3D Pong on the ATmega128 Microprocessor

Alvise Vianello
CID: [redacted]

***Abstract*—The aim of this project was to build a vector-based Pong-like arcade game on an ATMEGA128. Playing in 3D, the player moves his paddle in the plane of the screen while the ball is hit in and out of it. Compatibility with an I2C Nintendo Nunchuk controller was considered but the idea had to be abandoned in the development stage due to technical difficulties and strict time constraints. User input is acquired through a keypad instead. By the end of the project a fully functional game was successfully created offering a single player vs. artificial intelligence (AI) mode, score and hit detection systems, and multiple difficulty levels.**

I. Introduction

DESIGNED by Allan Alcorn in the early 1970s, Pong is one of the most iconic arcade video-games ever created. Shortly after being launched by Atari in 1972, it became one of the most popular and widespread arcade video-games of all time. Pong marked the start of a video-game revolution and laid the foundations for today's multi-billion dollar video-game industry. [1]

Inspired by the history of Pong, we decided to develop a similar game for this project. Designing and building a video-game is challenging, but is also an excellent way to to test the understanding of micro-controllers and AVR assembly language acquired in the first part of the course.

The game is based on the same mechanics as the classic Pong game, but is played in three dimensions. The player's paddle and the AI controlled paddle lay in two distinct planes parallel to the screen. The ball moves in and out of the screen from one paddle to the other. The player can freely move his paddle by pressing the corresponding direction on a keypad and the aim of the game is to be able to hit back the ball towards the enemy paddle for as long as possible. The ball can be imparted momentum by moving the paddle along a certain direction while the ball is hitting it. The ball will then bounce back in that same direction. If the player misses the ball the game ends instantly. It is also possible to score points by successfully hitting the ball: at 10 and 20 points, the ball increases in speed. The AI is programmed to never lose and will hit the ball back towards the player in a random direction.

II. High Level Design

The hardware required for this project is relatively simple and straightforward. It involves very few components, is easily replicable, and offers room for additional components to extend the project.

Central to the game system is the ATMEGA128 processor [2]. Running at 8 MHz it allows to capture the user input control data with very low latency, and can run the game software very smoothly. Once processed, the data outputted by the microcontroller is displayed on an oscilloscope after being converted to analog signals by two Digital to Analog Converters (DACs).

The two 8-bit multiplying digital-to-analog converters (TLC7524 [3]) play an essential role as they allow to convert digital data into analog data. More specifically, they allow to convert the data relative to the position of the game objects (ball, paddles, ...) into an analog signal which can then be displayed on the oscilloscope . Both DACs were configured in exactly the same way. One was dedicated to the conversion of the signal for the X position of the objects and the other was used to convert the signal for the Y position of the objects. The DACs were fed with an approximate fixed time interval sample of an analog signal corresponding to the position of the object that had to be drawn on the screen. The data for the ball, for example, consisted of uniformly sampled sine and cosine values to which offsets were applied to shift the values to their appropriate in-game position. When objects had a shape more difficult to replicate as a combination of simple functions (e.g. text), they were drawn by sending values to the DACs corresponding to the 'pixel' position of each of their points.

The game software is built around a single MAIN LOOP which manages all operations to be performed during a single *game cycle*. A *game cycle* is defined as starting when the player hits the ball, and ends when the ball gets back to the player after being hit by the opponent. Ancillary short loops managing the Start and Game Over screens are also present, but the core of the game resides in the MAIN LOOP.

Some optional features such as the system to change difficulty level were also implemented in the software. The transition to a higher difficulty level is done automatically after the player has hit the ball for a certain amount of times in a row without missing it. The ball then increases in velocity, making it more difficult to hit it back in time.

III. Hardware Design

Digital To Analog Converters:

The DACs interface to the microprocessor through two 8bit data-buses and the $\overline{\text{CS}}$ and $\overline{\text{WR}}$ control signals. Both control signals were constantly set to low to ensure that the input latches were transparent and that the analog output of the DACs matched directly the activity on the data bus inputs. (Fig. 1)

The inner working of a DAC relies on a series of switches controlling a resistor network. Operated in voltage mode, this so called *ladder network* (R-2R) allows to weigh the input bits in their contribution to the output voltage. By giving a reference voltage value (V_{ref}) to the DAC, a logic 0 bit input will correspond to a voltage $V = 0$, while a logic 1 bit input will correspond to $V = V_{ref}$. The output voltage of the ladder network will then have a stepped value between 0 and V_{ref} depending on which bits are set at the DAC input.

For this project, the DACs were set up in a bipolar mode (4-Quadrant Operation) in order to give a stepped output voltage in the range $-V_{ref}, +\frac{127}{128}V_{ref}$. This allowed to have negative voltage values and a better mapping of the input data to output voltages for the oscilloscope. (Table I)

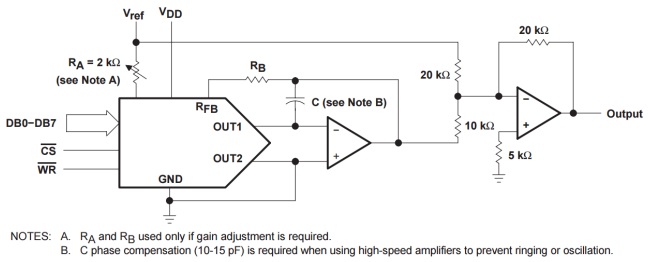

Fig. 1. Circuit diagram for a DAC in Bipolar Mode (4-Quadrant Operation). $\overline{CS}$ and $\overline{WR}$ control signals were held low to reflect the activity at the input pins DB0-DB7 [3]

TABLE I
BIPOLAR (OFFSET BINARY) CODE

DIGITAL INPUT	ANALOG OUTPUT
11111111	$V_{ref}(127/128)$
10000001	$V_{ref}(1/128)$
10000000	0
01111111	$-V_{ref}(1/128)$
00000000	$-V_{ref}$

Oscilloscope:

The analog signal outputted by the DACs was displayed on an oscilloscope set in X-Y mode. X-Y mode plots a varying voltage at the oscilloscope's Y input against another at its X input. This mode allows to use the oscilloscope as a vector display and also allows to track phase differences between the two input signals, producing *Lissajous* curves. The ball was drawn by sending sine and cosine waves into the oscilloscope. Their relative phase offset of $\pi/2$ produces a characteristic Lissajous curve, a circle. Objects like the paddles or the menu texts, were drawn through vector graphics. Sending sampled 'pixel' coordinates of the objects to the oscilloscope forces the beam to skip all points in between and only draw the selected ones.

Keypad:

Player input is acquired through a 4x4 keypad interfacing to the microprocessor through a 8-bit data bus. Only a 3x3 subsection of the keypad is used in the game but extra buttons could be easily configured to trigger additional functionalities. Buttons 1 to 9 move the paddle in a specific direction. For example, button 1 will move the ball diagonally towards the upper left corner while button 8 will move the ball downwards. The only button not mapped to a direction is button 5. The keypad is an extremely simple piece of hardware and is completely passive. Pressing a button simply connects a row to a column.

IV. SOFTWARE DESIGN

The game software architecture is based on a MAIN LOOP which manages the game in its *idle* state. It also inglobes the GAME LOOP which manages every other aspect of the game once the player decides to start playing against the AI.

The MAIN LOOP (Fig 2) is divided in three sub-loops, one of which is the GAME LOOP itself.

The fist sub-loop (*Start Screen*) manages the game's start menu. A text is displayed on the oscilloscope prompting the player to press a button to start playing. The ball and paddles are also displayed at their starting positions. On every loop, the software checks if a button has been pressed on the keypad. As soon as the condition is met, the software exits this sub-loop and jumps to the next one, the GAME LOOP. The GAME LOOP is the second of the three sub-loops managed by the MAIN LOOP and is explained in more details later on.

The third and last sub-loop of the MAIN LOOP is the GAME OVER LOOP, which is triggered when the player loses the game by missing the ball in the GAME LOOP. In the GAME OVER LOOP, the player is shown his score and a freezed frame with the position of the ball and paddles when he lost. The two frames alternate and are displayed three times each before the loop exit automatically to the *Start Screen* Loop. In this way the player has time to see how he lost and his score before having the opportunity to play again.

The GAME LOOP, which is triggered as soon as the player presses a button on the keypad, is at the core of the game software. Every iteration of this loop manages one *game cycle*. A *game cycle* is defined as the ensemble of events taking place from when the player hits the ball to when the ball gets back to the player after being hit by the AI. Every loop of the GAME LOOP will correspond to a 'hit' of the ball (Fig. 3).

The GAME LOOP performs the following operations until the game is lost:

1) The buttons are checked on the keypad to determine in which direction the ball is being hit.

2) The hit function is called
3) The final position of the ball is compared to the position of the player's paddle. If the centre of the ball is outside of the paddle the game is lost.

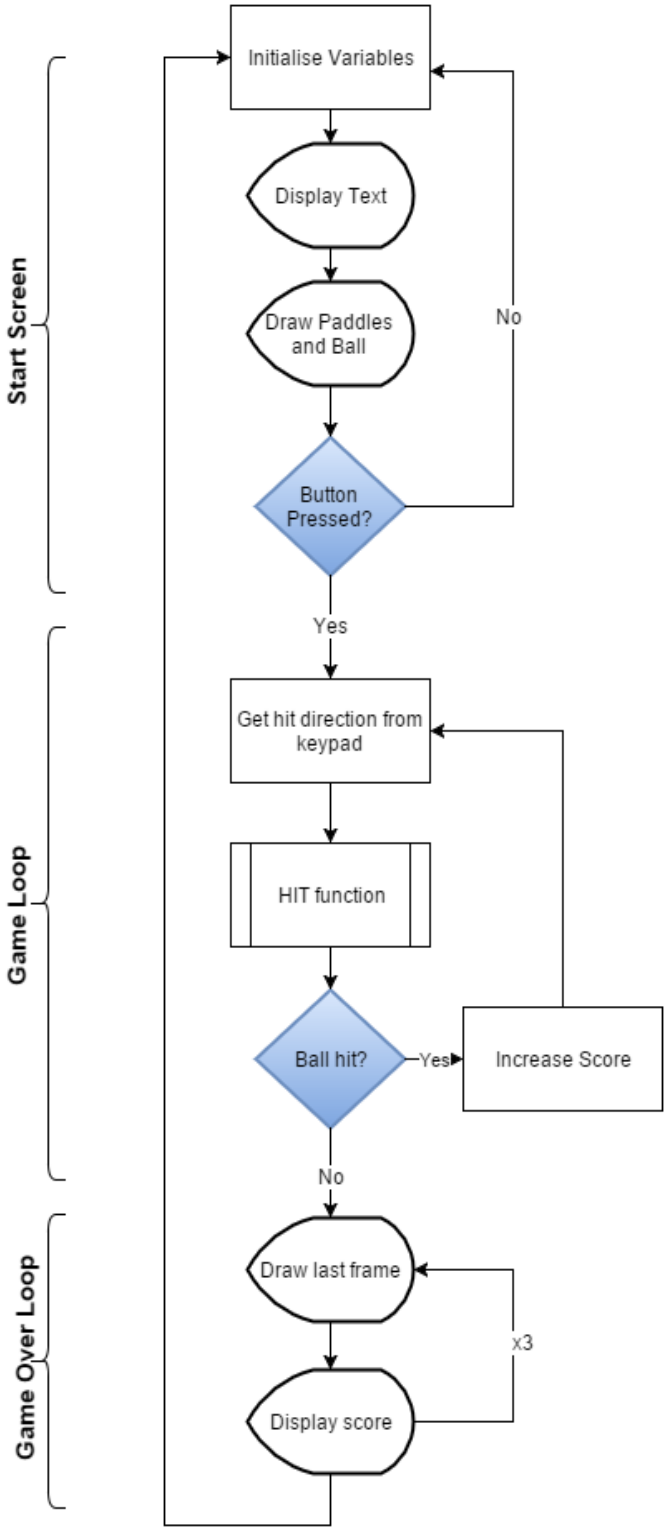

Fig. 2. Flowchart of the MAIN LOOP. The MAIN LOOP orchestrates the flow of the entire game. It is composed of three sub-loops: the START SCREEN, the GAME LOOP, and the GAME OVER LOOP. At the heart of the GAME LOOP is the HIT function, the most complex routine of this project.

