds TempRightX3, Tubex2      ; Extract Tube2 X-Coordinates
    lds TempLeftY4, Tubex2       ;
    add TempLeftY4, TempReg      ; Calculate Tube2 Left

```

```
    cpi TempLeftY4, $50         ;
    brlo compare2           ; If TubeLeft exceeds the right of display area, go to
    compare2
    rjmp notcompare2        ; If TubeLeft does not exceed the right of display area, go
    to notcompare2

compare2:
    cpi TempRightX3, $50        ;
    brlo tubedone           ; If TubeRight does not exceed the left of display area, go
    to tubedone
    ldi TempLeftY4, $FF         ; If TubeRight exceed the left of display area, set
    TubeLeft to left display boundary


notcompare2:
    cpi TempRightX3, $50        ;
    brsh tube2out           ; If TubeRight does exceed the right of display area, go to
    tube2out
    ldi TempRightX3, $50        ; If TubeRight exceed the right of display area, set
    TubeRight to right display boundary

tube2out:
    lds TempTopLength5, Tubey2      ; Extract Tube2 Height
    ldi TempBottom6, $E0        ; Calculate Bottom Tube2 Height
    rcall TubeOut           ; Display Bottom Tube1

    ldi TempTopLength5, $00       ;
    lds TempBottom6, Tubey2       ;
    subi TempBottom6, $48       ; Calculate Top Tube1 Height
    rcall TubeOut           ; Display Top Tube2

tubedone:
    ret
```

The letters and numbers are created from horizontal and vertical lines. Each character has its own routine to draw out such as OneOut and POut. Their positional placements are defined in register NumPositionx and NumPositiony. The information from the registers are extracted to draw the horizontal and vertical lines with routines HorizontalOut and VerticalOut as shown in Listing 5.

Listing 5: Assembly routines to draw horizontal and vertical lines.

```
;**************** Display Horizontal Line *****************
HorizontalOut:
    mov TempReg2, TempRightX3
    add TempReg2, TempTopLength5
    mov TempReg, TempRightX3
    out PORTC, TempReg
    out PORTD, TempLeftY4
    cbi PortB, 2            ; Enable Brightness
    rcall BigDel


HorizontalAgain:
    inc TempReg
    out PORTC, TempReg
    cp TempReg, TempReg2
    brne HorizontalAgain
    sbi PortB, 2            ; Disable Brightness
    rcall BigDel
    ret

;**************** Display Vertical Line ******************
VerticalOut:
    mov TempReg2, TempLeftY4
    add TempReg2, TempTopLength5
    mov TempReg, TempLeftY4
    out PORTC, TempRightX3
    out PORTD, TempReg
    cbi PortB, 2            ; Enable Brightness
    rcall BigDel
```

```
30
31
32 VerticalAgain:
33     inc TempReg
34     out PORTD, TempReg
35     cp  TempReg, TempReg2
36     brne VerticalAgain
37     sbi PortB, 2          ; Disable Brightness
38     rcall BigDel
39     ret
```

3.2 Scoring System

The scoring system is essential to the gameplay. Scores allow players to keep track of their progress as well as measure their mastery of the game. The current score is recorded in the Score register. On the other hand, the high score is kept in an EEPROM address, High Score. Such arrangement was needed as high score has to be persistent during power off. High score is recorded and read with the EEPROM_read and EEPROM_write routines obtained from the ATmega128 manual.

One problem encountered with the scoring system was that the scores are recorded in hex, but displayed in decimal for the user interface. A hex2dec routine (shown in Listing 6) was created to convert the hex number to decimal number. The routine outputs the decimal digits as the last digit of register TempReg and TempBottom6. Routine DigitOut was then carried out to determine the identity of the digit and output on the screen.

Listing 6: Assembly routine to covert heximal numbers to decimal digits.

```
1  ;*************** Convert Hex to Decimal ******************
2  hex2dec:
3      ldi TempReg2, $00
4      ldi TempBottom6, $00
5      ldi TempReg2, -1 + '0'
6  _bib3:
7      inc     TempReg2
8          subi    TempReg, low(100)
9          sbci    TempBottom6, high(100)
10         brcc    _bib3
11
12         ldi     TempBottom6, 10 + '0'
13 _bib4:
14     dec     TempBottom6
15         subi    TempReg, -10
16         brcs    _bib4
17
18         subi    TempReg, -'0'
19     ret
```

3.3 Game Levels

The game difficulty is created by the random positions of the tube openings. Generation of random tube heights require random numbers. A pseudo-random Linear Congruential random number generator was created. The number generator follows the following equation

$$X_{n+1} = (aX_n + c) \bmod m \tag{1}$$

where m, a and c are constants and X_n is the random number. A linear congruential generator produce a repeating random number sequence given a starting seed X_0. As such, the constants were carefully chosen to produce a long enough sequence such that the repetition is unrecognisable but short enough for the tubes' height difference to be challenging after scaling. Following human trials, the appropriate values for the constants were found to be $a = 18$, $c = 5$ and $m = 17$.

In order to reduce the computational time, the linear congruential sequence is recorded into table LevelDesign. Register Seed was then utilised to keep track of the position in the sequence. Consecutive players will continue from where the last player stop in the sequence. To the player, the height of the tube appears to be random.

3.4 Game Physics

With the bird being the central component of the game, its motion is crucial to portraying realism to the player. Realistic physics helps player getting used to the game more quickly, and thus lowering the barrier to entry of the game. As the bird is programmed to be falling, its motion was coded to be be governed classic equations of kinematics under effect of gravity. The speed of the bird v and the position of the bird y follow Equations 2 and 3.

$$\frac{dy}{dt} = v \tag{2}$$

$$\frac{dv}{dt} = g \tag{3}$$

where g is the gravitational constant and also the constant acceleration experienced by the bird. Subsequently, Finite Difference Method was applied to calculate the speed and the position of the bird at all time. Euler's Forward Method was applied to Equations 2 and 3.

$$\frac{dy}{dt} \approx \frac{y(t+\delta t) - y(t)}{\delta t} = v \tag{4}$$

$$\frac{dv}{dt} \approx \frac{v(t+\delta t) - v(t)}{\delta t} = g \tag{5}$$

Rearranging the terms, the following are obtained:

$$y(t+\delta t) = y(t) + v\delta t \tag{6}$$

$$v(t+\delta t) = v(t) + g\delta t \tag{7}$$

Euler's Forward is more accurate with smaller step size. Timer1A is programmed to trigger frequently, as such the motion of the bird will well-approximated. Each interrupt cycle, the position and speed of the bird are updated. $g = 1$ velocity unit/interrupt in the programme. Implementation of the formulas in Assembly is presented in Listing 7. Similar to during the gameplay, the motion of the bird is subjected to the same algorithm during crash animation. To add retro touches to the game, the crash animation took inspiration from the famous Mario game. In the Mario game, upon crashing, the protagonist jumps up before falling to the bottom of the screen. Similarly, the bird was programmed to fly upwards before falling down when collided with obstacles. Such motion was achieved by adding upward speed to the bird when it crashed.

Listing 7: Implementation in Assembly for the kinematics equations.

```
1    ldi TempReg, $01          ;
2    add BirdSpeed, TempReg       ; Update Bird Speed
3    add BirdPosition, BirdSpeed    ; Update Bird Position
```

Motion of the tube consists of them moving horizontally 1 unit/interval. The constant separation between the tubes was produced by delaying the Tube2's spawning 110 intervals after the spawn of Tube1. Bird-Tube collision detection was achieved with the use of comparison functions. The flow chart for the collision detection system is shown in Figure 8. The detection routine is carried out for both Tube1 and Tube2. In addition, Ground and ceiling detection was also programmed.

Upon detecting a crash, the game will be transferred to the crash state. In there the animation will go on for 48 more intervals (≈ 3 seconds) before resetting to the main screen. The constant time interval between frame updates are produced with Counter1 time signals. The timer was set up in CTC mode and Compare1A Interrupt was used to flag InterruptState which indicates the time interval.

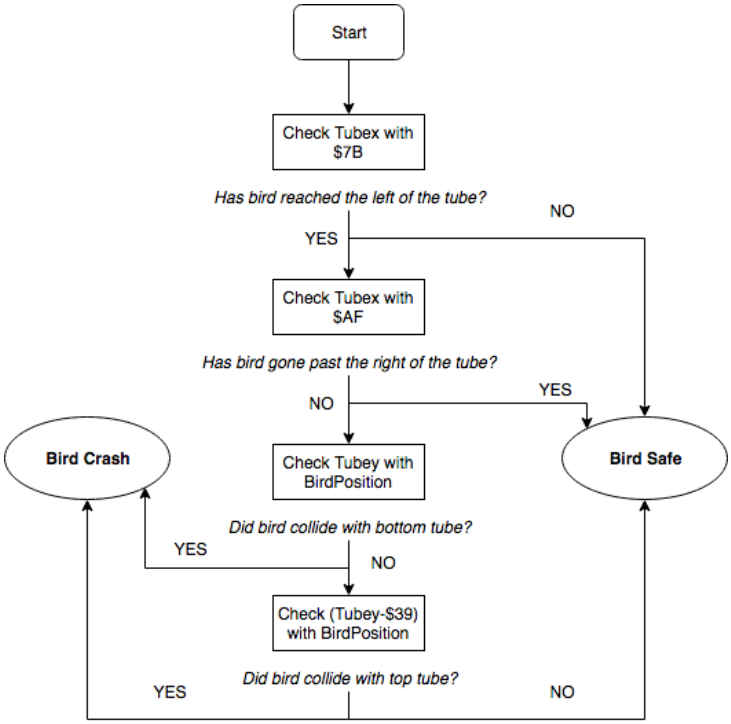

Figure 8: Collision Detection Flow Chart.

3.5 Player's Input

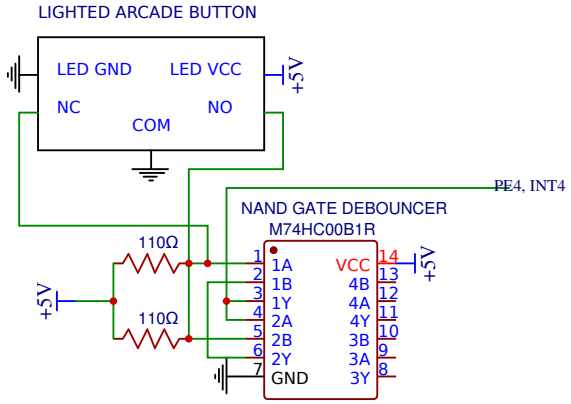

Figure 9: Electronic connection with the button.

The player's input comes from a lighted arcade push button. Its output is connected to External Interrupt 4 at Pin4 of PortE. External Interrupt 4 was set to only trigger upon falling edge. Due to phenomenon of bouncing, the signal from the push button can cause many falling edges and creating multiple presses from one input from the player. In order to eliminate problem, a SR debouncer was constructed from NAND gates (shown in Figure 9) [5]. The NAND gates were connected to NC (Normally Closed) and NO (Normally Open) port of the switch. As result, when the button is not pressed, the output is constantly only 1. On the other hand, when the button is pressed, the input is 0. With each button press, the bird will be given an additional 6 units of upwards speed. The value of speed addition was identified to be most friendly to the players, giving them enough control over the bird fall and maneuverability around the tubes.

3.6 Music

The music of the game was derived from the Mario theme song. The notes and tempo were obtained from `https://wiki.mikrotik.com/wiki/Super_Mario_Theme`. Modification was made to lower the octave for a lower and mellower tone. Using the frequency and timing information from the website, the song was created with the use of 2 timers: Timer3 which creates the note with its PWM operation, and Timer0 which controls the notes' timing.

Timer3 was set up in PWM, Phase and Frequency Correct Mode. In this configuration, the counter resets at the value of OCR3A. The timer's compare output 3A was set to toggle on compare match. As such, the frequency of the produced signal will be modified with changes to the value of OCR3A. Timer3 was pre-scaled by 1 as such, its default signal is 8MHz. To convert from the frequency value to the correct value for OCR3A, the following formula was used:

$$\text{OCR3A} = \frac{F}{2 * f} \tag{8}$$

where $F = 8\text{MHz}$ is the microprocessor's clock frequency and f is the frequency of the note.

Timer0 is set up in CTC mode and was used to produced a constant-period signal. The number of times Timer 0 signal produced is kept track by register TimeCounter. Note timing information is saved in NoteTimes table. The time duration value is extracted from the table and saved to CurrentTime. CurrentTime dictates how long the note or silence will be last. When CurrentTime matches with TimeCounter, the frequency of the music note will be change. At the transition, the note frequency is extracted from MusicNotes table and is programmed to Timer3's OCR3A. Due to the many music notes in MusicNotes, the pointer for the table, NotePointer, has to be 4-byte long. As such, it occupies SRAM address 0205 for the low byte and $040A$ for the high byte.

The full hardware design for the game system is presented in Figure 10.

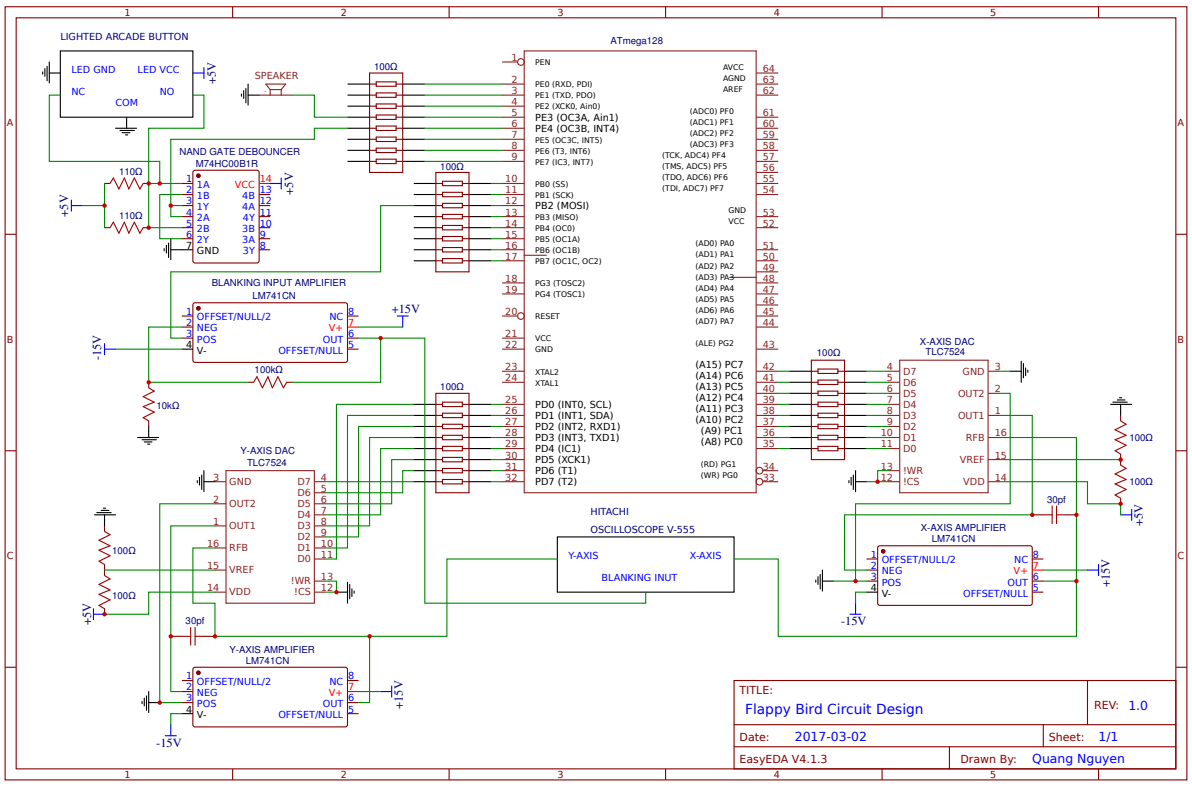

Figure 10: Full circuit design for the game system.

4 Results and Performance

The game successfully created. Figure 11 show the final circuit construction for the electronics components.

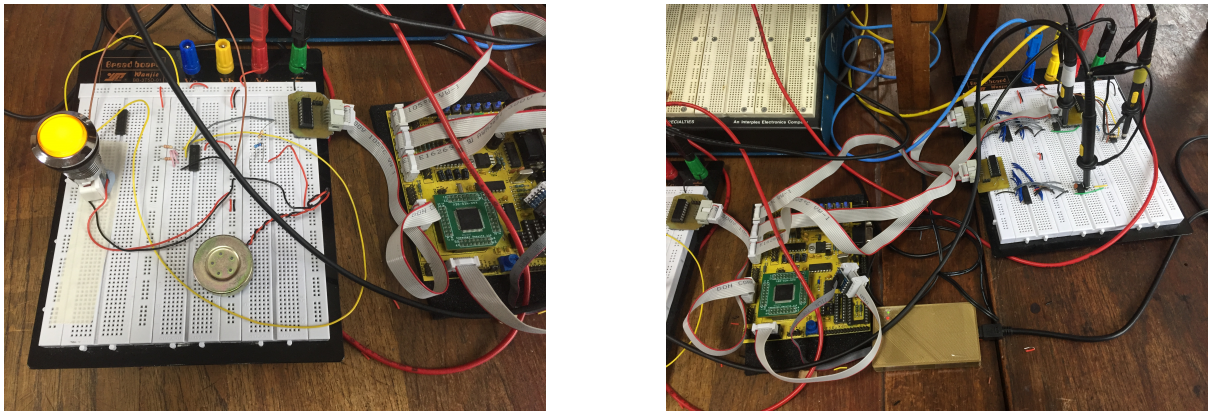

Figure 11: Pictures of the final electronic circuit with (left) connection to the lighted arcade switch and speaker and (right) connection to the oscilloscope channels and blanking input.

The resultant graphics for the game shown in Figure 12 are clear to the players without any transitional lines. The words displayed were also legible and readable by the players.

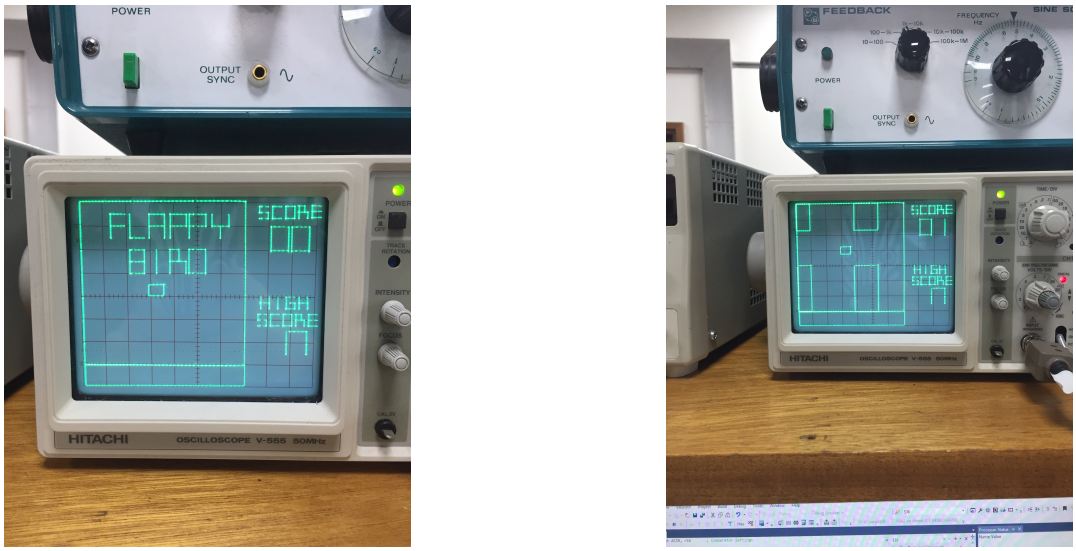

Figure 12: Pictures of the oscilloscope graphics (left) during standby and (right) during gameplay.

The scoring system also worked as expected and the high score remains persistent on the gaming system after multiple power cycles. In addition, the music played loud and clear. Even though the tempo was different from the original song, many players recognised the Mario theme song. The cause of the problem was attributed to the wrong timing information rather than software or hardware problems. The system engaged many players with all of them finding joy from playing the game. With feedback from the players, the pace of game and the physics were adjusted to make it more playable. The project was considered a success by the team.

5 Updates, Modifications and Improvements

In order to make the gaming system better, improvements were planned for both the graphics and the gameplay. Firstly, a picture of the bird can be drawn to replace the rectangle. This will make the name of the game 'Flappy Bird' more relevant. In doing so, drawing routines of diagonal lines and circles will have to be created. Secondly, a background can be created with clouds moving behind the bird. This will add another realism dimension to the game. However, in order to display the clouds as another layer to the display, depth of the image has to be created. Upgrading the blanking input was identified to be the necessary modification to add depth. Currently, the brightness is binary (on or off). However, by upgrading the blanking input to a DAC system, different level of brightness can be achieved. With such improvement, the clouds can be drawn faintly in the background without confusing the players whether or not that is an obstacle. Lastly, the gameplay's difficulty can be increased with gameplay time. Such task can be achieved by having a timer and a set of time checkpoints. Once the players have played for a certain amount of time, OCR1A can be reduced to speed up the game, making it harder for the player.

6 Conclusions

In conclusion, the framework for a gaming system was created with a Flappy Bird game installed. The gaming system was created and tested for both its inputs and outputs which include music and graphics. The game was proven to be a success with players finding joy from playing. The music played well while the graphics was adequate for playing. Improvements were planned to upgrade the graphics of the system and to the gameplay. The project has also provided a good test for the capability of ATmega128 microcontroller. The game was completed with plenty of excess resources. The project proved that the ATmega128 microcontroller is a good platform to build arcade game systems.

7 Product Specifications

- 1 x Atmel ATmega128
- 1 x Speaker
- 1 x Lighted Arcade Button
- 3 x LM741CN Amplifier
- 2 x TLC7524 DAC
- 4 x 4116RLF 100Ω Resistor Packs
- 1 x M74HC00B1R NAND Gates
- 2 x 30pf Capacitors
- 4 x 100Ω Resistors
- 2 x 110Ω Resistors
- 2 x Protoboards
- 1 x Hitachi Oscilloscope V-555

References

[1] Mayank. *What is a microcontroller? And how does it differ from a microprocessor?* `http://maxembedded.com/2011/06/mcu-vs-mpu/`. 2011.

[2] Atmel. *Home Appliances* `http://www.atmel.com/applications/homeappliances/default.aspx`. 2016.

[3] Jason Neckert. *Arcade game CPU timeline* http://triosdevelopers.com/jason.eckert/blog/Entries/2012/9/2_Arcade_game_CPU_timeline.html. 2012.

[4] Christina Warren. *28 Days of Fame: The Strange, True Story of 'Flappy Bird'* http://mashable.com/2014/02/10/flappy-bird-story/#fuNtmGxQggq1. 2014.

[5] Jack G. Ganssle. *A Guide to Debouncing.* The Ganssle Group, 3 edition, 2017.

Appendices

Full Assembly Code