The HIT function is a loop that manages all movements of the ball and paddles given an initial direction in which the ball is hit. The ball decreases in size to simulate its movement into the screen. It is then hit back by the AI in a random direction, and gradually returns to the player's paddle at its initial size. The player can also move the paddle independently while the ball is moving in and out of the screen.

For every size of the ball, that is from when the ball leaves the player paddle to when it comes back, the hit function will execute the following:

1) Decrease the size of the ball
2) Update the position of the ball based on the direction it was originally hit by the player. Checks are performed to see if part of the ball is outside of the screen boundaries.
3) Update the position of the AI. Checks are performed to see if part of the AI paddle is outside of the screen boundaries. If it is, the paddle is shifted back just enough to be inside of the boundaries.
4) Change the speed of the ball depending on the player's score. The ball will be drawn less times on the screen giving an illusion of greater speed.
5) The ball and paddles are appropriately drawn on the screen for the amount of time set by the difficulty level. If part of the ball is outside of the screen it is drawn squashed against the screen edge. In the meantime the keypad buttons are constantly checked for input giving the player full independent control of the paddle movement.
6) If the ball has reached the AI paddle, it is hit back in a random direction. If not, the ball is still traveling from one paddle to the other and the software loops back from 1)
7) Increase size of the ball. The ball is now traveling back towards the player.
8) Steps 2) to 5) are repeated.
9) If the ball has reached the player the hit function returns. If not, the software loops back from 7)

V. RESULTS AND PERFORMANCE

The goals of this project were particularly challenging as they involved a synchronous development of both the game and the interface with the NUNCHUK controller. Unfortunately the latter had to be interrupted due to time constraints but a lot of investment had been put into it. The code developed for it has been included nonetheless at the end of the APPENDIX to show the progress made and for future reference should this project be continued.

Even by switching to a keypad input, the design and development of the game were non-trivial. Far from being a simple Pong clone, the game adds a completely new

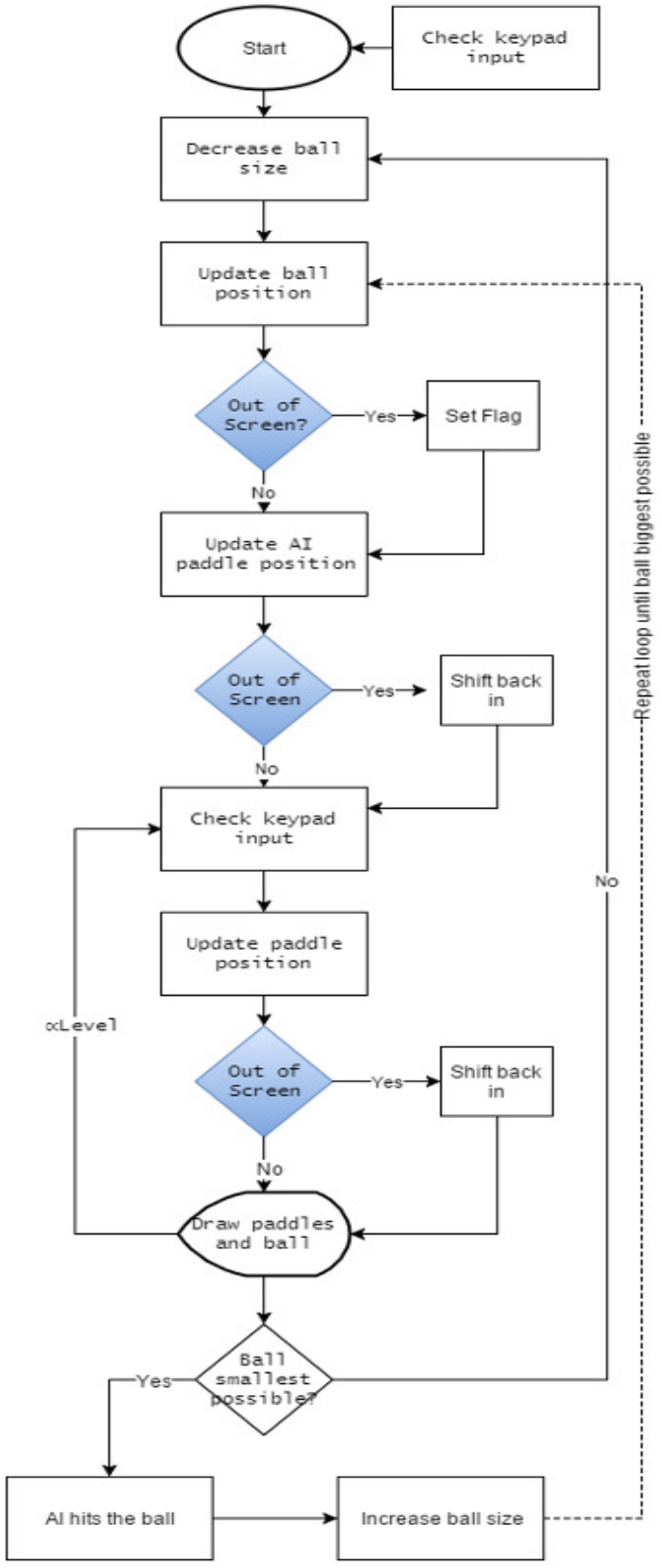

Fig. 3. Flowchart of the HIT function (compact representation). The function consists of two symmetric loop, one iterated through when the ball is going into the screen (decreasing in size), and one iterated through when the ball is hit back towards the player (ball increasing in size). The function returns once the ball has reached the player's paddle.

dimension to the gameplay and was much more complex than expected. The end results fulfills all of the goals that were set for the project while also implementing some extra functionalities such as the scoreboard and difficulty level systems. Some other functionalities (e.g sound) could not see the light because of time constraints but should be relatively easy to add in the future. Overall, the game produced is very performant and exceeded our expectations.

During the development phase, most difficulties were encountered either in the development of the NUNCHUK interface to the microprocessor or in the coding of routines managing the position of the objects on the oscilloscope.

The integration of the NUNCHUK was particularly challenging and time consuming. Despite it being an I2C compatible peripheral we were unable achieve a correct transfer of data from the NUNCHUK to the microcontroller. Initial progress was made by adapting the Two Wire Serial Interface TWI Master Transmitter code present in the ATMEGA128 manual. Further research uncovered extra quirks in the I2C protocol used by Nintendo peripherals (e.g. need for special *handshakes* between Master and Slave, encryption of data sent to the Master, ...). Attempts were made to address all those particularities but to no avail.

The correct display of the objects on the oscilloscope was also an aspect of the game where most time was invested.

One of the very first problems encountered was in the setup of the DAC circuits. Initially set in Unipolar Operation (2-Quadrant Multiplication) mode, negative output voltages could not be achieved forcing us to compensate by developing convoluted routines pre-processing the data fed to the DACs. This resulted in an extremely non intuitive mapping between data values and corresponding representation on the oscilloscope. The discovery of the Bipolar Operation Mode solved the issue although an initial lack of $20\,k\Omega$ resistors ($22\,k\Omega$ were used instead) generated asymmetric output voltages whose origin was difficult to trace back.

Linked to the setup of the DACs in Bipolar Mode, problems were also encountered when devising routines to manage objects outside of the screen boundaries. Bipolar Mode makes use of input data in Offset Binary format, ranging from $0b00000000$ for an output voltage of -$\mathrm{V_{ref}}$, to $0b11111111$ for an output voltage of $\frac{127}{128}\mathrm{V_{ref}}$ (Table I). Most Variables used in the body of the program were instead in 2s complement format. This is because the variables represented the position of the objects relative to the centre of the oscilloscope. A value of 0 would produce 0 offset, while negative or positive values would produce offset in the negative and positive directions respectively.Because of this particularity, routines involving operations between Offset Binary formatted data and 2s complement formatted values were a major source of confusion. The two formats differ for example in the value of the MSB for negative or positive voltages/offsets (0 for a negative voltage, 1 for a negative offset, and vice versa).

VI. Updates, Modifications and Improvements

This project offers plenty of opportunities for updates and improvements, in particular in relation to the problems encountered in the development phase that could not be addressed due to time constraints.

Good progress towards the Nunchuk integration could be made by starting to set-up a suitable debugging system. In this project, debugging was limited to the output of the Two Wire Interface Status Register (TWSR) onto the LEDs of the development board [4]. Output of data on an LCD screen was prohibited by the need to run the Nunchuk (and thus the board) at $3.3\,\mathrm{V}$, which was not enough to power the screen. An optimal solution would be to stream the data received by the Two Wire Serial Interface onto a computer through the board's RS232 port. This is relatively easily to implement (free COM port communication software is readily available online [5]) and should allow for a much thorough understanding of how the NUNCHUK and the microprocessor are interfacing.

The game could also be improved by modifying the behaviour of the AI. Instead of having an omniscient AI which always hits the ball, it could be possible to add a degree of randomness in its probability to hit it the ball back. The player could then be able to select the AI skill level before starting the game, for example by pressing a specific keypad button while in the START MENU screen. The direction in which the AI hits the ball back could also be modified in order to take into account its relative position on the screen and/or the position of the player. It is worth noting that development of the AI should be preferred over the addition of a two player variant as the geometric perspective of the game graphics would force on of the two players to play with a much smaller paddle.

Although fully functional, the game created in this project lends itself to a myriad of further improvements and spin-offs. Only to mention a few, the following would all be valid ideas: a permanent leader board (saved between games), a lives system, sound, obstacles between the player and the AI paddles, mini game based on the 3D Pong mechanic, etc ...

VII. Product Specifications

- 4x4 Keypad Interfacing
- Support for Oscilloscope X-Y Mode Display
- Start Menu Screen
- Single Player vs. AI mode
- Hit detection
- Momentum based directional hits
- AI automatic paddle movement
- Random AI directional hit
- Continuous Independent Paddle control
- Screen boundary Detection Routines
- Wall Interactions Support (Bouncing, Squashing)
- Score System up to 99 points
- Automatic Difficulty Level Increase
- Game Over screen
- Last game frame 'replay' support
- Score Display screen
- Automatic return to Start Menu on game loss
- No input/output latency

VIII. Conclusion

The aim of this project was to build a vector-based Pong-like arcade game on an ATMEGA128. A fully fledged three dimensional variant of the game was successfully produced, satisfying all the main goals that had been set for the project. In particular, interfacing with a keyboard controller, single player vs. AI mode, and vector-based graphic display on an oscilloscope have all been supported. Some but not all of the optional goals have also been achieved. For example, a scoring system and a difficulty level system have both been added to the core functionalities of the game. A few optional goals were not reached due to technical difficulties or a lack of time. For each specific feature, solutions or ideas on how to implement them have been discussed in the report.

Relying on readily available hardware and based upon a very simple circuit design, this project can be replicated very easily. Together with the code and the circuit schematic this project represents a very interesting resource and starting block for any AVR microcontroller enthusiast who would then be able to invest his time in the implementation of new ideas and features.