```
;
; FlappyBird.asm
;
; Created: 07/02/2017 10:00:19
;

; Author : Quang Nguyen, Chris Page
; Institution : Imperial College Lodon
;
.DEVICE ATmega128
.include "m128def.inc"
;
    .ORG  $0

    jmp Init          ;
    nop reti       ; External 0 interrupt  Vector
    nop reti       ; External 1 interrupt  Vector
    nop reti       ; External 2 interrupt  Vector
    nop reti       ; External 3 interrupt  Vector
    jmp Button       ; External 4 interrupt  Vector
    nop reti       ; External 5 interrupt  Vector
    nop reti       ; External 6 interrupt  Vector
    nop reti       ; External 7 interrupt  Vector
    nop reti       ; Timer 2 Compare Vector
    nop reti       ; Timer 2 Overflow Vector
    nop reti       ; Timer 1 Capture  Vector
    jmp GameInter      ; Timer 1 CompareA  Vector
    nop reti       ; Timer 1 CompareB  Vector
    nop reti       ; Timer 1 Overflow  Vector
    jmp MusicInter     ; Timer 0 Compare  Vector
    nop reti       ; Timer 0 Overflow interrupt  Vector
    nop reti       ; SPI  Vector
    nop reti       ; UART Receive  Vector
    nop reti       ; UDR Empty  Vector
    nop reti       ; UART Transmit  Vector
    nop reti       ; ADC Conversion Complete Vector
    nop reti       ; EEPROM Ready Vector
    nop reti       ; Analog Comparator  Vector

.org    $0080      ; start address well above interrupt table



;*******************************************************
;***************                              **********
;***************      VARIABLE SETUP          **********
;***************                              **********
;*******************************************************

;************** Non-Immediate Registers ******************
    .def CurrentLevel = r3  ; Level Position Indicator
    .def ClockCounter = r5  ; Number of Game Interrupts occured
    .def TimeCounter  = r6  ; Number of Music Timer Interrupts occured
    .def TimePointer  = r7  ; Music Time Position Indicator
    .def CurrentTime  = r8  ; Curent Music Time

    .def BirdPosition = r9  ; Bird Position
    .def BirdSpeed    = r10 ; Bird Speed

    .def NumPositionx = r11 ; Number Position X-Coordinate
    .def NumPositiony = r12 ; Number Position Y-Coordinate

;****************** Immediate Registers ******************
    .def TempReg    = r16 ; Temporary Register 1
    .def TempReg2   = r17 ; Temporary Register 2
```

```
66
67      .def TempRightX3  = r18 ; Tube Right Bound/Line Start X-Coordinate
68      .def TempLeftY4   = r19 ; Tube Left Bound/Line Start Y-Coordinate
69      .def TempTopLength5 = r20 ; Tube Top Bound/Line Length
70      .def TempBottom6  = r21 ; Tube Bottom Bound
71
72      .def InterruptState = r23 ; Game Timer Interrupt
73      .def Crash      = r24 ; Crash State
74      .def ButtonState  = r25 ; Button State
75
76  ;************************ SRAM   **********************
77      .equ Tubex1     = $0200 ; Tube 1 Position X-Coordinate
78      .equ Tubey1     = $0201 ; Tube 1 Position Y-Coordinate
79      .equ Tubex2     = $0202 ; Tube 2 Position X-Coordinate
80      .equ Tubey2     = $0203 ; Tube 2 Position Y-Coordinate
81
82      .equ Score      = $0204 ; Current Score
83      .equ NotePointer  = $0205 ; Music Note Position Indicator
84
85  ;************************ EEPROM ***********************
86      .equ HighScore    = $0200 ; High Score
87
88    rjmp Init             ; Jump over Database and Intertupt to Initialisation
89
90
91
92  ;*******************************************************
93  ;***************                         ****************
94  ;***************         DATABASE        ****************
95  ;***************                         ****************
96  ;*******************************************************
97
98  ;******************** Level Design **********************
99    LevelDesign:
100     .db $6C,$98,$C4,$5A,$86,$B2,$49,$75,$A1,$CD,$63,$8F,$BB,$51,$7D,$AA,$D6,$6C
101   MusicNotes:
102     .dw $0BD6,$0000,$0BD6,$0000,$0BD6,$0000,$0F51,$0000,$0BD6,$0000,$0A25,$0000,$148F,
      $0000,$0F51,$0000,$148F,$0000,$1869,$0000,$11C1,$0000,$1046,$0000,$115C,$0000,$122B,
      $0000,$148F,$0000,$0BD6,$0000,$0A47,$0000,$0915,$0000,$0B29,$0000,$0A47,$0000,$0BD6,
      $0000,$0F06,$0000,$0D78,$0000,$1046,$0000,$0F51,$0000,$148F,$0000,$1869,$0000,$11C1,
      $0000,$1046,$0000,$115C,$0000,$122B,$0000,$148F,$0000,$0BD6,$0000,$0A47,$0000,$0915,
      $0000,$0B29,$0000,$0A47,$0000,$0BD6,$0000,$0F06,$0000,$0D78,$0000,$1046,$0000,$0F9F,
      $0000,$0A47,$0000,$0AD9,$0000,$0B7D,$0000,$0C99,$0000,$0C04,$0000,$148F,$0000,$122B,
      $0000,$0F9F,$0000,$122B,$0000,$0F9F,$0000,$0DB4,$0000,$0F9F,$0000,$0A47,$0000,$0AD9,
      $0000,$0B7D,$0000,$0C99,$0000,$0C04,$0000,$07A8,$0000,$07A8,$0000,$07A8,$0000,$148F,
      $0000,$0F9F,$0000,$0A47,$0000,$0AD9,$0000,$0B7D,$0000,$0C99,$0000,$0C04,$0000,$148F,
      $0000,$122B,$0000,$0F9F,$0000,$122B,$0000,$0F9F,$0000,$0DB4,$0000,$0D5A,$0000,$0E34,
      $0000,$0F9F,$0000,$148F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0A47,
      $0000,$0AD9,$0000,$0B7D,$0000,$0C99,$0000,$0C04,$0000,$148F,$0000,$122B,$0000,$0F9F,
      $0000,$122B,$0000,$0F9F,$0000,$0DB4,$0000,$0F9F,$0000,$0A47,$0000,$0AD9,$0000,$0B7D,
      $0000,$0C99,$0000,$0C04,$0000,$07A8,$0000,$07A8,$0000,$07A8,$0000,$148F,$0000,$0F9F,
      $0000,$0A47,$0000,$0AD9,$0000,$0B7D,$0000,$0C99,$0000,$0C04,$0000,$148F,$0000,$122B,
      $0000,$0F9F,$0000,$122B,$0000,$0F9F,$0000,$0DB4,$0000,$0D78,$0000,$0E34,$0000,$0F9F,
      $0000,$148F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,
      $0000,$0F9F,$0000,$0D78,$0000,$0BD6,$0000,$0F9F,$0000,$122B,$0000,$148F,$0000,$0F9F,
      $0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0D78,$0000,$0BD6,$0000,$08FA,$0000,$0A47,
      $0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0F9F,$0000,$0D78,$0000,$0BD6,$0000,$0F9F,
      $0000,$122B,$0000,$148F,$0000,$0BD6,$0000,$0BD6,$0000,$0BD6,$0000,$0F51,$0000,$0BD6,
      $0000,$0A25,$0000,$148F
103   NoteTimes:
104     .db $18, $25, $18, $18, $4A, $18, $18, $18, $18, $4A, $18, $89, $18, $8F, $18, $70,
      $18, $63, $18, $7C, $18, $4A, $13, $52, $18, $25, $18, $4A, $18, $31, $13, $31, $C,
      $25, $18, $4A, $13, $25, $0C, $57, $13, $4A, $13, $25, $13, $25, $13, $7C, $18, $70,
       $18, $63, $18, $7C, $18, $4A, $13, $52, $18, $25, $18, $4A, $18, $31, $13, $31, $0C
      , $25, $18, $4A, $13, $25, $0C, $57, $13, $4A, $13, $25, $13, $25, $13, $7C, $18,
      $4A, $18, $18, $18, $25, $18, $25, $25, $4A, $25, $4A, $18, $25, $18, $25, $18, $4A,
       $18, $25, $18, $18, $18, $36, $18, $4A, $18, $18, $18, $25, $18, $25, $25, $4A, $31
      , $4A, $13, $4A, $13, $25, $13, $4A, $18, $4A, $18, $4A, $18, $18, $18, $25, $18,
      $25, $25, $4A, $25, $4A, $18, $25, $18, $25, $18, $4A, $18, $25, $18, $18, $18, $68,
       $18, $70, $18, $68, $18, $59, $18, $4A, $18, $4A, $18, $25, $18, $4A, $18, $4A, $18
```

```
    , $18, $18, $25, $18, $25, $25, $4A, $25, $4A, $18, $25, $18, $25, $18, $4A, $18,
    $25, $18, $18, $18, $36, $18, $4A, $18, $18, $18, $25, $18, $25, $25, $4A, $31, $4A,
     $13, $4A, $13, $C, $13, $4A, $18, $4A, $18, $4A, $18, $18, $18, $25, $18, $25, $25,
     $4A, $25, $4A, $18, $25, $18, $25, $18, $4A, $18, $25, $18, $18, $18, $68, $18, $70
    , $18, $68, $18, $59, $18, $4A, $18, $4A, $18, $25, $18, $4A, $0E, $25, $13, $4A,
    $0E, $57, $13, $25, $13, $57, $13, $25, $13, $4A, $13, $25, $13, $95, $0E, $25, $13,
     $4A, $0E, $57, $13, $25, $13, $25, $13, $89, $13, $51, $13, $95, $0E, $25, $13, $4A
    , $0E, $57, $13, $25, $13, $57, $13, $25, $13, $4A, $13, $25, $13, $95, $18, $25,
    $18, $4A, $18, $4A, $18, $18, $18, $4A, $18, $89, $18, $8F
105
106
107
108 ;*******************************************************
109 ;***************                              **********
110 ;***************    INTERRUPT ROUTINES        **********
111 ;***************                              **********
112 ;*******************************************************
113
114 ;*************** Game Timer Interrupts ******************
115 GameInter:
116     in r4,SREG          ; save SREG
117     ldi InterruptState, $FF ; indicate Game Timer Interrupt has been triggered
118     out SREG,r4         ; restore SREG
119     reti            ; return
120
121 ;************** External Button Interrupts **************
122 Button:
123     in r4,SREG          ; save SREG
124     ldi ButtonState, $FF  ; indicate Button External Interrupt has been triggered
125     out SREG,r4         ; restore SREG
126     reti            ; return
127
128 ;************** External Button Interrupts **************
129 MusicInter:
130     in r4,SREG          ; save SREG
131     inc TimeCounter       ; indicate Music Timer Interrupt has been triggered
132     out SREG,r4         ; restore SREG
133     reti            ; return
134
135
136
137 ;*******************************************************
138 ;***************                              **********
139 ;***************      INITIALISATION          **********
140 ;***************                              **********
141 ;*******************************************************
142
143 Init:
144 ;***************** Stack Pointer Setup ******************
145     ldi r16, $0F        ; Stack Pointer Setup
146     out SPH,r16         ; Stack Pointer High Byte
147     ldi r16, $FF        ; Stack Pointer Setup
148     out SPL,r16         ; Stack Pointer Low Byte
149
150 ;******************** RAMPZ Setup ***********************
151     ldi  r16, $00       ; 1 = EPLM acts on upper 64K
152     out RAMPZ, r16        ; 0 = EPLM acts on lower 64K
153
154 ;**************** Comparator Setup Code *****************
155     ldi r16,$80         ; Comparator Disabled, Input Capture Disabled
156     out ACSR, r16       ; Comparator Settings
157
158 ;****************** TIMER0 Setup Code *******************
159     ldi r16,$0D         ; CTC Mode
160     out TCCR0, r16        ; Timer - PRESCALE TCK0 BY 256
161                   ; clear timer on OCR0 match
162     ldi r16,$90         ; load OCR0 with n=125
163     out OCR0,r16        ; The counter will go every n*256*125 nsec
164
165 ;****************** TIMER1A Setup Code ******************
```

```
166     ldi r16,$00         ; Normal Operation
167     out TCCR1A, r16       ;
168     ldi r16,$0F         ; CTC Mode, Prescale 1024
169     out TCCR1B, r16       ;
170     ldi r16,$00         ; High byte of OCR1A
171     out OCR1AH,r16        ;
172     ldi r16,$80         ; Low byte of OCR1A
173     out OCR1AL,r16        ;
174
175 ;****************** TIMER3A Setup Code ******************
176     ldi r16,$41         ; Normal Operation
177     sts TCCR3A, r16       ;
178     ldi r16,$11         ; PWM Mode
179     sts TCCR3B, r16       ;
180     ldi r16,$00         ; High byte of OCR3A
181     sts OCR3AH,r16        ;
182     ldi r16,$00         ; Low byte of OCR3A
183     sts OCR3AL,r16        ;
184
185 ;****************** PORTB Setup Code ******************
186     ldi r16, $FF        ; OUTPUT (BLANKING INPUT)
187     out DDRB , r16        ; Port B Direction Register
188     ldi r16, $FF        ;
189     out PORTB, r16        ;
190
191 ;****************** PORTC Setup Code ******************
192     ldi r16, $FF        ; OUTPUT (X-AXIS)
193     out DDRC , r16        ; Port C Direction Register
194     ldi r16, $FF        ;
195     out PORTC, r16        ;
196
197 ;****************** PORTD Setup Code ******************
198     ldi r16, $FF        ; Port D OUTPUT (Y-AXIS)
199     out DDRD, r16       ; Port D Direction Register
200     ldi r16, $FF        ;
201     out PORTD, r16        ;
202
203 ;****************** PORTE Setup Code ******************
204     ldi r16, $0F        ; 0-3 INPUT (USER INPUT), 4-7 OUTPUT (SPEAKER)
205     out DDRE, r16       ; Port E Direction Register
206     ldi r16, $FF        ; Port E Pull-up Register
207     out PORTE, r16        ;
208
209 ;************** EXTERNAL INTERRUPT 4 Setup Code **********
210     ldi r16, $10        ; Enable INT4
211     out EIMSK, r16        ;
212     ldi r16, $02        ; Trigger Interrupt on falling edge
213     out EICRB, r16        ;
214
215   sei             ; Enable All Interrupts
216
217
218
219 ;*******************************************************
220 ;***************                     *****************
221 ;***************     MAIN PROGRAMME  *****************
222 ;***************                     *****************
223 ;*******************************************************
224
225 ;***************** Level Initialisation ******************
226     ldi TempReg, $01        ; Reset Level Position Indicator upon booting
227     mov CurrentLevel, TempReg   ;
228
229 ;****************** Standby Screen ***********************
230 Main:
231     rcall Ground            ; Display Ground
232     ldi TempReg, $7F        ; reset Bird Position
233     mov BirdPosition, TempReg     ;
234     rcall Bird              ; Display Bird
235     rcall HighScoreOut        ; Display High Score
```

```
236     rcall ScoreOut                  ; Display Current Score
237     rcall FlappyOut             ; Display 'FLAPPY BIRD'
238     sbic PINE, 4                ; If Button is pressed, initialise Game
239     rjmp GameInit               ;
240
241     ldi TempReg2, LOW(HighScore)   ; Load EEPROM Address of High Score
242     ldi TempRightX3, High(HighScore);
243     rcall EEPROM_read           ; Read High Score from EEPROM
244     lds TempReg2, Score         ; Compare Current Score with High Score
245     cp TempReg2, TempReg        ;
246     ldi TempReg, $00            ; Reset Current Score to 0
247     sts Score, TempReg          ;
248     brlo Main                   ; If Current Score is lower, continue Standby Screen
249     mov TempReg, TempReg2       ; If Current Score is higher, load new High Score
250     ldi TempReg2, LOW(HighScore)   ; Load EEPROM Address of High Score
251     ldi TempRightX3, High(HighScore);
252     rcall EEPROM_write          ; Write new High Score to EEPROM
253     rjmp Main                ; When done, continue Standby Screen
254
255 ;*************** Game Initialisation ********************
256 GameInit:
257
258     ldi TempReg, $00            ;
259     sts Tubex1, TempReg         ; Reset Tubes1 X-Coordinates to zero
260     sts Tubex2, TempReg         ; Reset Tubes1 X-Coordinates to zero
261
262     ldi TempReg, $7F            ;
263     sts Tubey1, TempReg         ; Reset Tubes 1 Height to centre
264     sts Tubey2, TempReg         ; Reset Tubes 2 Height to centre
265     mov BirdPosition, TempReg   ; Reset Bird Position to centre
266
267     ldi TempReg, $00            ;
268     mov BirdSpeed, TempReg      ; Reset Bird Speed to zero
269     mov InterruptState, TempReg    ; Reset Game Interrupt State to None
270     mov Crash, TempReg          ; Reset Crash State to None
271     mov ClockCounter, TempReg      ; Reset Game Interrupt Counter to zero
272     mov CurrentTime, TempReg    ; Reset Current Music Time to zero
273     ldi XH,HIGH(NotePointer)    ; Load NotePointer SRAM Position
274     ldi XL,LOW(NotePointer)     ;
275     st X, TempReg               ; Reset Music Note Position to zero
276     mov TimeCounter, TempReg        ; Reset Music Interrupt Counter to zero
277     mov TimePointer, TempReg    ; Reset Music Time Position to zero
278
279     ldi r16, $12                ; Enable Game Timer and Music Timer Interrupts
280     out TIMSK, r16              ;
281
282 ;****************** GamePlay ****************************
283 Gameplay:
284     rcall Ground                ; Display Ground
285     rcall Bird                  ; Display Bird
286     rcall Tube                  ; Display Tube
287     rcall ScoreOut              ; Display Score
288     rcall HighScoreOut          ; Display High Score
289     cpi InterruptState, $FF     ; Check for Game Timer Interrupt
290     breq Move                ; If Game Interrupt happened, Move
291
292 FlyDone:
293     LDI ZH, HIGH(2*NoteTimes)   ; Load NoteTimes table
294     LDI ZL, LOW(2*NoteTimes)    ;
295     ldi TempReg, $00            ;
296     add ZL, TimePointer         ; Move Z-Pointer to Current Music Time Position
297     adc ZH, TempReg             ;
298     lpm CurrentTime, Z          ; Extract Current Music Time
299     cp TimeCounter, CurrentTime    ; Compare Music Interrupt Counter with Music Time
300     brlo NoteDone               ; If Counter is lower, do note change Music Note
301     rcall NoteChange            ; If Counter matches, change Musics Note
302
303 NoteDone:
304     cpi Crash, $FF              ;
305     breq Crashed                ; If bird was crashed, go to crash animation
```

```
306     rjmp GamePlay           ; If bird was not crashed, continue gameplay
307
308 ;****************** Crash Animation **********************
309 Crashed:
310     ldi TempReg, $0F        ;
311     cp BirdPosition, TempReg    ; Check whether bird hits ceiling
312     ldi TempReg, $FA        ;
313     mov BirdSpeed, TempReg      ; If bird does not hit ceiling, add upward speed
314     brsh TopCheckDone           ;
315     ldi TempReg, $0F        ;
316     mov BirdPosition, TempReg     ; If bird hit ceiling, stop bird at ceiling
317     ldi TempReg, $00        ; If bird hit ceiling, reset Bird Speed to zero
318     mov BirdSpeed, TempReg      ;
319
320 TopCheckDone:
321     ldi TempReg, $00        ;
322     mov ClockCounter, TempReg     ; Disable Music
323     sts OCR3AH,TempReg          ;
324     sts OCR3AL,TempReg          ;
325
326
327 CrashScreen:
328     rcall Ground            ; Display Ground
329     rcall Bird              ; Display Bird
330     rcall Tube              ; Display Tube
331     rcall ScoreOut            ; Display Score
332     rcall HighScoreOut          ; Display High Score
333     cpi InterruptState, $FF     ; Check for Game Timer Interrupt
334     breq CrashMove          ; If Game Interrupt happened, CrashMove
335
336 CrashMoveDone:
337     ldi TempReg, $30        ;
338     cp TempReg, ClockCounter    ; Compare ClockCounter with $30
339     breq CrashDone          ; If Clock Counter reaches $30, stop crash animation
340     rjmp CrashScreen        ; If Clock Counter lower than $30, continue crash animation
341
342 CrashMove:
343     ldi InterruptState, $00     ; Reset Timer Interrupt State
344     inc ClockCounter        ; Increase Clock Counter
345     ldi TempReg, $01        ;
346     add BirdSpeed, TempReg      ; Update Bird Speed
347     add BirdPosition, BirdSpeed   ; Update Bird Position
348     ldi TempReg, $E0        ;
349     cp BirdPosition, TempReg    ; Check whether bird hits Ground
350     brlo CrashTopCheck        ; If bird higher than Ground, check whether bird hits
    ceiling
351     mov BirdPosition, TempReg     ; If bird hits Ground, stop bird at Ground
352     ldi TempReg, $00        ;
353     mov BirdSpeed, TempReg      ; If bird hits Ground, reset Bird Speed to zero
354     rjmp CrashMoveDone        ; Continue crash sequence
355
356 CrashTopCheck:
357     ldi TempReg, $0F        ;
358     cp BirdPosition, TempReg    ; Check whether bird hits ceiling
359     brsh CrashMoveDone        ; If bird does not hit ceiling, continue crash sequence
360     mov BirdPosition, TempReg     ; If bird hit ceiling, stop bird at ceiling
361     ldi TempReg, $00        ;
362     mov BirdSpeed, TempReg      ; If bird hit ceiling, reset Bird Speed to zero
363     rjmp CrashMoveDone        ; Continue crash sequence
364
365
366 CrashDone:
367     ldi r16, $00            ; Disable Game Timer and Music Timer Interrupts
368     out TIMSK, r16            ;
369     rjmp Main               ; Go to Standby Screen
370
371
372
373 ;*****************************************************
374 ;***************                          *****************
```