References

[1] Matt Barton, and Bill Loguidice. The History Of Pong: Avoid Missing Game to Start Industry. From *Vintage Games: An Insider Look at the History of Grand Theft Auto, Super Mario, and the Most Influential Games of All Time*, Jan. 2009.

[2] Atmel. ATmega128/ATmega128L User Manual. *"8-bit Atmel Microcontroller with 128KBytes In-System Programmable Flash"*, Rev. 2467X-AVR-06/11

[3] Texas Instruments. TLC7524C, TLC7524E, TLC7524I, 8-bit Multiplying Digital-To-Analog Converters, *Datasheet*. Sept 1986, Rev. Jun 2007.

[4] Atmel. megaAVR Starter Kit User Guide. Feb 1999.

[5] Bray Terminal - Terminal emulation program for RS-232. http://hw-server.com/terminal-terminal-emulation-program-rs-232

Texas Instruments. TL07XX Low-Noise JFET-Input Operational Amplifiers, em Datasheet. Sept 1978, Rev. Jun 201.

NXP Semiconductors. UM10204 I2C-bus specification and user manual. Rev. 6 4 Apr 2014. http://www.nxp.com/documents/user_manual/UM10204.pdf

Geoffrey Brown. Discovering the STM32 Microcontroller, em Chapter 9 - I2C Nunchuk. Rev 0743cf9 (2015-11-11) http://www.cs.indiana.edu/~geobrown/book.pdf

Inex Innovative Experiment. ZX Nunchuk Datasheet. http://www.robotshop.com/media/files/PDF/inex-zx-nunchuck-datasheet.pdf

Todbot Blog. Whiichuk adaptor and Arduino/Nunchuk interfacing resources. https://todbot.com/blog/2008/02/18/wiichuck-wii-nunchuck-adapter-available/

Jed Margolin. The Secret Life of Vector Generators. *Companion piece to The Secret Life of XY Monitors*. Apr 22, 2001. Rev. Jul 21, 2001; Jul 20, 2003. http://www.jmargolin.com/vgens/vgens.htm

APPENDIX

The following pages contain the entire AVR assembly code necessary to run the game. The very last part, commented out, contains all the routines developed for the I2C interface with the NUNCHUK.

```
;  **  ATmega128(L) Assembly Language File - IAR Assembler Syntax **

.DEVICE ATmega128
.ORG    0
.include "m128def.inc"          ; Add required path to IAR Directory


; =====================================================================
;
DEFINITIONS
; =====================================================================

.def    TEMPREG =  r16      ; Temporary Register
.def    RANDOM_N = r17      ; random number
.def    XPOS = r18          ; ball x position on screen 0b1 = 2.5/128 V = 4/128 of division
on scope
.def    YPOS = r19          ; ball y position on screen
.def    SCALING = r20       ; radius of the ball (R=R0/2**scaling)
.def    COUNTER = r21       ; multi purpose counter
.def    DRAW_TIMES = r22    ; amount of times object is redrawn on screen
.def    FLAG = r23          ; flag: ball_out_scr_x ¦ ball_out_scr_y ¦ enemy_out_scr_x ¦
enemy_out_scr_y ¦ 3 ¦ draw_score_10s ¦ ball_not_on_paddle ¦ button_pressed
.def    DELTAX = r24        ; change in ball x position per frame
.def    DELTAY = r25        ; change in ball y position per frame
.def    PADDLE_XPOS = r4    ; paddle x position on screen
.def    PADDLE_YPOS = r5    ; paddle y position on screen
.def    PADDLE_DELTAX = r6  ; change in paddle x position per frame
.def    PADDLE_DELTAY = r7  ; change in paddle y position per frame
.def    TEMPREG2 = r8       ; secondary temporary register
.def    ENEMY_HIT_COUNTER = r9  ; number of times enemy has hit the ball since start of
whole game
.def    SCORE = r10         ; player score

; =====================================================================
;
INITIALISATION
; =====================================================================

rjmp Init                       ; Jump to initialisation routine

Init:
    ; ************* Stack Pointer Set-up Code *************
    ldi r16, $0F            ; Stack Pointer Set-up
    out SPH,r16             ; Stack Pointer High Byte
    ldi r16, $FF            ; Stack Pointer Set-up
    out SPL,r16             ; Stack Pointer Low Byte

    ; ************* RAMPZ Set-up Code *************
    ldi  r16, $00           ; 1 = EPLM acts on upper 64K
    out RAMPZ, r16          ; 0 = EPLM acts on lower 64K

    ; ************* Sleep Mode And SRAM  *************
    ldi r16, $C0            ; Idle Mode - SE bit in MCUCR not set
    out MCUCR, r16          ; External SRAM Enable Wait State Enabled

    ; ************* Comparator Set-up Code *************
    ldi r16,$80             ; Comparator Disabled, Input Capture Disabled
    out ACSR, r16           ; Comparator Settings

    ; ************* Port D Set-up Code *************
    ldi r16, $00        ; Input
    out DDRD, r16       ; Port D Direction Register
    ldi r16, $FF        ; Initialise value
    out PORTD, r16      ; Port D value

    ; ************* Port B Set-up Code *************
    ldi r16, $FF            ; Output
    out DDRB , r16          ; Port B Direction Register
    ldi r16, $FF            ; Initialise value
    out PORTB, r16          ; Port B value
    ; ************* Port E Set-up Code *************
```

```
    ldi r16, $FF            ; Output
    out DDRE, r16           ; Port E Direction Register
    ldi r16, $FF            ; Initialise value
    out PORTE, r16          ; Port E value

; ============================================================================
;
MAIN
; ============================================================================

ldi ZL, low(2*enemy_random_hits)    ; load pointer to predetermined directions for enemy hits
ldi ZH, high(2*enemy_random_hits)   ; ... pointer has to be conserved throughout the game
ldi TEMPREG, 0
mov ENEMY_HIT_COUNTER, TEMPREG      ; initialise ENEMY_HIT_COUNTER to 0

Start_Screen:

    ldi TEMPREG, 0
    mov SCORE, TEMPREG          ; initialise score to 0
    mov PADDLE_XPOS, TEMPREG    ; initialise paddle ...
    mov PADDLE_YPOS, TEMPREG    ; ... on centre of screen
    ldi XPOS, 0                 ; initialise ball ...
    ldi YPOS, 0                 ; ... on centre of screen
    ldi SCALING, 1              ; initialise size of the ball to R0/2
    ldi FLAG, 0                 ; initialise flag to 0
    ldi COUNTER, 0

    ldi DRAW_TIMES, 5
    rcall draw_startMenu        ; draw start menu text on screen

    ldi DRAW_TIMES, 5
    rcall draw_ball             ; draw ball on screen

    ldi DRAW_TIMES, 5
    rcall draw_paddle           ; draw paddle on screen

    clr FLAG
    rcall Check_Buttons_Hit     ; check status of buttons
    sbrc FLAG, 0                ; if a button has been pressed (flag bit0 set)...
    rjmp Game_loop              ; ... start the game
    rjmp Start_Screen           ; ... else keep waiting

Game_loop:                      ; Main Game, every loop manages one hit cycle

    cbr FLAG, 0b00000001        ; clear button_hit flag
    rcall Check_Buttons_Hit     ; update ball movement values depending on button pressed
    cbr FLAG, 0b00000001        ; clear flag (not using it)

    rcall hit                   ; animate one game cycle (until ball goes back to paddle)

    cbr FLAG, 0b00000010        ; clear ball_on_paddle flag
    rcall check_ball_on_paddle  ; check if ball came back on paddle
    sbrc FLAG, 1                ; check ball_on_paddle flag ...
    rjmp Game_Over              ; ... ball missed ... Game Over!
    rjmp Game_loop              ; ... ball hit ... continue playing!

Game_Over:

    rcall convert_to_decimal        ; convert score to decimal value to be displayed on screen

    ldi COUNTER, 3                  ; loop in game_over screen 3 times
loop_between_last_scene_and_score:
    push COUNTER                    ; push COUNTER so that it can be reused in loop with
    different values

    ldi COUNTER, 100                ; freezes game frame when Game Over is triggered
freeze:
    ldi DRAW_TIMES, 5
    rcall draw_ball             ; draw frozen ball on screen

    ldi DRAW_TIMES, 5
```

```
    rcall draw_paddle           ; draw frozen paddle on screen

    ldi DRAW_TIMES, 2
    rcall draw_enemy            ; draw frozen paddle on screen

    dec COUNTER
    brne freeze

    ldi COUNTER, 255                ; freezes score screen
freezeScore:
    ldi DRAW_TIMES, 1
    rcall draw_yourScore            ;draw 'your score' text on screen

    cbr FLAG, 0b00000100            ; clear draw_score_10s flag
    ldi DRAW_TIMES, 5
    rcall draw_score_10s            ; draws left hand score number (10s)
    cbr FLAG, 0b00000100            ; clear draw_score_10s flag

    ldi DRAW_TIMES, 5
    rcall draw_score                ; draws right hand score number (1s)

    dec COUNTER
    brne freezeScore

    pop COUNTER                     ; restore saved value of COUNTER
    dec COUNTER
    brne loop_between_last_scene_and_score

    rjmp Start_Screen               ; ... restart the game

;=====================================================================
;
ROUTINES
; =====================================================================

; ---------------------------------------------------------------------
;                               HIT
; ---------------------------------------------------------------------

hit:                                ; Calculate positions and draws ball and paddle for one
game cycle
get_small:                          ; Decrease size of ball (ball going into screen)

    inc SCALING                     ; decrease size of ball

    rcall move_ball_x               ; update x position of the ball and checks boundaries
    rcall move_ball_y               ; update y position of the ball and checks boundaries

    rcall check_enemy_out_of_screen_x   ; check if enemy paddle is going out of screen x
    boundaries by following ball
    rcall check_enemy_out_of_screen_y   ; check if enemy paddle is going out of screen y
    boundaries by following ball

    ldi COUNTER, 5                  ; COUNTER will change how quickly the ball goes in and
    out the screen (1=FAST, 5=NORMAL, ...)
    rcall check_change_level        ; change level (COUNTER speed) depending on SCORE value

stay_small:

    clr PADDLE_DELTAX               ; clear paddle x movement values from previous loop
    clr PADDLE_DELTAY               ; clear paddle y movement values from previous loop
    rcall Check_Buttons_Paddle      ; check if buttons pressed to move paddle, while inside
    this loop it will be independent of ball movement

    rcall move_paddle_x             ; update x position of the paddle and checks boundaries
    rcall move_paddle_y             ; update y position of the paddle and checks boundaries

    ldi DRAW_TIMES, 5
    rcall draw_ball                 ; draw ball at its new position and new size

    ldi DRAW_TIMES, 5
```

```
    rcall draw_paddle               ; draw paddle at its new position and new size

    ldi DRAW_TIMES, 2
    rcall draw_enemy                ; draw enemy paddle on screen

    dec COUNTER
    brne stay_small                 ; keep looping COUNTER amout of times at current ball size

    cpi SCALING, 6                  ; once smallest radius (SCALING=6) is drawn ...
    brne get_small                  ; ... break out of loop

    rcall get_random_hit            ; fetch value from enemy_random_hits table
    call map_random_hit             ; map random hit directions to ball offset values

get_big:                            ; Increase size of ball (ball going out of screen)

    dec SCALING                     ; increase size of ball

    rcall move_ball_x               ; update x position of the ball and checks boundaries
    rcall move_ball_y               ; update y position of the ball and checks boundaries

    rcall check_enemy_out_of_screen_x   ; check if enemy paddle is going out of screen x
    boundaries by following ball
    rcall check_enemy_out_of_screen_y   ; check if enemy paddle is going out of screen y
    boundaries by following ball

    ldi COUNTER, 5                  ; COUNTER will change how quickly the ball goes in and
    out the screen (1=FAST, 5=NORMAL, ...)
    rcall check_change_level        ; change level (COUNTER speed) depending on SCORE value

stay_big:

    clr PADDLE_DELTAX               ; clear paddle x movement values from previous loop
    clr PADDLE_DELTAY               ; clear paddle y movement values from previous loop
    rcall Check_Buttons_Paddle      ; check if buttons pressed to move paddle, while inside
    this loop it will be independent of ball movement

    rcall move_paddle_x             ; update x position of the paddle and checks boundaries
    rcall move_paddle_y             ; update y position of the paddle and checks boundaries

    ldi DRAW_TIMES, 5
    rcall draw_ball                 ; draw ball at its new position and new size

    ldi DRAW_TIMES, 5
    rcall draw_paddle               ; draw paddle at its new position and new size

    ldi DRAW_TIMES, 2
    rcall draw_enemy                ; draw enemy paddle on screen

    dec COUNTER
    brne stay_big                   ; keep looping COUNTER amout of times at current ball size

    cpi SCALING, 1                  ; once biggest radius (SCALING=1) is drawn ...
    brne get_big                    ; ... break out of loop

    ret

; ---------------------------------------------------------------------
;                           DRAW BALL [DRAW_TIMES, SCALING]
; ---------------------------------------------------------------------

draw_ball:                  ; Draws one size of ball on screen

    push COUNTER            ; saving variables used outside ...
    push TEMPREG            ; ...
    push ZH                 ; ...
    push ZL                 ; ...

draw_again:                 ; draw same thing until it has been drawn DRAW_TIMES
times
```

```
    LDI XH, HIGH(2*Sin)     ; load X coordinates of circle in X registers
    LDI XL, LOW(2*Sin)
    LDI YH, HIGH(2*Cos)     ; load Y coordinates of circle in Y registers
    LDI YL, LOW(2*Cos)

    LDI COUNTER, 100        ; amount of coordinates in program memory

draw_loop:                  ; draw one coordinate pair of the ball until all have been drawn
    mov ZH, XH              ; start with x coordinate
    mov ZL, XL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; save it in TEMPREG

    rcall scale_loop        ; scale value to correct size of the ball

    add TEMPREG, XPOS       ; draw pixel relative to the centre of the ball

    sbrc FLAG, 7               ; clear ball_out_scr_x flag
    rcall change_pixel_x_value  ; ... check if pixel out of the screen
    out PORTB, TEMPREG          ; then draw pixel on screen
    ADIW XL, $01                ; increment pointer to next value

    mov ZH, YH              ; start with y coordinate
    mov ZL, YL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; save it in TEMPREG

    rcall scale_loop        ; scale value to correct size of the ball

    add TEMPREG, YPOS       ; draw pixel relative to the centre of the ball

    sbrc FLAG, 6               ; clear ball_out_scr_y flag
    rcall change_pixel_y_value  ; check if ball out of screen
    out PORTE, TEMPREG      ; then draw pixel on screen
    ADIW YL, $01            ; increment pointer to next value

    DEC COUNTER             ; if all values have been drawn ...
    breq return             ; ... exit loop

    RJMP draw_loop          ; ... else keep drawing pixels

return:
    dec DRAW_TIMES          ; if ball not drawn DRAW_TIMES times ...
    brne draw_again         ; ... draw all pixels again
                            ; else ...
    pop ZL                  ; restore saved values ...
    pop ZH                  ; ...
    pop TEMPREG             ; ...
    pop COUNTER             ; ...

    ret                     ; and return

; ---------------------------------------------------------------------------
;                           CHANGE PIXEL X VALUE [TEMPREG]
; ---------------------------------------------------------------------------

change_pixel_x_value:       ; Check if pixel should be drawn along the wall to simulate
squashed ball

    sbrs XPOS, 7            ; check if ball is in positive or negative x region
    rjmp ball_positive      ; ... if bit7 of XPOS is clear ball is in x>0
    rjmp ball_negative      ; ... if bit7 of XPOS is set ball is in x<0

ball_positive:              ; if ball in x>0
    cpi TEMPREG, 255        ; compare x pixel value with right screen limit
    brge draw_at_wall_right ; if pixel greater than limit, draw it against the wall instead
    (must be signed)
    ret                     ; ... else draw pixel normally

draw_at_wall_right:
    ldi TEMPREG, 255        ; value of right wall
```

```
    ret

ball_negative:              ;if ball in x<0
    cpi TEMPREG, 0          ;compare x pixel value with left screen limit
    brlt draw_at_wall_left  ; if pixel less than limit, draw against the wall instead (must
    be signed)
    ret

draw_at_wall_left:
    ldi TEMPREG, 0          ; value of left wall
    ret

; ---------------------------------------------------------------------
;                           CHANGE PIXEL Y VALUE [TEMPREG]
; ---------------------------------------------------------------------

change_pixel_y_value:       ; Check if y pixel should be drawn along the wall to simulate
squashed ball

    sbrs YPOS, 7            ; check if ball is in positive or negative y region
    rjmp ball_positive_y    ; ... if bit7 of YPOS is clear, ball is in y>0
    rjmp ball_negative_y    ; ... if bit7 of YPOS is set, ball is in y<0

ball_positive_y:            ; if ball in y>0 ...
    cpi TEMPREG, 255        ; compare y pixel value with upper screen limit
    brge draw_at_wall_up    ; if pixel greater than limit, draw it against the wall instead
    (must be signed)
    ret                     ; ... else draw pixel normally

draw_at_wall_up:
    ldi TEMPREG, 255        ; value of upper wall
    ret

ball_negative_y:            ; if ball in y<0 ...
    cpi TEMPREG, 0          ;compare y pixel value with lower screen limit
    brlt draw_at_wall_down  ; if pixel less than limit, draw against the wall instead (must
    be signed)
    ret                     ; ... else draw pixel normally

draw_at_wall_down:
    ldi TEMPREG, 0          ; value of bottom wall
    ret

; ---------------------------------------------------------------------
;                           CHECK CHANGE LEVEL [SCORE]
; ---------------------------------------------------------------------

check_change_level:

    ldi TEMPREG, 10     ; every 10 hits change level
    cp SCORE, TEMPREG
    brsh level2

    ret

level2:
    ldi COUNTER, 4

    ldi TEMPREG, 20         ; 10+10
    cp SCORE, TEMPREG
    brsh level3

    ret

level3:
    ldi COUNTER, 3
    ret

; ---------------------------------------------------------------------
;                           DRAW PADDLE [XPOS, YPOS, SCALING,
DRAW_TIMES]
```

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

draw_paddle:                ; draw player paddle on screen (same as DRAW BALL but paddle on
boundary check is performed in move_paddle_x/y)

    push ZH                 ; saving variables used outside ...
    push ZL                 ; ...
    push COUNTER            ; ...

draw_paddle_again:          ; draw all paddle pixels until paddle has been drawn DRAW_TIMES
times

    LDI XH, HIGH(2*PaddleX) ; load X coordinates of paddle in X registers
    LDI XL, LOW(2*PaddleX)
    LDI YH, HIGH(2*PaddleY) ; load Y coordinates of paddle in Y registers
    LDI YL, LOW(2*PaddleY)

    LDI COUNTER, 100        ; amount of coordinates in program memory

draw_paddle_loop:           ; draw one coordinate pair of the paddle until all have been drawn

    mov ZH, XH              ; start with x coordinate
    mov ZL, XL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; saves it in TEMPREG

    rcall divide_by_2       ; divide paddle x size by 2 (is hard coded full screen in
    program memory)

    add TEMPREG, PADDLE_XPOS    ; offset x pixel relative to the centre of the paddle

    out PORTB, TEMPREG      ; draw x pixel on oscilloscope
    ADIW XL, $01            ; move pointer to next x pixel

    mov ZH, YH              ; move to y coordinate
    mov ZL, YL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; save it in TEMPREG

    rcall divide_by_2       ; divide paddle y size by 2 (is hard coded full screen in
    program memory)

    add TEMPREG, PADDLE_YPOS    ; offset y pixel relative to the centre of the paddle

    out PORTE, TEMPREG      ; draw y pixel on oscilloscope
    ADIW YL, $01            ; move pointer to next y pixel

    DEC COUNTER             ; if all values have been drawn ...
    breq paddle_return      ; ... continue
    RJMP draw_paddle_loop   ; ... else loop again to draw next pixel pair

paddle_return:
    dec DRAW_TIMES          ; if paddle not drawn DRAW_TIMES times ...
    brne draw_paddle_again  ; ... draw all pixels again

    pop COUNTER             ; ... else restore saved values
    pop ZL
    pop ZH
    ret

; ---------------------------------------------------------------------
;                           DIVIDE BY 2 [TEMPREG]
; ---------------------------------------------------------------------

divide_by_2:                    ; Divide value in TEMPREG by 2
    lsr TEMPREG                 ;logical shift right (division by two)
    subi TEMPREG, 0b11000000    ; add 64 to convert number to bipolar mode (4-quadrants)
    ret

; ---------------------------------------------------------------------
;                           SCALE LOOP [TEMPREG, SCALING]
```

```
; ---------------------------------------------------------------------------
scale_loop:                 ; Divide value in TEMPREG by 2, SCALING amount of times:

    push COUNTER            ; push COUNTER to use as temporary register

    cpi SCALING, 0          ; if SCALING is already zero (fullscreen) ...
    breq return2            ; ... do not scale

    mov COUNTER, SCALING    ; ... else use COUNTER as temporary register

again:
    rcall divide_by_2       ; call divide_by_2
    dec COUNTER             ; if not scaled SCALING amount of times (saved in counter) ...
    brne again              ; ... loop back again
                            ; ... else
return2:
    pop COUNTER             ; restore saved value of COUNTER
    ret

; ---------------------------------------------------------------------------
;                           DRAW ENEMY [XPOS, YPOS,
DRAW_TIMES]
; ---------------------------------------------------------------------------

draw_enemy:                 ; draw enemy paddle on screen (same as DRAW PADDLE but boundary
check is not implemented yet)

    push ZH                 ; saving variables used outside ...
    push ZL                 ; ...
    push COUNTER            ; ...

draw_enemy_again:           ; draw all enemy paddle pixels until paddle has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*PaddleX) ; load X coordinates of enemy paddle in X registers
    LDI XL, LOW(2*PaddleX)
    LDI YH, HIGH(2*PaddleY) ; load Y coordinates of enemy paddle in Y registers
    LDI YL, LOW(2*PaddleY)

    LDI COUNTER, 100        ; amount of coordinates in program memory

draw_enemy_loop:            ; draw one coordinate pair of the enemy paddle until all have
been drawn

    mov ZH, XH              ; start with x coordinate
    mov ZL, XL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; save it in TEMPREG

    rcall divide_by_2       ; scale enemy paddle x size to ...
    rcall divide_by_2       ; ... 8 times smaller than length of full screen ...
    rcall divide_by_2       ; ... (half side length=16)

    add TEMPREG, XPOS       ; offset x pixel relative to the centre of the enemy paddle

    sbrc FLAG, 5               ; skip if part of enemy paddle is not out of screen
    rcall change_pixel_x_value  ; ... check if pixel out of the screen
    out PORTB, TEMPREG      ; draw x pixel on oscilloscope
    ADIW XL, $01            ; move pointer to next x pixel

    mov ZH, YH              ; move to y coordinate
    mov ZL, YL
    LPM                     ; fetch value from program memory
    MOV TEMPREG, r0         ; save it in TEMPREG

    rcall divide_by_2       ; scale enemy paddle y size to ...
    rcall divide_by_2       ; ... 8 times smaller than length of full screen ...
    rcall divide_by_2       ; ... (half side length=16)

    add TEMPREG, YPOS       ; offset y pixel relative to the centre of the enemy paddle
```