```
;***************       GAME PHYSICS        *****************
;***************                           *****************
;*********************************************************

;****************** Movement Physics ********************
Move:
    inc ClockCounter          ; Increase number of Game Interrupts Occured
    lds TempReg, Tubex1         ;
    cpi TempReg, $00          ; Check whether Tube1 X-Coordinate is at zero
    brne move1              ; If Tube1 is already moving, go to move1
    ldi TempReg, $01          ;
    cp ClockCounter, TempReg    ; If Tube1 is not already moving, move it after 1 Game
    Interrupt
    brne movedone             ;
    ldi TempReg, $2B          ; Shift Tube1 X-Coordinate to $2B

move1:
    inc TempReg             ; Move Tube1 horizontally by 1
    cpi TempReg, $FF          ;
    brne move1done            ; If Tube1 does not reach the end of the screen, go to
    move1done

    LDI ZH, HIGH(2*LevelDesign)   ; Load LevelDesign table
    LDI ZL, LOW(2*LevelDesign)    ;
    ldi TempReg, $00          ;
    add ZL, CurrentLevel        ; Move Z-Pointer to Current LevelDesign Position
    adc ZH, TempReg           ;
    lpm                 ; Extract Current LevelDesign Height
    inc CurrentLevel          ; Increase Current LevelDesign Pointer
    ldi TempReg, $11          ;
    cp CurrentLevel, TempReg    ; If Current LevelDesign Pointer reaches the end of the
    sequence,
    ldi TempReg, $01          ; reset it back to 1
    brne modify1b           ;
    mov CurrentLevel, TempReg     ;
modify1b:
    sts Tubey1, R0            ; Change the height of Tube1
    ldi TempReg, $2C          ; Reset X-Coordinate of Tube1 to right of the screen

move1done:
    sts Tubex1, TempReg         ; Change X-Coordinate of Tube1

    lds TempReg, Tubex2         ;
    cpi TempReg, $00          ; Check whether Tube2 X-Coordinate is at zero
    brne move2              ; If Tube2 is already moving, go to move2
    ldi TempReg, $6F          ;
    cp ClockCounter, TempReg    ; If Tube2 is not already moving, move it after 111 Game
     Interrupts
    brne movedone             ;
    ldi TempReg, $2B          ; Shift Tube2 X-Coordinate to $2B

move2:
    inc TempReg             ; Move Tube1 horizontally by 1
    cpi TempReg, $FF          ;
    brne move2done            ; If Tube1 does not reach the end of the screen, go to
    move2done

    LDI ZH, HIGH(2*LevelDesign)   ; Load LevelDesign table
    LDI ZL, LOW(2*LevelDesign)    ;
    ldi TempReg, $00          ;
    add ZL, CurrentLevel        ; Move Z-Pointer to Current LevelDesign Position
    adc ZH, TempReg           ;
    lpm                 ; Extract Current LevelDesign Height
    inc CurrentLevel          ; Increase Current LevelDesign Pointer
    ldi TempReg, $11          ;
    cp CurrentLevel, TempReg    ; If Current LevelDesign Pointer reaches the end of the
    sequence,
    ldi TempReg, $01          ; reset it back to 1
    brne modify2b             ;
    mov CurrentLevel, TempReg     ;
```

```
439 modify2b:
440     sts Tubey2, R0          ; Change the height of Tube2
441     ldi TempReg, $2C        ; Reset X-Coordinate of Tube2 to right of the screen
442
443 move2done:
444     sts Tubex2, TempReg       ; Change X-Coordinate of Tube1
445
446 movedone:
447     ldi TempReg, $7B        ;
448     lds TempReg2, Tubex1      ;
449     cp TempReg2, TempReg      ;
450     breq incscore           ; If bird has moved across the Tube1, increase the score
451     lds TempReg2, Tubex2      ;
452     cp TempReg2, TempReg      ;
453     breq incscore           ; If bird has moved across the Tube2, increase the score
454
455 incscoredone:
456     ldi InterruptState, $00     ; Reset InterruptState to None
457     cpi ButtonState, $FF      ; Check whether the button has been pressed
458     breq Flap           ; If Pressed, got to Flap
459 Flapdone:
460     ldi TempReg, $01
461     add BirdSpeed, TempReg      ; Update Bird Speed
462     add BirdPosition, BirdSpeed   ; Update Bird Position
463
464     ldi TempReg, $E0        ;
465     cp BirdPosition, TempReg    ;
466     brsh HitGround          ; If bird has hit the Ground, go to HitGround
467
468     ldi TempReg, $0F        ;
469     cp BirdPosition, TempReg    ;
470     brlo BirdCrash          ; If bird has hit the ceiling, go to BirdCrash
471
472 checktube1:
473     ldi TempReg, $7B        ;
474     lds TempReg2, Tubex1      ;
475     cp TempReg2, TempReg      ; Check if bird has reached Tube1
476     brlo checktube2         ; If bird has not reached Tube1, go to checktube2
477     ldi TempReg, $AF        ;
478     cp TempReg2, TempReg      ; Check if bird is still within reach of Tube1
479     brsh checktube2         ; If bird has gone beyond reach of Tube1, go to checktube2
480     lds TempReg2, Tubey1      ;
481     cp BirdPosition, TempReg2     ;
482     brsh BirdCrash          ; If bird has hit Tube1 bottom opening, go to BirdCrash
483     mov TempReg, TempReg2       ;
484     subi TempReg, $39         ;
485     cp BirdPosition, TempReg    ;
486     brlo BirdCrash          ; If bird has hit Tube1 top opening, go to BirdCrash
487
488 checktube2:
489     ldi TempReg, $7B        ;
490     lds TempReg2, Tubex2      ;
491     cp TempReg2, TempReg      ; Check if bird has reached Tube2
492     brlo checktubedone        ; If bird has not reached Tube2, go to checktubedone
493     ldi TempReg, $AF        ;
494     cp TempReg2, TempReg      ; Check if bird is still within reach of Tube2
495     brsh checktubedone        ; If bird has gone beyond reach of Tube2, go to
    checktubedone
496     lds TempReg2, Tubey2      ;
497     cp BirdPosition, TempReg2     ; If bird has hit Tube2 bottom opening, go to
    BirdCrash
498     brsh BirdCrash          ;
499     mov TempReg, TempReg2       ;
500     subi TempReg, $39         ;
501     cp BirdPosition, TempReg    ;
502     brlo BirdCrash          ; If bird has hit Tube2 top opening, go to BirdCrash
503
504 checktubedone:
505     rjmp FlyDone          ; When all tubes have been checked, go to FlyDone
506
```

```
HitGround:
    ldi TempReg, $E0          ;
    mov BirdPosition, TempReg     ; If bird has hit Ground, it remains on Ground
    ldi Crash, $FF            ; Crash State changed to True
    rjmp FlyDone            ; Go to FlyDone

BirdCrash:
    ldi Crash, $FF            ; Crash State changed to True
    rjmp FlyDone            ; Go to FlyDone

;******************** User Input **********************
Flap:
    ldi ButtonState, $00      ; Reset Button State to None
    ldi TempReg, $F9
    mov BirdSpeed, TempReg      ; Add vertival upward speed to bird
    rjmp Flapdone           ; Go to Flap Done


incscore:
    lds TempReg, Score          ;
    inc TempReg             ; Increase Score
    sts Score, TempReg          ; Save to SRAM
    rjmp incscoredone           ;

;****************** Music Output **********************
NoteChange:
    ldi TempReg, $00          ;
    mov TimeCounter, TempReg    ;
    LDI ZH, HIGH(2*MusicNotes)    ; Load MusicNotes table
    LDI ZL, LOW(2*MusicNotes)     ;
    ldi XH,HIGH(NotePointer)      ; Load Low Byte of NotePointer SRAM Position to X
    ldi XL,LOW(NotePointer)     ;
    ld TempReg, X             ; Extract Low Byte of NotePointer to TempReg
    add ZL, TempReg           ; Add Low Byte of NotePointer to Z-Pointer
    ldi XH,HIGH(2*NotePointer)    ; Load High Byte of NotePointer SRAM Position to X
    ldi XL,LOW(2*NotePointer)     ;
    ld TempReg, X             ; Extract High Byte of NotePointer to TempReg
    adc ZH, TempReg           ; Add with carry Low Byte of NotePointer to Z-Pointer
    lpm TempReg, Z+           ; Extract Musics Note Low Byte
    lpm TempReg2, Z           ; Extract Musics Note High Byte
    sts OCR3AH,TempReg2         ; Change PWM period High Byte
    sts OCR3AL,TempReg          ; Change PWM period Low Byte
    inc TimePointer           ; Increase TimePointer

    ldi XH,HIGH(NotePointer)      ;
    ldi XL,LOW(NotePointer)     ;
    ld TempReg, X             ;
    ldi TempReg2, $02           ;
    add TempReg, TempReg2       ; Increase NotePointer by 2
    st X, TempReg             ;
    ldi XH,HIGH(2*NotePointer)    ;
    ldi XL,LOW(2*NotePointer)     ;
    ld TempReg, X             ;
    ldi TempReg2, $00           ;
    adc TempReg, TempReg2       ;
    st X, TempReg             ; Store Note Pointer to its SRAM Position
    ldi TempReg2, $01           ; If NotePointer reaches 340,
    cpse TempReg, TempReg2      ;
    ret                 ;
    ldi XH,HIGH(NotePointer)    ;
    ldi XL,LOW(NotePointer)     ;
    ld TempReg, X             ;
    ldi TempReg2, $54           ;
    cpse TempReg, TempReg2      ;
    ret                 ;
    ldi TempReg, $00          ; Reset NotePointer to 0
    st X, TempReg             ;
    ldi XH,HIGH(2*NotePointer)    ;
    ldi XL,LOW(2*NotePointer)     ;
    ldi TempReg, $00          ;
```