```
    sbrc FLAG, 4                ; skip if part of enemy paddle is not out of screen
    rcall change_pixel_y_value  ; ... check if pixel out of the screen
    out PORTE, TEMPREG      ; draw y pixel on oscilloscope
    ADIW YL, $01            ; move pointer to next y pixel

    DEC COUNTER             ; if all values have been drawn ...
    breq enemy_return       ; ... continue
    RJMP draw_enemy_loop    ; ... else loop again to draw next pixel pair

enemy_return:
    dec DRAW_TIMES          ; if enemy paddle not drawn DRAW_TIMES times ...
    brne draw_enemy_again   ; ... draw all pixels again

    pop COUNTER             ; ... else restore all saved values
    pop ZL
    pop ZH

    ret

; ---------------------------------------------------------------------------
;                           MOVE_PADDLE_X [PADDLE_XPOS, PADDLE_DELTAX]
; ---------------------------------------------------------------------------

move_paddle_x:                      ; Move player paddle along x direction and manage
behaviour at screen edges:

    add PADDLE_XPOS, PADDLE_DELTAX  ; update paddle x position with offset given by keyboard
    input
    mov TEMPREG, PADDLE_XPOS        ; save value in TEMPREG

    sbrs PADDLE_DELTAX, 7           ; check if paddle input movement is positive or negative
    rjmp dx_positive                ; ... if bit7 of PADDLE_DELTAX is clear, paddle is
    moving right
    rjmp dx_negative                ; ... elseif bit7 of PADDLE_DELTAX is set, paddle is
    moving left

dx_positive:                            ; paddle is moving right:
    sbrs PADDLE_XPOS, 7                 ; check if paddle is in positive or negative x region
    rjmp dx_positive_and_xpos_positive  ; ... if bit7 of PADDLE_XPOS is clear, paddle is in
    x>0
    rjmp dx_positive_and_xpos_negative  ; ... elseif bit7 of PADDLE_XPOS is set, paddle is
    in x<0

dx_negative:                            ; paddle is moving left:
    sbrs PADDLE_XPOS, 7                 ; check if paddle is in positive or negative x region
    rjmp dx_negative_and_xpos_positive  ; ... if bit7 of PADDLE_XPOS is clear, paddle is in
    x>0
    rjmp dx_negative_and_xpos_negative  ; ... elseif bit7 of PADDLE_XPOS is set, paddle is
    in x<0

dx_positive_and_xpos_positive:  ; paddle is in x>0 and is moving right:
    cpi TEMPREG, 64             ; check if paddle centre is such that paddle is entirely
    within right boundary (64 = 128 - paddle half side length)
    brlt within_boundary        ; if centre of paddle lower than 64 the paddle is within
    boundary ...
    ldi TEMPREG, 63             ; ... else paddle is outside boundary ...
    mov PADDLE_XPOS, TEMPREG    ; ... snap back paddle inside boundary (XPOS = 63)
    ret

dx_positive_and_xpos_negative:  ; paddle is in x>0 and is moving left:
    rjmp within_boundary        ; ... paddle can only be within boundary

dx_negative_and_xpos_positive:  ; paddle is in x<0 and is moving right:
    rjmp within_boundary        ; ... paddle can only be within boundary

dx_negative_and_xpos_negative:  ; paddle is in x<0 and is moving left:
    cpi TEMPREG, 0b11000000     ; check if paddle centre is such that paddle is entirely
    within left boundary (0b11000000 = -64 = -128 + paddle half side length)
    brge within_boundary        ; if centre of paddle higher than or equal to -64 the paddle
    is within boundary ...
    ldi TEMPREG, 0b11000000     ; ... else paddle is outside boundary ...
```

```
    mov PADDLE_XPOS, TEMPREG    ; ... snap back paddle to inside boundary (XPOS = -64)
    ret

within_boundary:                ; paddle is within screen boundary
    mov PADDLE_XPOS, TEMPREG    ; confirm position
    ret

; ---------------------------------------------------------------------------
;                           MOVE_PADDLE_Y [PADDLE_YPOS, PADDLE_DELTAY]
; ---------------------------------------------------------------------------

move_paddle_y:                      ; Move player paddle along y direction and manage
behaviour at screen edges:

    add PADDLE_YPOS, PADDLE_DELTAY  ; update paddle y position with offset given by keyboard
    input
    mov TEMPREG, PADDLE_YPOS        ; save value in TEMPREG

    sbrs PADDLE_DELTAY, 7           ; check if paddle input movement is positive or negative
    rjmp dy_positive                ; ... if bit7 of PADDLE_DELTAY is clear, paddle is
    moving up
    rjmp dy_negative                ; ... elseif bit7 of PADDLE_DELTAY is set, paddle is
    moving down

dy_positive:                            ; paddle is moving up:
    sbrs PADDLE_YPOS, 7                 ; check if paddle is in positive or negative y region
    rjmp dy_positive_and_ypos_positive  ; ... if bit7 of PADDLE_YPOS is clear, paddle is in
    y>0
    rjmp dy_positive_and_ypos_negative  ; ... elseif bit7 of PADDLE_YPOS is set, paddle is
    in y<0

dy_negative:                            ; paddle is moving down:
    sbrs PADDLE_YPOS, 7                 ; check if paddle is in positive or negative y region
    rjmp dy_negative_and_ypos_positive  ; ... if bit7 of PADDLE_YPOS is clear, paddle is in
    y>0
    rjmp dy_negative_and_ypos_negative  ; ... elseif bit7 of PADDLE_YPOS is set, paddle is
    in y<0

dy_positive_and_ypos_positive:      ; paddle is in y>0 and is moving up:
    cpi TEMPREG, 64                 ; check if paddle centre is such that paddle is entirely
    within top boundary (64 = 128 - paddle half side length)
    brlt y_within_boundary          ; if centre of paddle lower than 64 the paddle is within
    boundary ...
    ldi TEMPREG, 63                 ; ... else paddle is outside boundary ...
    mov PADDLE_YPOS, TEMPREG        ; ... snap back paddle inside boundary (YPOS = 63)
    ret

dy_positive_and_ypos_negative:      ; paddle is in y>0 and is moving down:
    rjmp y_within_boundary          ; ... paddle can only be within boundary

dy_negative_and_ypos_positive:      ; paddle is in y<0 and is moving up:
    rjmp y_within_boundary          ; ... paddle can only be within boundary

dy_negative_and_ypos_negative:      ; paddle is in y<0 and is moving down:
    cpi TEMPREG, 0b11000000         ; check if paddle centre is such that paddle is entirely
    within bottom boundary (0b11000000 = -64 = -128 + paddle half side length)
    brge y_within_boundary          ; if centre of paddle higher than or equal to -64 the
    paddle is within boundary ...
    ldi TEMPREG, 0b11000000         ; ... else paddle is outside boundary ...
    mov PADDLE_YPOS, TEMPREG        ; ... snap back paddle to inside boundary (YPOS = -64)
    ret

y_within_boundary:                  ; paddle is within screen boundary
    mov PADDLE_YPOS, TEMPREG        ; confirm position
    ret

; ---------------------------------------------------------------------------
;                           MOVE_BALL_X [XPOS, DELTAX]
; ---------------------------------------------------------------------------

move_ball_x:                            ; Move ball along x axis from keypad values and
```

```
manages bounces on the walls.

    add XPOS, DELTAX                    ; update x position of the ball

    cbr FLAG, 0b10000000                ; clear ball_out_scr_x flag used in
    check_ball_out_of_screen_x
    rcall check_ball_out_of_screen_x    ; check if part of the ball is out of the screen. If
    so sets flag.

    sbrc FLAG, 7                        ; check ball_out_scr_x flag (is part of x ball out
    of the screen?)
    rjmp bounce_back_x                  ; if part of ball is out, bounce back
    ret                                 ; ... else position of the ball is ok, return

bounce_back_x:                          ; part of the ball is out (right or left)
    neg DELTAX                          ; switch moving direction, ball will 'bounce' on wall
    ret

; ---------------------------------------------------------------------
;                           MOVE_BALL_Y [YPOS, DELTAY]
; ---------------------------------------------------------------------

move_ball_y:                            ; Move ball along y axis from keypad values and
manages bounces on the walls.

    add YPOS, DELTAY                    ; update y position of the ball

    cbr FLAG, 0b01000000                ; clear ball_out_scr_y flag used in
    check_ball_out_of_screen_y
    rcall check_ball_out_of_screen_y    ; check if part of the ball is out of the screen. If
    so sets flag.

    sbrc FLAG, 6                        ; check ball_out_scr_y flag (is part of y ball out
    of the screen?)
    rjmp bounce_back_y                  ; if part of ball is out, bounce back
    ret                                 ; ... else position of the ball is ok, return

bounce_back_y:                          ; part of the ball is out (right or left)
    neg DELTAY                          ; switch moving direction, ball will 'bounce' on wall
    ret

; ---------------------------------------------------------------------
;                           CALC_RADIUS [SCALING]
; ---------------------------------------------------------------------

calc_radius:                ; Calculate radius of ball and store it in TEMPREG2. r ===
2**(6-SCALING)

    ldi TEMPREG, 1          ;load TEMPREG2 with ...
    mov TEMPREG2, TEMPREG   ;... 1

    ldi TEMPREG, 7          ; load TEMPREG with ...
    sub TEMPREG, SCALING    ;... 7 - scaling

keep_shifting_x:
    lsl TEMPREG2            ; shifts 0b00000001 by ...
    dec TEMPREG             ; ... 6 - scaling ...
    brne keep_shifting_x    ; ... i.e. 2**(6-SCALING)

    ret

; ---------------------------------------------------------------------
;                           CHECK BALL ON
PADDLE
; ---------------------------------------------------------------------

check_ball_on_paddle:           ; Check if ball overlaps paddle enough to be recognised as
hit (centre of ball within paddle)

    mov TEMPREG, PADDLE_XPOS    ; load x position of the paddle in TEMPREG
    subi TEMPREG, 192           ; calculate position of right edge of paddle = PADDLE_XPOS+64
```