```
577     st X, TempReg           ;
578     clr TimePointer         ;
579     ret
580
581
582
583 ;*******************************************************
584 ;***************                          ***************
585 ;***************       ANIMATION          ***************
586 ;***************                          ***************
587 ;*******************************************************
588
589 ;******************* Display Score *********************
590 ScoreOut:
591     ldi TempReg, $05        ;
592     mov NumPositiony, TempReg     ;
593     ldi TempReg, $0C        ;
594     mov NumPositionx, TempReg     ;
595     rcall EOut              ; Display 'E'
596
597     ldi TempReg, $18        ;
598     mov NumPositionx, TempReg     ;
599     rcall ROut              ; Display 'R'
600
601     ldi TempReg, $24        ;
602     mov NumPositionx, TempReg     ;
603     rcall OOut              ; Display 'O'
604
605     ldi TempReg, $30        ;
606     mov NumPositionx, TempReg     ;
607     rcall COut              ; Display 'C'
608
609     ldi TempReg, $3C        ;
610     mov NumPositionx, TempReg     ;
611     rcall SOut              ; Display 'S'
612
613     lds TempReg, Score        ;
614     rcall hex2dec           ; Convert Score from Hex to Decimal
615
616     cbi PortB, 2            ; Enable Brightness
617     ldi TempLeftY4, $15       ;
618     mov NumPositionx, TempLeftY4  ;
619     ldi TempLeftY4, $20       ;
620     mov NumPositiony, TempLeftY4  ;
621     rcall DigitOut          ; Display the Lower Digit
622
623     ldi TempLeftY4, $2A       ;
624     mov NumPositionx, TempLeftY4  ;
625     mov TempReg, TempBottom6    ;
626     rcall DigitOut          ; Display the Higher Digit
627     ret               ;
628
629
630 ;*************** Display High Score *********************
631 HighScoreOut:
632     ldi TempReg, $80        ;
633     mov NumPositiony, TempReg     ;
634     ldi TempReg, $16        ;
635     mov NumPositionx, TempReg     ;
636     rcall HOut              ; Display 'H'
637
638     ldi TempReg, $22        ;
639     mov NumPositionx, TempReg     ;
640     rcall GOut              ; Display 'G'
641
642     ldi TempReg, $2E        ;
643     mov NumPositionx, TempReg     ;
644     rcall IOut              ; Display 'I'
645
646     ldi TempReg, $3A        ;
```

```
647     mov NumPositionx, TempReg      ;
648     rcall HOut              ; Display 'H'
649
650     ldi TempReg, $98        ;
651     mov NumPositiony, TempReg      ;
652     ldi TempReg, $0C        ;
653     mov NumPositionx, TempReg      ;
654     rcall EOut              ; Display 'E'
655
656     ldi TempReg, $18        ;
657     mov NumPositionx, TempReg      ;
658     rcall ROut              ; Display 'R'
659
660     ldi TempReg, $24        ;
661     mov NumPositionx, TempReg      ;
662     rcall OOut              ; Display 'O'
663
664     ldi TempReg, $30        ;
665     mov NumPositionx, TempReg      ;
666     rcall COut              ; Display 'C'
667
668     ldi TempReg, $3C        ;
669     mov NumPositionx, TempReg      ;
670     rcall SOut              ; Display 'S'
671
672     ldi TempReg2, LOW(HighScore)   ; Load High Score EEPROM Address
673     ldi TempRightX3, High(HighScore);
674     rcall EEPROM_read          ; Read High Score from EEPROM
675     rcall hex2dec              ; Convert High Score from Hex to Decimal
676
677     cbi PortB, 2            ; Enable Brightness
678     ldi TempLeftY4, $15        ;
679     mov NumPositionx, TempLeftY4   ;
680     ldi TempLeftY4, $B0        ;
681     mov NumPositiony, TempLeftY4   ;
682     rcall DigitOut             ; Display Lower Digit
683
684     ldi TempLeftY4, $2A        ;
685     mov NumPositionx, TempLeftY4   ;
686     mov TempReg, TempBottom6     ;
687     rcall DigitOut             ; Display Higher Digit
688     ret                ;
689
690 ;******************** Display Tube **********************
691 Tube:
692     ldi TempReg, $24        ;
693     lds TempRightX3, Tubex1      ; Extract Tube1 X-Coordinates
694     lds TempLeftY4, Tubex1       ;
695     add TempLeftY4, TempReg      ; Calculate Tube1 Left
696
697     cpi TempLeftY4, $50        ;
698     brlo compare1          ; If TubeLeft exceeds the right of display area, go to
        compare1
699     rjmp notcompare1         ; If TubeLeft does not exceed the right of display area, go
        to notcompare1
700
701 compare1:
702     cpi TempRightX3, $50       ;
703     brlo tube2             ; If TubeRight does not exceed the left of display area, go to
         tube2
704     ldi TempLeftY4, $FF        ; If TubeRight exceed the left of display area, set
        TubeLeft to left display boundary
705
706 notcompare1:
707     cpi TempRightX3, $50       ;
708     brsh tube1out          ; If TubeRight does exceed the right of display area, go to
        tube1out
709     ldi TempRightX3, $50       ; If TubeRight exceed the right of display area, set
        TubeRight to right display boundary
710
```

```
711 tube1out:
712     lds TempTopLength5, Tubey1     ; Extract Tube1 Height
713     ldi TempBottom6, $E0       ; Calculate Bottom Tube1 Height
714     rcall TubeOut            ; Display Bottom Tube1
715
716     ldi TempTopLength5, $00      ;
717     lds TempBottom6, Tubey1      ;
718     subi TempBottom6, $48        ; Calculate Top Tube1 Height
719     rcall TubeOut            ; Display Top Tube1
720
721 tube2:
722     ldi TempReg, $24         ;
723     lds TempRightX3, Tubex2      ; Extract Tube2 X-Coordinates
724     lds TempLeftY4, Tubex2       ;
725     add TempLeftY4, TempReg      ; Calculate Tube2 Left
726
727     cpi TempLeftY4, $50        ;
728     brlo compare2          ; If TubeLeft exceeds the right of display area, go to
        compare2
729     rjmp notcompare2         ; If TubeLeft does not exceed the right of display area, go
        to notcompare2
730
731 compare2:
732     cpi TempRightX3, $50       ;
733     brlo tubedone            ; If TubeRight does not exceed the left of display area, go
        to tubedone
734     ldi TempLeftY4, $FF        ; If TubeRight exceed the left of display area, set
        TubeLeft to left display boundary
735
736
737 notcompare2:
738     cpi TempRightX3, $50       ;
739     brsh tube2out          ; If TubeRight does exceed the right of display area, go to
        tube2out
740     ldi TempRightX3, $50       ; If TubeRight exceed the right of display area, set
        TubeRight to right display boundary
741
742 tube2out:
743     lds TempTopLength5, Tubey2     ; Extract Tube2 Height
744     ldi TempBottom6, $E0       ; Calculate Bottom Tube2 Height
745     rcall TubeOut            ; Display Bottom Tube1
746
747     ldi TempTopLength5, $00      ;
748     lds TempBottom6, Tubey2      ;
749     subi TempBottom6, $48      ; Calculate Top Tube1 Height
750     rcall TubeOut            ; Display Top Tube2
751
752 tubedone:
753     ret
754
755 ;*************** Convert Hex to Decimal ******************
756 hex2dec:
757     ldi TempReg2, $00
758     ldi TempBottom6, $00
759     ldi TempReg2, -1 + '0'
760 _bib3:
761     inc     TempReg2
762         subi    TempReg, low(100)
763         sbci    TempBottom6, high(100)
764         brcc    _bib3
765
766         ldi     TempBottom6, 10 + '0'
767 _bib4:
768     dec     TempBottom6
769         subi    TempReg, -10
770         brcs    _bib4
771
772         subi    TempReg, -'0'
773     ret
774
```

```
;*************** Display 'Flappy Bird' ******************
FlappyOut:
    ldi TempReg, $10        ;
    mov NumPositiony, TempReg     ;
    ldi TempReg, $C9        ;
    mov NumPositionx, TempReg     ;
    rcall FOut              ; Display 'F'

    ldi TempReg, $B4        ;
    mov NumPositionx, TempReg     ;
    rcall LOut              ; Display 'L'

    ldi TempReg, $9F        ;
    mov NumPositionx, TempReg     ;
    rcall AOut              ; Display 'A'

    ldi TempReg, $8A        ;
    mov NumPositionx, TempReg     ;
    rcall POut              ; Display 'P'

    ldi TempReg, $75        ;
    mov NumPositionx, TempReg     ;
    rcall POut              ; Display 'P'

    ldi TempReg, $60        ;
    mov NumPositionx, TempReg     ;
    rcall YOut              ; Display 'Y'

    ldi TempReg, $40        ;
    mov NumPositiony, TempReg     ;
    ldi TempReg, $B4        ;
    mov NumPositionx, TempReg     ;
    rcall BOut              ; Display 'B'

    ldi TempReg, $9F        ;
    mov NumPositionx, TempReg     ;
    rcall BigIOut           ; Display 'I'

    ldi TempReg, $8A        ;
    mov NumPositionx, TempReg     ;
    rcall BigROut           ; Display 'R'

    ldi TempReg, $75        ;
    mov NumPositionx, TempReg     ;
    rcall DOut              ; Display 'D'

    ret                ;

;******************** Display Digit **********************
DigitOut:
    lsl TempReg
    lsl TempReg
    lsl TempReg
    lsl TempReg
    cpi TempReg, $00
    brne NotZero
    rcall ZeroOut
    ret
NotZero:
    cpi TempReg, $10
    brne NotOne
    rcall OneOut
    ret
NotOne:
    cpi TempReg, $20
    brne NotTwo
    rcall TwoOut
    ret
NotTwo:
    cpi TempReg, $30
```

```
    brne NotThree
    rcall ThreeOut
    ret
NotThree:
    cpi TempReg, $40
    brne NotFour
    rcall FourOut
    ret
NotFour:
    cpi TempReg, $50
    brne NotFive
    rcall FiveOut
    ret
NotFive:
    cpi TempReg, $60
    brne NotSix
    rcall SixOut
    ret
NotSix:
    cpi TempReg, $70
    brne NotSeven
    rcall SevenOut
    ret
NotSeven:
    cpi TempReg, $80
    brne NotEight
    rcall EightOut
    ret
NotEight:
    cpi TempReg, $90
    brne NotNine
    rcall NineOut
NotNine:
    ret

;******************** Display Tube **********************
TubeOut:
    cp TempRightX3, TempLeftY4
    brsh tubedone

    mov TempReg, TempRightX3

    out PORTD, TempTopLength5
    out PORTC, TempReg

    cbi PortB, 2          ; Enable Brightness
    rcall BigDel

Top:
    inc TempReg
    out PORTD, TempTopLength5
    out PORTC, TempReg

    cp TempReg, TempLeftY4
    brne Top

    mov TempReg, TempTopLength5

Left:
    inc TempReg
    out PORTD, TempReg
    cp TempReg, TempBottom6
    brne Left

    mov TempReg, TempLeftY4

Bottom:
    dec TempReg
    out PORTC, TempReg
    cp TempReg, TempRightX3
```

```
    brne Bottom

    mov TempReg, TempBottom6

Right:
    dec TempReg
    out PORTD, TempReg

    cp TempReg, TempTopLength5
    brne Right

    sbi PortB, 2          ; Disable Brightness
    rcall BigDel
    ret


;****************** Display Ground **********************
Ground:
    ldi TempRightX3, $50
    ldi TempLeftY4, $FF
    ldi TempTopLength5, $E0
    ldi TempBottom6, $FF
    rcall TubeOut

    ldi TempRightX3, $50
    ldi TempLeftY4, $FF
    ldi TempTopLength5, $00
    ldi TempBottom6, $E0
    rcall TubeOut

    ret

;******************** Display Bird **********************
Bird:
    ldi TempRightX3, $9F
    ldi TempLeftY4, $AF
    mov TempTopLength5, BirdPosition
    mov TempBottom6, TempTopLength5
    subi TempTopLength5, $0F
    rcall TubeOut
    ret