```
    cp XPOS, TEMPREG            ; compare centre of ball with right edge of paddle
    brge ball_x_out_right       ; if centre of ball greater than or equal to paddle right
    edge, the ball is outside of the paddle
                                ; ... else
    subi TEMPREG, 127           ; calculate position of left edge of paddle = PADDLE_XPOS-63
    = paddle right edge - 127
    cp XPOS, TEMPREG            ; compare centre of ball with left edge of paddle
    brlt ball_x_out_left        ; if centre of ball lower than paddle left edge, the ball is
    outside of the paddle
                                ; ... else

    mov TEMPREG, PADDLE_YPOS    ; load y position of the paddle in TEMPREG
    subi TEMPREG, 192           ; calculate position of top edge of paddle = PADDLE_YPOS+64

    cp YPOS, TEMPREG            ; compare centre of ball with top edge of paddle
    brge ball_y_out_up          ; if centre of ball greater than or equal to paddle top
    edge, the ball is outside of the paddle
                                ; ... else
    subi TEMPREG, 127           ; calculate position of bottom edge of paddle =
    PADDLE_YPOS-63 = paddle top edge - 127
    cp YPOS, TEMPREG            ; compare centre of ball with bottom edge of paddle
    brlt ball_y_out_down        ; if centre of ball lower than paddle bottom edge, the ball
    is outside of the paddle

    inc SCORE                   ; ball has been hit, scores 1 point
    ldi TEMPREG, 100            ; check if score ...
    cp SCORE, TEMPREG           ; ... greater >= 100
    brsh cap_score              ; if so reset it to 0
    ret

ball_x_out_right:               ; if ball is outside of the paddle
ball_x_out_left:
ball_y_out_up:
ball_y_out_down:
    sbr FLAG, 0b00000010        ; set ball_not_on_paddle flag
    ret

cap_score:
    ldi TEMPREG, 0
    mov SCORE, TEMPREG
    ret
; ---------------------------------------------------------------------------
;                           CHECK BALL OUT OF SCREEN
X
; ---------------------------------------------------------------------------

check_ball_out_of_screen_x:     ; Check if the ball is out of the screen left or right edges

    rcall calc_radius           ; calculate ball radius. Stored in TEMPREG2

    ldi TEMPREG, 0
    add TEMPREG, XPOS           ; load y centre of ball (XPOS) in TEMPREG
    sub TEMPREG, TEMPREG2       ; calculate and load leftmost point of ball (XPOS-r) in
    TEMPREG
    add TEMPREG2, XPOS          ; calculate and load rightmost point of ball (XPOS+r) in
    TEMPREG2

    cp TEMPREG, TEMPREG2        ; compare positions of leftmost and rightmost edges of the
    ball
    brge ball_out_screen_x      ; if leftmost edge > rightmost edge (happens if ball goes
    out right into negative region or goes out left into positive region) branch

    ret                         ; ... else part of the ball is not out of the screen

ball_out_screen_x:              ; part of the ball is out of the screen
    sbr FLAG, 0b10000000        ; set ball_out_scr_x flag
    ret

; ---------------------------------------------------------------------------
;                           CHECK BALL OUT OF SCREEN
```

```
Y
; ---------------------------------------------------------------------

check_ball_out_of_screen_y:     ; Check if the ball is out of the screen left or right edges

    rcall calc_radius           ; calculate ball radius. Stored in TEMPREG2

    ldi TEMPREG, 0
    add TEMPREG, YPOS           ; load x centre of ball (YPOS) in TEMPREG
    sub TEMPREG, TEMPREG2       ; calculate and load bottommost point of ball (YPOS-r) in
    TEMPREG
    add TEMPREG2, YPOS          ; calculate and load topmost point of ball (XPOS+r) in
    TEMPREG2

    cp TEMPREG, TEMPREG2        ; compare positions of bottommost and topmost edges of the
    ball
    brge ball_out_screen_y      ; if bottommost edge > topmost edge (happens if ball goes
    out up into negative region or goes out down into positive region)

    ret

ball_out_screen_y:              ; part of the ball is out of the screen
    sbr FLAG, 0b01000000        ; set ball_out_scr_y flag
    ret

; ---------------------------------------------------------------------
;                           CHECK_ENEMY_OUT_OF_SCREEN_X [XPOS]
; ---------------------------------------------------------------------

check_enemy_out_of_screen_x:            ; Check if enemy paddle is outside of the screen x
boundaries (movement is automatic as it follows the ball XPOS).

    cbr FLAG, 0b00100000                ; clear enemy_out_scr_x  flag

    ldi TEMPREG, 16                     ; load TEMPREG2 with ...
    mov TEMPREG2, TEMPREG               ; ... half side length ('r') of enemy paddle (16)
    ldi TEMPREG, 0
    add TEMPREG, XPOS                   ; load x centre of enemy paddle (centred on ball) in
    TEMPREG
    sub TEMPREG, TEMPREG2               ; calculate and load leftmost point of enemy paddle
    (XPOS-r) in TEMPREG
    add TEMPREG2, XPOS                  ; calculate and load rightmost point of enemy paddle
    (XPOS+r) in TEMPREG2

    cp TEMPREG, TEMPREG2                ; compare positions of leftmost and rightmost edges
    of the enemy paddle
    brge enemy_out_screen_x             ; if leftmost edge > rightmost edge (happens if
    enemy paddle goes out right into negative region or goes out left into positive region)

    ret

enemy_out_screen_x:                     ; part of the enemy paddle is out of the screen

    sbr FLAG, 0b00100000                ; set enemy_out_scr_x  flag so that it can be drawn
    differently on oscilloscope
    ret

; ---------------------------------------------------------------------
;                           CHECK_ENEMY_OUT_OF_SCREEN_Y [YPOS]
; ---------------------------------------------------------------------

check_enemy_out_of_screen_y:            ; Check if enemy paddle is outside of the screen y
boundaries (movement is automatic as it follows the ball YPOS).

    cbr FLAG, 0b00010000                ; clear enemy_out_scr_y flag

    ldi TEMPREG, 16                     ; load TEMPREG2 with ...
    mov TEMPREG2, TEMPREG               ; ... half side length ('r') of enemy paddle (16)
    ldi TEMPREG, 0
    add TEMPREG, YPOS                   ; load y centre of enemy paddle (centred on ball) in
    TEMPREG
```

```
    sub TEMPREG, TEMPREG2               ; calculate and load leftmost point of enemy paddle
    (YPOS-r) in TEMPREG
    add TEMPREG2, YPOS                  ; calculate and load rightmost point of enemy paddle
    (YPOS+r) in TEMPREG2

    cp TEMPREG, TEMPREG2                ; compare positions of bottommost and topmost edges
    of the enemy paddle
    brge enemy_out_screen_y             ; if bottommost edge > leftmost edge (happens if
    enemy paddle goes out up into negative region or goes out down into positive region)

    ret

enemy_out_screen_y:                     ; part of the enemy paddle is out of the screen

    sbr FLAG, 0b00010000                ; set enemy_out_scr_y  flag so that it can be drawn
    differently on oscilloscope
    ret

; ---------------------------------------------------------------------------
;                           DRAW START MENU [XPOS, YPOS, SCALING,
DRAW_TIMES]
; ---------------------------------------------------------------------------
draw_startMenu:                         ; Draws the start menu text on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_startMenu:                   ; draw all pixels until start menu text has been
drawn DRAW_TIMES times

    LDI XH, HIGH(2*StartMenu20X)        ; load X coordinates of start menu text in X registers
    LDI XL, LOW(2*StartMenu20X)
    LDI YH, HIGH(2*StartMenu20Y)        ; load Y coordinates of start menu text in Y registers
    LDI YL, LOW(2*StartMenu20Y)

    ldi COUNTER, 8                      ; main counter: 8 for 'double'; 5 for 'basic'; 5 for
    'alt', 6 for '20'
    push COUNTER                        ; save main counter
draw_more_points:                       ; text data is hard coded with more than 255 bytes.
Need to loop COUNTER amount of times to draw all pixels.
    pop COUNTER                         ; retrieve main counter

    dec COUNTER                         ; decrement main counter
    breq return_startMenu               ; if looped COUNTER amount of times, all pixels have
    been drawn ...

    push COUNTER                        ; save updated value of main counter
                                        ; ... else ...
    ldi COUNTER, 225                    ; load sub counter: 250 for 'double'; 187 for
    'basic'; 162 for 'alt', 255 for '20'

draw_loop_startMenu:                    ; draw one coordinate pair of the start menu text
until all have been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel
```

```
    dec COUNTER                         ; if all values for the sub counter have been drawn
    ...
    breq draw_more_points               ; ... go to next set of pixels
    rjmp draw_loop_startMenu            ; ... else loop again to draw next pixel in current
    set

return_startMenu:                       ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_startMenu           ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;---------------------------------------------------------------------
;                           DRAW YOUR SCORE [TEMPREG2, SCORE]
; --------------------------------------------------------------------
draw_yourScore:                         ; Draws the your score text on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_yourScore:                   ; draw all pixels until start menu text has been
drawn DRAW_TIMES times

    LDI XH, HIGH(2*YourScoreX)      ; load X coordinates of start menu text in X registers
    LDI XL, LOW(2*YourScoreX)
    LDI YH, HIGH(2*YourScoreY)      ; load Y coordinates of start menu text in Y registers
    LDI YL, LOW(2*YourScoreY)

    ldi COUNTER, 8                      ; main counter: 8 for 'double'; 5 for 'basic'; 5 for
    'alt', 6 for '20'
    push COUNTER                        ; save main counter
draw_more_score_points:                     ; text data is hard coded with more than 255
bytes. Need to loop COUNTER amount of times to draw all pixels.
    pop COUNTER                         ; retrieve main counter

    dec COUNTER                         ; decrement main counter
    breq return_yourScore               ; if looped COUNTER amount of times, all pixels have
    been drawn ...

    push COUNTER                        ; save updated value of main counter
                                        ; ... else ...
    ldi COUNTER, 115                    ; load sub counter: 250 for 'double'; 187 for
    'basic'; 162 for 'alt', 255 for '20'

draw_loop_yourScore:                    ; draw one coordinate pair of the start menu text
until all have been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have been drawn
    ...
```

```
    breq draw_more_score_points             ; ... go to next set of pixels
    rjmp draw_loop_yourScore            ; ... else loop again to draw next pixel in current
    set

return_yourScore:                       ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_yourScore           ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;-----------------------------------------------------------------------
;                           DRAW SCORE 1s AND 10s [DRAW_TIMES]
; ----------------------------------------------------------------------
draw_score:
    ldi TEMPREG, 0
    cp SCORE, TEMPREG
    breq score0
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score1
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score2
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score3
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score4
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score5
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score6
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score7
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score8
    inc TEMPREG
    cp SCORE, TEMPREG
    breq score9
    ret         ; just in case
score0:
    rcall draw_score0
    ret
score1:
    rcall draw_score1
    ret
score2:
    rcall draw_score2
    ret
score3:
    rcall draw_score3
    ret
score4:
    rcall draw_score4
    ret
score5:
    rcall draw_score5
    ret
score6:
    rcall draw_score6
    ret
```

```
score7:
    rcall draw_score7
    ret
score8:
    rcall draw_score8
    ret
score9:
    rcall draw_score9
    ret

draw_score_10s:

    sbr FLAG, 0b00000100    ; set flag draw_score_10s
    ldi TEMPREG, 0
    cp TEMPREG2, TEMPREG
    breq score0
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score1
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score2
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score3
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score4
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score5
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score6
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score7
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score8
    inc TEMPREG
    cp TEMPREG2, TEMPREG
    breq score9
    cbr FLAG, 0b00000100
    ret
; ---------------------------------------------------------------------------
;                           DRAW SCORE 0 [DRAW_TIMES]
; ---------------------------------------------------------------------------
draw_score0:                            ; Draws the number 0 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score0:                  ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score0x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score0x)
    LDI YH, HIGH(2*score0y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score0y)

    ldi COUNTER, 84                 ; load sub counter with PM table length

draw_loop_score0:                   ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG
```

```
    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score0               ; ... loop again to draw next pixel in current set

return_score0:                        ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score0          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret


;---------------------------------------------------------------------
;                           DRAW SCORE 1 [DRAW_TIMES]
; --------------------------------------------------------------------
draw_score1:                            ; Draws the number 1 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score1:                    ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score1x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score1x)
    LDI YH, HIGH(2*score1y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score1y)

    ldi COUNTER, 42                 ; load sub counter with PM table length

draw_loop_score1:                     ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel
```

```
    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score1               ; ... loop again to draw next pixel in current set

return_score1:                        ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score1          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;---------------------------------------------------------------------
;                           DRAW SCORE 2 [DRAW_TIMES]
; --------------------------------------------------------------------
draw_score2:                            ; Draws the number 2 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score2:                    ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score2x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score2x)
    LDI YH, HIGH(2*score2y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score2y)

    ldi COUNTER, 105                    ; load sub counter with PM table length

draw_loop_score2:                     ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score2               ; ... loop again to draw next pixel in current set

return_score2:                        ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score2          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret
```

```
;-----------------------------------------------------------------------
;                           DRAW SCORE 3 [DRAW_TIMES]
; ---------------------------------------------------------------------
draw_score3:                            ; Draws the number 3 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score3:                    ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score3x)           ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score3x)
    LDI YH, HIGH(2*score3y)           ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score3y)

    ldi COUNTER, 84                   ; load sub counter with PM table length

draw_loop_score3:                     ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score3               ; ... loop again to draw next pixel in current set

return_score3:                          ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score3              ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;-----------------------------------------------------------------------
;                           DRAW SCORE 4 [DRAW_TIMES]
; ---------------------------------------------------------------------
draw_score4:                            ; Draws the number 4 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score4:                    ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score4x)           ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score4x)
```

```
    LDI YH, HIGH(2*score4y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score4y)

    ldi COUNTER, 63                 ; load sub counter with PM table length

draw_loop_score4:                   ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score4               ; ... loop again to draw next pixel in current set

return_score4:                      ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score4          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;---------------------------------------------------------------------
;                           DRAW SCORE 5 [DRAW_TIMES]
; --------------------------------------------------------------------
draw_score5:                            ; Draws the number 5 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score5:                  ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score5x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score5x)
    LDI YH, HIGH(2*score5y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score5y)

    ldi COUNTER, 105                    ; load sub counter with PM table length

draw_loop_score5:                   ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32
```

```
    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score5               ; ... loop again to draw next pixel in current set

return_score5:                         ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score5          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;-----------------------------------------------------------------------
;                           DRAW SCORE 6 [DRAW_TIMES]
; ----------------------------------------------------------------------
draw_score6:                            ; Draws the number 6 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score6:                     ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score6x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score6x)
    LDI YH, HIGH(2*score6y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score6y)

    ldi COUNTER, 105                    ; load sub counter with PM table length

draw_loop_score6:                      ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score6               ; ... loop again to draw next pixel in current set
```

```
return_score6:                      ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score6          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

    ;-----------------------------------------------------------------------
;                           DRAW SCORE 7 [DRAW_TIMES]
; ---------------------------------------------------------------------
draw_score7:                            ; Draws the number 7 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score7:                  ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score7x)         ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score7x)
    LDI YH, HIGH(2*score7y)         ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score7y)

    ldi COUNTER, 42                 ; load sub counter with PM table length

draw_loop_score7:                   ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score7               ; ... loop again to draw next pixel in current set

return_score7:                      ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score7          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;-----------------------------------------------------------------------
;                           DRAW SCORE 8 [DRAW_TIMES]
; ---------------------------------------------------------------------
```

```
draw_score8:                            ; Draws the number 8 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score8:                      ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score8x)             ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score8x)
    LDI YH, HIGH(2*score8y)             ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score8y)

    ldi COUNTER, 126                    ; load sub counter with PM table length

draw_loop_score8:                       ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score8               ; ... loop again to draw next pixel in current set

return_score8:                          ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score8              ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

;---------------------------------------------------------------------
;                           DRAW SCORE 9 [DRAW_TIMES]
; --------------------------------------------------------------------
draw_score9:                            ; Draws the number 9 on screen:

    push ZH                             ; saving variables used outside ...
    push ZL                             ; ...
    push COUNTER                        ; ...

draw_again_score9:                      ; draw all pixels until start menu text has been drawn
DRAW_TIMES times

    LDI XH, HIGH(2*score9x)             ; load X coordinates of score number in X registers
    LDI XL, LOW(2*score9x)
    LDI YH, HIGH(2*score9y)             ; load Y coordinates of score number in Y registers
    LDI YL, LOW(2*score9y)

    ldi COUNTER, 105                    ; load sub counter with PM table length
```

```
draw_loop_score9:                   ; draw one coordinate pair of score value until all have
been drawn

    mov ZH, XH                          ; start with x coordinate
    mov ZL, XL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    sbrc FLAG, 2
    subi TEMPREG, 32

    out PORTB, TEMPREG                  ; draw x pixel on oscilloscope
    adiw XL, $01                        ; move pointer to next x pixel

    mov ZH, YH                          ; then move to y coordinate
    mov ZL, YL
    lpm                                 ; fetch value from program memory
    mov TEMPREG, r0                     ; save it in TEMPREG

    out PORTE, TEMPREG                  ; draw y pixel on oscilloscope
    adiw YL, $01                        ; move pointer to next y pixel

    dec COUNTER                         ; if all values for the sub counter have not been
    drawn ...
    brne draw_loop_score9               ; ... loop again to draw next pixel in current set

return_score9:                      ; all pixels of the start menu text have been drawn

    dec DRAW_TIMES                      ; if start menu text not drawn DRAW_TIMES times ...
    brne draw_again_score9          ; ... draw it again

    pop COUNTER                         ; ... else restore saved values
    pop ZL
    pop ZH

    ret

; ---------------------------------------------------------------------
;                           KEYPAD ROUTINES
; ---------------------------------------------------------------------

setupReadColumns:               ; initialises I/O port D to read columns
    ldi TEMPREG, 0b00001111     ; bits 4-7 (col): Inputs (0), bits 0-3 (rows): Outputs (1)
    out DDRD, TEMPREG           ; send ^ config to DDRD
    ldi TEMPREG, 0b11110000     ; bits 4-7 (col): pull-up resistors (1) , bits 0-3 (rows):
    drive low (0)
    out PORTD, TEMPREG          ; send ^ config to PORTD
    rcall settle                ; let pins settle
    ret

setupReadRows:                  ; initialises I/O port D to read rows
    ldi TEMPREG, 0b11110000     ; bits 4-7 (col): Outputs (1), bits 0-3 (rows): Inputs (0)
    out DDRD, TEMPREG           ; send ^ config to DDRD
    ldi TEMPREG, 0b00001111     ; bits 4-7 (col): drive low (0), bits 0-3 (rows): pull-up
    resistors (1)
    out PORTD, TEMPREG          ; send ^ config to PORTD
    rcall settle                ; let pins settle
    ret

settle:                         ; allows input pins to settle after pulling them up
    rcall DEL600mus             ; 0.25s delay
    ret

; ---------------------------------------------------------------------
;                           CHECK BUTTONS [SETS FLAG, DELTAX, DELTAY]
; ---------------------------------------------------------------------

Check_Buttons_Hit:              ; Checks if a keypad button has been pressed to give hit
direction to ball
```

```
    push PADDLE_DELTAX          ; save paddle offsets ...
    push PADDLE_DELTAY          ; ... as we are only interested in updating ball offsets

    call Check1Pressed          ; check if button 1 is pressed
    call Check2Pressed          ; check if button 2 is pressed
    call Check3Pressed          ; check if button 3 is pressed
    call Check4Pressed          ; check if button 4 is pressed
    call Check5Pressed
    call Check6Pressed          ; check if button 6 is pressed
    call Check7Pressed          ; check if button 7 is pressed
    call Check8Pressed          ; check if button 8 is pressed
    call Check9Pressed          ; check if button 9 is pressed

    pop PADDLE_DELTAY           ; restore saved values ...
    pop PADDLE_DELTAX           ; ...

    ret

Check_Buttons_Paddle:           ; Checks if a keypad button has been pressed to give hit
direction to ball

    push DELTAX                 ; save ball offsets ...
    push DELTAY                 ; ... as we are only interested in updating paddle offsets

    call Check1Pressed          ; check if button 1 is pressed
    call Check2Pressed          ; check if button 2 is pressed
    call Check3Pressed          ; check if button 3 is pressed
    call Check4Pressed          ; check if button 4 is pressed
    ;call Check5Pressed
    call Check6Pressed          ; check if button 6 is pressed
    call Check7Pressed          ; check if button 7 is pressed
    call Check8Pressed          ; check if button 8 is pressed
    call Check9Pressed          ; check if button 9 is pressed

    pop DELTAY                  ; restore saved values ...
    pop DELTAX                  ; ...

    ret

Check1Pressed:                                  ; Check if button 1 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b01110000     ; if column 1 pressed ...
                    breq column_pressed1        ; ... branch
                    brne button_not_pressed1    ; ... else return

column_pressed1:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00000111     ; if row 1 not pressed ...
                    brne button_not_pressed1    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 251             ; ... load ball X offset corresponding to
                    button 1 (-5)
                    ldi DELTAY, 5               ; ... load ball Y offset corresponding to
                    button 1 (+5)
                    ldi TEMPREG, 251
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 1 (-5)
                    ldi TEMPREG, 5
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 1 (+5)

button_not_pressed1:
                    ret


Check2Pressed:                                  ; Check if button 2 on keypad has been
```

```
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b10110000     ; if column 2 pressed ...
                    breq column_pressed2        ; ... branch
                    brne button_not_pressed2    ; ... else return

column_pressed2:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00000111     ; if row 2 not pressed ...
                    brne button_not_pressed2    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 0               ; ... load ball X offset corresponding to
                    button 2 (+0)
                    ldi DELTAY, 5               ; ... load ball y offset corresponding to
                    button 2 (+5)
                    ldi TEMPREG, 0
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 2 (+0)
                    ldi TEMPREG, 5
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 2 (+5)

button_not_pressed2:
                    ret

Check3Pressed:                                  ; Check if button 3 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b11010000     ; if column 3 pressed ...
                    breq column_pressed3        ; ... branch
                    brne button_not_pressed3    ; ... else return

column_pressed3:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00000111     ; if row 3 not pressed ...
                    brne button_not_pressed3    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 5               ; ... load ball X offset corresponding to
                    button 3 (+5)
                    ldi DELTAY, 5               ; ... load ball Y offset corresponding to
                    button 3 (+5)
                    ldi TEMPREG, 5
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 3 (+5)
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 3 (+5)

button_not_pressed3:
                    ret

Check4Pressed:                                  ; Check if button 4 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b01110000     ; if column 4 pressed ...
                    breq column_pressed4        ; ... branch
                    brne button_not_pressed4    ; ... else return

column_pressed4:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001011     ; if row 4 not pressed ...
```

```
                    brne button_not_pressed4    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 251             ; ... load ball X offset corresponding to
                    button 4 (-5)
                    ldi DELTAY, 0               ; ... load ball Y offset corresponding to
                    button 4 (+0)
                    ldi TEMPREG, 251
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 4 (-5)
                    ldi TEMPREG, 0
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 4 (+0)

button_not_pressed4:
                    ret

Check5Pressed:                                  ; Check if button 5 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b10110000     ; if column 5 pressed ...
                    breq column_pressed5        ; ... branch
                    brne button_not_pressed5    ; ... else return

column_pressed5:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001011     ; if row 5 not pressed ...
                    brne button_not_pressed5    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 0               ; ... load ball X offset corresponding to
                    button 5 (+0)
                    ldi DELTAY, 0               ; ... load ball Y offset corresponding to
                    button 5 (+0)
                    ldi TEMPREG, 0
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 5 (+0)
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 5 (+0)

button_not_pressed5:
                    ret

Check6Pressed:                                  ; Check if button 6 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b11010000     ; if column 6 pressed ...
                    breq column_pressed6        ; ... branch
                    brne button_not_pressed6    ; ... else return

column_pressed6:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001011     ; if row 6 not pressed ...
                    brne button_not_pressed6    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 5               ; ... load ball X offset corresponding to
                    button 6 (+5)
                    ldi DELTAY, 0               ; ... load ball Y offset corresponding to
                    button 6 (+0)
                    ldi TEMPREG, 5
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 6 (+5)
                    ldi TEMPREG, 0
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
```