;**************** Display Horizontal Line *****************
HorizontalOut:
    mov TempReg2, TempRightX3
    add TempReg2, TempTopLength5
    mov TempReg, TempRightX3
    out PORTC, TempReg
    out PORTD, TempLeftY4
    cbi PortB, 2          ; Enable Brightness
    rcall BigDel


HorizontalAgain:
    inc TempReg
    out PORTC, TempReg
    cp TempReg, TempReg2
    brne HorizontalAgain
    sbi PortB, 2          ; Disable Brightness
    rcall BigDel
    ret

;**************** Display Vertical Line *****************
VerticalOut:
    mov TempReg2, TempLeftY4
    add TempReg2, TempTopLength5
    mov TempReg, TempLeftY4
    out PORTC, TempRightX3
    out PORTD, TempReg
    cbi PortB, 2          ; Enable Brightness
```

```
985     rcall BigDel
986
987
988 VerticalAgain:
989     inc TempReg
990     out PORTD, TempReg
991     cp  TempReg, TempReg2
992     brne VerticalAgain
993     sbi PortB, 2          ; Disable Brightness
994     rcall BigDel
995     ret
996
997 ;*************** Display Number 1 ***********************
998 OneOut:
999     mov TempRightX3, NumPositionx
1000    mov TempLeftY4, NumPositiony
1001    ldi TempReg, $20
1002    mov TempTopLength5, TempReg
1003    rcall VerticalOut
1004    ret
1005
1006 ;*************** Display Number 2 ***********************
1007 TwoOut:
1008    mov TempRightX3, NumPositionx
1009    mov TempLeftY4, NumPositiony
1010    ldi TempReg, $10
1011    mov TempTopLength5, TempReg
1012    rcall HorizontalOut
1013    ldi TempReg, $0B
1014    mov TempTopLength5, TempReg
1015    rcall VerticalOut
1016    ldi TempReg, $0D
1017    add TempLeftY4, TempReg
1018    ldi TempReg, $10
1019    mov TempTopLength5, TempReg
1020    rcall HorizontalOut
1021    ldi TempReg, $13
1022    add TempLeftY4, TempReg
1023    rcall HorizontalOut
1024    ldi TempReg, $13
1025    sub TempLeftY4, TempReg
1026    mov TempRightX3, NumPositionx
1027    ldi TempReg, $10
1028    add TempRightX3, TempReg
1029    rcall VerticalOut
1030    ret
1031
1032 ;*************** Display Number 3 ***********************
1033 ThreeOut:
1034    mov TempRightX3, NumPositionx
1035    mov TempLeftY4, NumPositiony
1036    ldi TempReg, $10
1037    mov TempTopLength5, TempReg
1038    rcall HorizontalOut
1039    ldi TempReg, $20
1040    mov TempTopLength5, TempReg
1041    rcall VerticalOut
1042    ldi TempReg, $0D
1043    add TempLeftY4, TempReg
1044    ldi TempReg, $10
1045    mov TempTopLength5, TempReg
1046    rcall HorizontalOut
1047    ldi TempReg, $13
1048    add TempLeftY4, TempReg
1049    rcall HorizontalOut
1050    ret
1051
1052 ;*************** Display Number 4 ***********************
1053 FourOut:
1054    mov TempRightX3, NumPositionx
```

```
1055      mov TempLeftY4, NumPositiony
1056      ldi TempReg, $20
1057      mov TempTopLength5, TempReg
1058      rcall VerticalOut
1059
1060      ldi TempReg, $0D
1061      add TempLeftY4, TempReg
1062      ldi TempReg, $10
1063      mov TempTopLength5, TempReg
1064      rcall HorizontalOut
1065
1066      ldi TempReg, $0D
1067      sub TempLeftY4, TempReg
1068      ldi TempReg, $10
1069      add TempRightX3, TempReg
1070      ldi TempReg, $0D
1071      mov TempTopLength5, TempReg
1072      rcall VerticalOut
1073      ret
1074
1075 ;*************** Display Number 5 **********************
1076 FiveOut:
1077      mov TempRightX3, NumPositionx
1078      mov TempLeftY4, NumPositiony
1079      ldi TempReg, $10
1080      mov TempTopLength5, TempReg
1081      rcall HorizontalOut
1082      ldi TempReg, $0D
1083      add TempLeftY4, TempReg
1084      rcall HorizontalOut
1085      ldi TempReg, $13
1086      add TempLeftY4, TempReg
1087      rcall HorizontalOut
1088      mov TempLeftY4, NumPositiony
1089      ldi TempReg, $10
1090      add TempRightX3, TempReg
1091      ldi TempReg, $0D
1092      mov TempTopLength5, TempReg
1093      rcall VerticalOut
1094      ldi TempReg, $0D
1095      add TempLeftY4, TempReg
1096      mov TempRightX3, NumPositionx
1097      ldi TempReg, $13
1098      mov TempTopLength5, TempReg
1099      rcall VerticalOut
1100      ret
1101
1102 ;*************** Display Number 6 **********************
1103 SixOut:
1104      mov TempRightX3, NumPositionx
1105      mov TempLeftY4, NumPositiony
1106      ldi TempReg, $10
1107      mov TempTopLength5, TempReg
1108      rcall HorizontalOut
1109      ldi TempReg, $0D
1110      add TempLeftY4, TempReg
1111      rcall HorizontalOut
1112      ldi TempReg, $13
1113      add TempLeftY4, TempReg
1114      rcall HorizontalOut
1115      mov TempLeftY4, NumPositiony
1116      ldi TempReg, $10
1117      add TempRightX3, TempReg
1118      ldi TempReg, $20
1119      mov TempTopLength5, TempReg
1120      rcall VerticalOut
1121      ldi TempReg, $0D
1122      add TempLeftY4, TempReg
1123      mov TempRightX3, NumPositionx
1124      ldi TempReg, $13
```

```
1125      mov TempTopLength5, TempReg
1126      rcall VerticalOut
1127      ret
1128
1129  ;*************** Display Number 7 **********************
1130  SevenOut:
1131      mov TempRightX3, NumPositionx
1132      mov TempLeftY4, NumPositiony
1133      ldi TempReg, $20
1134      mov TempTopLength5, TempReg
1135      rcall VerticalOut
1136      ldi TempReg, $10
1137      mov TempTopLength5, TempReg
1138      rcall HorizontalOut
1139      ret
1140
1141  ;*************** Display Number 8 **********************
1142  EightOut:
1143      mov TempRightX3, NumPositionx
1144      mov TempLeftY4, NumPositiony
1145      ldi TempReg, $10
1146      mov TempTopLength5, TempReg
1147      rcall HorizontalOut
1148      ldi TempReg, $0D
1149      add TempLeftY4, TempReg
1150      rcall HorizontalOut
1151      ldi TempReg, $13
1152      add TempLeftY4, TempReg
1153      rcall HorizontalOut
1154      mov TempLeftY4, NumPositiony
1155      ldi TempReg, $10
1156      add TempRightX3, TempReg
1157      ldi TempReg, $20
1158      mov TempTopLength5, TempReg
1159      rcall VerticalOut
1160      mov TempRightX3, NumPositionx
1161      ldi TempReg, $20
1162      mov TempTopLength5, TempReg
1163      rcall VerticalOut
1164      ret
1165
1166  ;*************** Display Number 9 **********************
1167  NineOut:
1168      mov TempRightX3, NumPositionx
1169      mov TempLeftY4, NumPositiony
1170      ldi TempReg, $20
1171      mov TempTopLength5, TempReg
1172      rcall VerticalOut
1173      ldi TempReg, $10
1174      mov TempTopLength5, TempReg
1175      rcall HorizontalOut
1176      ldi TempReg, $0D
1177      add TempLeftY4, TempReg
1178      ldi TempReg, $10
1179      mov TempTopLength5, TempReg
1180      rcall HorizontalOut
1181
1182      ldi TempReg, $0D
1183      sub TempLeftY4, TempReg
1184      ldi TempReg, $10
1185      add TempRightX3, TempReg
1186      ldi TempReg, $0D
1187      mov TempTopLength5, TempReg
1188      rcall VerticalOut
1189      ret
1190
1191  ;*************** Display Number 0 **********************
1192  ZeroOut:
1193      mov TempRightX3, NumPositionx
1194      mov TempLeftY4, NumPositiony
```

```
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall HorizontalOut
    ldi TempReg, $20
    add TempLeftY4, TempReg
    rcall HorizontalOut
    mov TempLeftY4, NumPositiony
    ldi TempReg, $10
    add TempRightX3, TempReg
    ldi TempReg, $20
    mov TempTopLength5, TempReg
    rcall VerticalOut
    mov TempRightX3, NumPositionx
    ldi TempReg, $20
    mov TempTopLength5, TempReg
    rcall VerticalOut
    ret

;************ Display Small Letter H **********************
HOut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut
    ldi TempReg, $08
    add TempRightX3, TempReg
    rcall VerticalOut
    ldi TempReg, $08
    add TempLeftY4, TempReg
    mov TempTopLength5, TempReg
    mov TempRightX3, NumPositionx
    rcall HorizontalOut
    ret

;************ Display Small Letter I **********************
IOut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $04
    add TempRightX3, TempReg
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut
    ret

;************ Display Small Letter G **********************
GOut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $08
    mov TempTopLength5, TempReg
    rcall HorizontalOut
    ldi TempReg, $10
    add TempLeftY4, TempReg
    rcall HorizontalOut

    mov TempLeftY4, NumPositiony
    ldi TempReg, $08
    add TempRightX3, TempReg
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut

    mov TempRightX3, NumPositionx
    ldi TempReg, $08
    mov TempTopLength5, TempReg
    mov TempLeftY4, NumPositiony
    add TempLeftY4, TempReg
    rcall VerticalOut
```

```
1265      ldi TempReg, $04
1266      mov TempTopLength5, TempReg
1267      rcall HorizontalOut
1268      ret
1269
1270  ;************ Display Small Letter S **********************
1271  SOut:
1272      mov TempRightX3, NumPositionx
1273      mov TempLeftY4, NumPositiony
1274      ldi TempReg, $08
1275      mov TempTopLength5, TempReg
1276      rcall HorizontalOut
1277      ldi TempReg, $08
1278      add TempLeftY4, TempReg
1279      rcall HorizontalOut
1280      ldi TempReg, $08
1281      add TempLeftY4, TempReg
1282      rcall HorizontalOut
1283      mov TempLeftY4, NumPositiony
1284      ldi TempReg, $08
1285      add TempRightX3, TempReg
1286      ldi TempReg, $08
1287      mov TempTopLength5, TempReg
1288      rcall VerticalOut
1289      ldi TempReg, $08
1290      add TempLeftY4, TempReg
1291      mov TempRightX3, NumPositionx
1292      ldi TempReg, $08
1293      mov TempTopLength5, TempReg
1294      rcall VerticalOut
1295      ret
1296
1297  ;************ Display Small Letter C **********************
1298  COut:
1299      mov TempRightX3, NumPositionx
1300      mov TempLeftY4, NumPositiony
1301      ldi TempReg, $08
1302      mov TempTopLength5, TempReg
1303      rcall HorizontalOut
1304      ldi TempReg, $10
1305      add TempLeftY4, TempReg
1306      rcall HorizontalOut
1307
1308      mov TempLeftY4, NumPositiony
1309      ldi TempReg, $08
1310      add TempRightX3, TempReg
1311      ldi TempReg, $10
1312      mov TempTopLength5, TempReg
1313      rcall VerticalOut
1314      ret
1315
1316  ;************ Display Small Letter O **********************
1317  OOut:
1318      mov TempRightX3, NumPositionx
1319      mov TempLeftY4, NumPositiony
1320      ldi TempReg, $08
1321      mov TempTopLength5, TempReg
1322      rcall HorizontalOut
1323      ldi TempReg, $10
1324      mov TempTopLength5, TempReg
1325      rcall VerticalOut
1326      ldi TempReg, $08
1327      mov TempTopLength5, TempReg
1328      ldi TempReg, $10
1329      add TempLeftY4, TempReg
1330      rcall HorizontalOut
1331
1332      mov TempLeftY4, NumPositiony
1333      ldi TempReg, $08
1334      add TempRightX3, TempReg
```

```
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut
    ret

;************ Display Small Letter R **********************
ROut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $03
    add TempRightX3, TempReg
    ldi TempReg, $05
    mov TempTopLength5, TempReg
    rcall HorizontalOut
    ldi TempReg, $08
    mov TempTopLength5, TempReg
    rcall VerticalOut

    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $08
    add TempLeftY4, TempReg
    rcall VerticalOut
    rcall HorizontalOut

    ldi TempReg, $08
    add TempRightX3, TempReg
    sub TempLeftY4, TempReg
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut
    ret

;************ Display Small Letter E **********************
EOut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $08
    mov TempTopLength5, TempReg
    rcall HorizontalOut

    ldi TempReg, $08
    add TempLeftY4, TempReg
    ldi TempReg, $08
    mov TempTopLength5, TempReg
    rcall HorizontalOut
    ldi TempReg, $08
    add TempLeftY4, TempReg
    rcall HorizontalOut

    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $08
    add TempRightX3, TempReg
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall VerticalOut

    ret

;************ Display Large Letter F **********************
FOut:
    mov TempRightX3, NumPositionx
    mov TempLeftY4, NumPositiony
    ldi TempReg, $10
    mov TempTopLength5, TempReg
    rcall HorizontalOut


    ldi TempReg, $10
```

```
1405      add TempLeftY4, TempReg
1406      ldi TempReg, $10
1407      mov TempTopLength5, TempReg
1408      rcall HorizontalOut
1409
1410
1411      mov TempRightX3, NumPositionx
1412      mov TempLeftY4, NumPositiony
1413      ldi TempReg, $10
1414      add TempRightX3, TempReg
1415      ldi TempReg, $20
1416      mov TempTopLength5, TempReg
1417      rcall VerticalOut
1418      ret
1419
1420 ;************ Display Large Letter L **********************
1421 LOut:
1422      mov TempRightX3, NumPositionx
1423      mov TempLeftY4, NumPositiony
1424      ldi TempReg, $10
1425      mov TempTopLength5, TempReg
1426      ldi TempReg, $20
1427      add TempLeftY4, TempReg
1428      ldi TempReg, $10
1429      mov TempTopLength5, TempReg
1430      rcall HorizontalOut
1431
1432      mov TempRightX3, NumPositionx
1433      mov TempLeftY4, NumPositiony
1434      ldi TempReg, $10
1435      add TempRightX3, TempReg
1436      ldi TempReg, $20
1437      mov TempTopLength5, TempReg
1438      rcall VerticalOut
1439      ret
1440
1441 ;************ Display Large Letter A **********************
1442 AOut:
1443      mov TempRightX3, NumPositionx
1444      mov TempLeftY4, NumPositiony
1445      ldi TempReg, $10
1446      mov TempTopLength5, TempReg
1447      rcall HorizontalOut
1448      ldi TempReg, $10
1449      add TempLeftY4, TempReg
1450      rcall HorizontalOut
1451      mov TempLeftY4, NumPositiony
1452      ldi TempReg, $10
1453      add TempRightX3, TempReg
1454      ldi TempReg, $20
1455      mov TempTopLength5, TempReg
1456      rcall VerticalOut
1457      mov TempRightX3, NumPositionx
1458      ldi TempReg, $20
1459      mov TempTopLength5, TempReg
1460      rcall VerticalOut
1461      ret
1462
1463 ;************ Display Large Letter P **********************
1464 POut:
1465      mov TempRightX3, NumPositionx
1466      mov TempLeftY4, NumPositiony
1467      ldi TempReg, $10
1468      mov TempTopLength5, TempReg
1469      rcall HorizontalOut
1470      rcall VerticalOut
1471      ldi TempReg, $10
1472      add TempLeftY4, TempReg
1473      rcall HorizontalOut
1474      mov TempLeftY4, NumPositiony
```

```
1475      ldi TempReg, $10
1476      add TempRightX3, TempReg
1477      ldi TempReg, $20
1478      mov TempTopLength5, TempReg
1479      rcall VerticalOut
1480      ret
1481
1482  ;************ Display Large Letter Y **********************
1483  YOut:
1484      mov TempRightX3, NumPositionx
1485      mov TempLeftY4, NumPositiony
1486      ldi TempReg, $10
1487      mov TempTopLength5, TempReg
1488      rcall VerticalOut
1489      ldi TempReg, $10
1490      add TempLeftY4, TempReg
1491      rcall HorizontalOut
1492      mov TempLeftY4, NumPositiony
1493      ldi TempReg, $10
1494      add TempRightX3, TempReg
1495      rcall VerticalOut
1496
1497      ldi TempReg, $08
1498      sub TempRightX3, TempReg
1499      ldi TempReg, $10
1500      add TempLeftY4, TempReg
1501      rcall VerticalOut
1502      ret
1503
1504  ;************ Display Large Letter B **********************
1505  BOut:
1506      mov TempRightX3, NumPositionx
1507      mov TempLeftY4, NumPositiony
1508      ldi TempReg, $10
1509      mov TempTopLength5, TempReg
1510      rcall HorizontalOut
1511      ldi TempReg, $10
1512      add TempLeftY4, TempReg
1513      rcall HorizontalOut
1514      ldi TempReg, $10
1515      add TempLeftY4, TempReg
1516      rcall HorizontalOut
1517      mov TempLeftY4, NumPositiony
1518      ldi TempReg, $10
1519      add TempRightX3, TempReg
1520      ldi TempReg, $20
1521      mov TempTopLength5, TempReg
1522      rcall VerticalOut
1523      mov TempRightX3, NumPositionx
1524      ldi TempReg, $20
1525      mov TempTopLength5, TempReg
1526      rcall VerticalOut
1527      ret
1528
1529  ;************ Display Large Letter I **********************
1530  BigIOut:
1531      mov TempRightX3, NumPositionx
1532      mov TempLeftY4, NumPositiony
1533      ldi TempReg, $08
1534      add TempRightX3, TempReg
1535      ldi TempReg, $20
1536      mov TempTopLength5, TempReg
1537      rcall VerticalOut
1538      ret
1539
1540  ;************ Display Large Letter R **********************
1541  BigROut:
1542      mov TempRightX3, NumPositionx
1543      mov TempLeftY4, NumPositiony
1544      ldi TempReg, $06
```

```
1545      add TempRightX3, TempReg
1546      ldi TempReg, $0A
1547      mov TempTopLength5, TempReg
1548      rcall HorizontalOut
1549      ldi TempReg, $10
1550      mov TempTopLength5, TempReg
1551      rcall VerticalOut
1552
1553      mov TempRightX3, NumPositionx
1554      mov TempLeftY4, NumPositiony
1555      ldi TempReg, $10
1556      add TempLeftY4, TempReg
1557      rcall VerticalOut
1558      rcall HorizontalOut
1559
1560      ldi TempReg, $10
1561      add TempRightX3, TempReg
1562      sub TempLeftY4, TempReg
1563      ldi TempReg, $20
1564      mov TempTopLength5, TempReg
1565      rcall VerticalOut
1566      ret
1567
1568 ;************ Display Large Letter D **********************
1569 DOut:
1570      mov TempRightX3, NumPositionx
1571      mov TempLeftY4, NumPositiony
1572      ldi TempReg, $10
1573      mov TempTopLength5, TempReg
1574      rcall HorizontalOut
1575      ldi TempReg, $20
1576      mov TempTopLength5, TempReg
1577      rcall VerticalOut
1578      ldi TempReg, $10
1579      mov TempTopLength5, TempReg
1580      ldi TempReg, $20
1581      add TempLeftY4, TempReg
1582      rcall HorizontalOut
1583
1584      mov TempLeftY4, NumPositiony
1585      ldi TempReg, $10
1586      add TempRightX3, TempReg
1587      ldi TempReg, $20
1588      mov TempTopLength5, TempReg
1589      rcall VerticalOut
1590      ret
1591
1592
1593
1594 ;*******************************************************
1595 ;***************                        ****************
1596 ;***************     EEPROM COMMANDS    ****************
1597 ;***************                        ****************
1598 ;*******************************************************
1599
1600 ;******************* EEPROM Write **********************
1601 EEPROM_write:
1602
1603      sbic EECR,EEWE          ; Wait for completion of previous write
1604      rjmp EEPROM_write         ;
1605      out EEARH, TempRightX3      ; Set up address (TempRightX3:TempReg2) in address
         register
1606      out EEARL, TempReg2       ;
1607      out EEDR,TempReg          ; Write data (TempReg) to data register
1608      sbi EECR,EEMWE            ; Write logical one to EEMWE
1609      sbi EECR,EEWE             ; Start eeprom write by setting EEWE
1610      ret                ;
1611
1612 ;******************* EEPROM Read ***********************
1613 EEPROM_read:
```

```
1614      sbic EECR,EEWE            ; Wait for completion of previous write
1615      rjmp EEPROM_read          ;
1616      out EEARH, TempRightX3       ; Set up address (TempRightX3:TempReg2) in address
          register
1617      out EEARL, TempReg2        ;
1618      sbi EECR,EERE             ; Start eeprom read by writing EERE
1619      in TempReg,EEDR           ; Read data from data register
1620      ret                 ;
1621
1622
1623
1624 ;*****************************************************
1625 ;***************                              ******************
1626 ;***************           DELAY              ******************
1627 ;***************                              ******************
1628 ;*****************************************************
1629
1630 BigDEL:
1631          ldi ZH, HIGH(10)
1632          ldi ZL, LOW (10)
1633 CountBigDel:
1634          sbiw ZL, 1
1635          brne CountBigDel
1636          ret
```