```
                    button 6 (+0)

button_not_pressed6:
                    ret

Check7Pressed:                              ; Check if button 7 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b01110000     ; if column 7 pressed ...
                    breq column_pressed7        ; ... branch
                    brne button_not_pressed7    ; ... else return

column_pressed7:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001101     ; if row 7 not pressed ...
                    brne button_not_pressed7    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 251             ; ... load ball X offset corresponding to
                    button 7 (-5)
                    ldi DELTAY, 251             ; ... load ball Y offset corresponding to
                    button 7 (-5)
                    ldi TEMPREG, 251
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 7 (-5)
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 7 (-5)

button_not_pressed7:
                    ret

Check8Pressed:                              ; Check if button 8 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b10110000     ; if column 8 pressed ...
                    breq column_pressed8        ; ... branch
                    brne button_not_pressed8    ; ... else return

column_pressed8:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001101     ; if row 8 not pressed ...
                    brne button_not_pressed8    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 0               ; ... load ball X offset corresponding to
                    button 8 (+0)
                    ldi DELTAY, 251             ; ... load ball Y offset corresponding to
                    button 8 (-5)
                    ldi TEMPREG, 0
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 8 (+0)
                    ldi TEMPREG, 251
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 8 (-5)

button_not_pressed8:
                    ret

Check9Pressed:                              ; Check if button 9 on keypad has been
pressed and maps to in game offsets:
                    rcall setupReadColumns      ; set up keypad to read
                    columns
                    in TEMPREG, PinD            ; read column status from keypad
                    cpi TEMPREG, 0b11010000     ; if column 9 pressed ...
                    breq column_pressed9        ; ... branch
```

```
                    brne button_not_pressed9    ; ... else return

column_pressed9:
                    rcall setupReadRows         ; set up keypad to read rows
                    in TEMPREG, PinD            ; read row status from keypad
                    cpi TEMPREG, 0b00001101     ; if row 9 not pressed ...
                    brne button_not_pressed9    ; ... loop back
                    sbr FLAG, 0b00000001        ; ... else set button_pressed flag

                    ldi DELTAX, 5               ; ... load ball X offset corresponding to
                    button 9 (+5)
                    ldi DELTAY, 251             ; ... load ball Y offset corresponding to
                    button 9 (-5)
                    ldi TEMPREG, 5
                    mov PADDLE_DELTAX, TEMPREG  ; ... load paddle X offset corresponding to
                    button 9 (+5)
                    ldi TEMPREG, 251
                    mov PADDLE_DELTAY, TEMPREG  ; ... load paddle Y offset corresponding to
                    button 9 (-5)

button_not_pressed9:
                    ret

; ---------------------------------------------------------------------------
;                       RANDOM ENEMY HIT DIRECTION
; ---------------------------------------------------------------------------

get_random_hit:         ; load value from enemy_random_hits table

    LPM                 ; fetch value from program memory
    MOV RANDOM_N, r0    ; save it in RANDOM_N
    adiw Z, 1           ; move pointer to next value
    inc ENEMY_HIT_COUNTER ; increment number of times enemy has hit the ball

    ldi TEMPREG, 254
    cp ENEMY_HIT_COUNTER, TEMPREG   ; reached end of random hit table
    brsh reset_pointer
    ret

reset_pointer:
    ldi ZL, low(2*enemy_random_hits)    ; load pointer to predetermined directions for enemy
    hits
    ldi ZH, high(2*enemy_random_hits)   ; ... pointer has to be conserved throughout the game
    ret

map_random_hit:         ; maps directions in which ball is it to corresponding offset values

    cpi RANDOM_N, 0
    breq random0
    cpi RANDOM_N, 1
    breq random1
    cpi RANDOM_N, 2
    breq random2
    cpi RANDOM_N, 3
    breq random3
    cpi RANDOM_N, 4
    breq random4
    cpi RANDOM_N, 5
    breq random5
    cpi RANDOM_N, 6
    breq random6
    cpi RANDOM_N, 7
    breq random7
    cpi RANDOM_N, 8
    breq random8
    ret
random0:
    ldi DELTAX, 5
    ldi DELTAY, 251
    rjmp return_from_random
random1:
```

```
    ldi DELTAX, 251
    ldi DELTAY, 5
    rjmp return_from_random
random2:
    ldi DELTAX, 0
    ldi DELTAY, 5
    rjmp return_from_random
random3:
    ldi DELTAX, 5
    ldi DELTAY, 5
    rjmp return_from_random
random4:
    ldi DELTAX, 251
    ldi DELTAY, 0
    rjmp return_from_random
random5:
    ldi DELTAX, 0
    ldi DELTAY, 0
    rjmp return_from_random
random6:
    ldi DELTAX, 5
    ldi DELTAY, 0
    rjmp return_from_random
random7:
    ldi DELTAX, 251
    ldi DELTAY, 251
    rjmp return_from_random
random8:
    ldi DELTAX, 0
    ldi DELTAY, 251
    rjmp return_from_random
return_from_random:
    ret

; ---------------------------------------------------------------------------
;                        CONVERT SCORE TO DECIMAL
; ---------------------------------------------------------------------------

convert_to_decimal:
    ldi TEMPREG, 0
    mov TEMPREG2, TEMPREG

sub10:
    ldi TEMPREG, 10
    cp SCORE, TEMPREG
    brsh greater_than_9
    rjmp finished_factorisation

greater_than_9:
    sub SCORE, TEMPREG  ; SCORE-10
    inc TEMPREG2        ; 10s increased by 1
    rjmp sub10

finished_factorisation: ; TEMPREG2 = 10s, SCORE = 1s
    ret


;********* DELAYS ***********************************************
; costs: rcall -> 3 CC | ret -> 4 CC | LDI -> 1 CC | SBIW -> 2 CC | BRNE -> 1/2 CC | DEC = 1

;BIGDEL:
BigDEL:                         ; 0.5 s delay routine
        rcall Del20ms           ; 20 ms delay
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
        rcall Del20ms           ; ,,
```

```
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        rcall Del20ms               ; ,,
        ret


;DEL20MS:
DEL20ms:                            ; 20 ms delay routine
            LDI XH, HIGH(65535)     ; $FF, this is the max delay we can make in a single
            routine
            LDI XL, LOW(65535)
COUNT:
            SBIW XL, 1
            BRNE COUNT
            RET

;DEL15MS:
DEL15ms:                            ;15 ms delay routine
        LDI XH, HIGH(47997)
        LDI XL, LOW(47997)
COUNT1:
        SBIW XL, 1
        BRNE COUNT1
        RET

;DEL4P1MS:
DEL4P1ms:                           ;4 ms delay routine
        LDI XH, HIGH(12797)
        LDI XL, LOW(12797)
COUNT2:
        SBIW XL, 1
        BRNE COUNT2
        RET

;DEL600MUS:
DEL600mus:                          ;600 mus delay routine
            LDI XH, HIGH(1917)
            LDI XL, LOW(1917)
COUNT3:
            SBIW XL, 1
            BRNE COUNT3
            RET

;DEL100MUS:                         ;100 mus delay routine
DEL100mus:
            LDI XH, HIGH(316)
            LDI XL, LOW(316)
COUNT4:
            SBIW XL, 1
            BRNE COUNT4
            RET

;DEL30MUS:                          ;30 mus delay routine
DEL30mus:
            ldi XL, 155
COUNT5:
            dec XL
            BRNE COUNT5
```

```
            RET

;******************** PM
********************************************************************************************************************
Sin:
.db $ff, $fe, $fd, $fc, $fa, $f8, $f5, $f2, $ee, $ea, $e6, $e1, $db, $d6, $cf, $c9, $c2,
$bb, $b4, $ad, $a5, $9d, $95, $8d, $85, $7d, $75, $6d, $65, $5d, $55, $4e, $46, $3f, $38,
$32, $2c, $26, $20, $1b, $16, $12, $e, $a, $7, $5, $3, $1, $0, $0, $0, $0, $1, $3, $5, $7,
$a, $e, $12, $16, $1b, $20, $26, $2c, $32, $38, $3f, $46, $4e, $55, $5d, $65, $6d, $75, $7d,
$85, $8d, $95, $9d, $a5, $ad, $b4, $bb, $c2, $c9, $cf, $d6, $db, $e1, $e6, $ea, $ee, $f2,
$f5, $f8, $fa, $fc, $fd, $fe, $ff

Cos:
.db $85, $8d, $95, $9d, $a5, $ad, $b4, $bb, $c2, $c9, $cf, $d6, $db, $e1, $e6, $ea, $ee,
$f2, $f5, $f8, $fa, $fc, $fd, $fe, $ff, $ff, $fe, $fd, $fc, $fa, $f8, $f5, $f2, $ee, $ea,
$e6, $e1, $db, $d6, $cf, $c9, $c2, $bb, $b4, $ad, $a5, $9d, $95, $8d, $85, $7d, $75, $6d,
$65, $5d, $55, $4e, $46, $3f, $38, $32, $2c, $26, $20, $1b, $16, $12, $e, $a, $7, $5, $3,
$1, $0, $0, $0, $0, $1, $3, $5, $7, $a, $e, $12, $16, $1b, $20, $26, $2c, $32, $38, $3f,
$46, $4e, $55, $5d, $65, $6d, $75, $7d

PaddleX:
.db $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00,
$00, $00, $00, $00, $00, $00, $00, $00, $a, $15, $1f, $2a, $35, $3f, $4a, $55, $5f, $6a,
$74, $7f, $8a, $94, $9f, $aa, $b4, $bf, $c9, $d4, $df, $e9, $f4, $ff , $ff, $ff, $ff, $ff,
$ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff,
$ff, $ff, $ff, $ff, $f4, $e9, $df, $d4, $c9, $bf, $b4, $aa, $9f, $94, $8a, $7f, $74, $6a,
$5f, $55, $4a, $3f, $35, $2a, $1f, $15, $a, $0

PaddleY:
.db $00, $a, $15, $1f, $2a, $35, $3f, $4a, $55, $5f, $6a, $74, $7f, $8a, $94, $9f, $aa, $b4,
$bf, $c9, $d4, $df, $e9, $f4, $ff , $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff,
$ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $f4, $e9, $df,
$d4, $c9, $bf, $b4, $aa, $9f, $94, $8a, $7f, $74, $6a, $5f, $55, $4a, $3f, $35, $2a, $1f,
$15, $a, $0, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00,
$00, $00, $00, $00, $00, $00, $00, $00

StartMenu20X:
.db $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $18, $18, $19, $1a, $1b, $1c, $1e, $1f, $20, $21, $22, $24, $25, $26, $27,
$28, $2a, $2b, $2c, $2d, $2e, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $2e, $2d, $2c, $2b, $2a, $28, $27,
$26, $25, $24, $22, $21, $20, $1f, $1e, $1c, $1b, $1a, $19, $18, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3d,
$3e, $3f, $40, $42, $43, $44, $45, $46, $48, $49, $4a, $4b, $4c, $4e, $4f, $50, $51, $52,
$54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $52, $51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44,
$43, $42, $40, $3f, $3e, $3d, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3d, $3e, $3f, $40, $42, $43, $44,
$45, $46, $48, $49, $4a, $4b, $4c, $4e, $4f, $50, $51, $52, $54, $78, $76, $75, $74, $73,
$72, $70, $6f, $6e, $6d, $6c, $6a, $69, $68, $67, $66, $64, $63, $62, $61, $60, $60, $60,
$60, $60, $60
.db $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$61, $62, $63, $64, $66, $67, $68, $69, $6a, $6c, $6d, $6e, $6f, $70, $72, $73, $74, $75,
$76, $78, $60, $60, $61, $61, $62, $63, $63, $64, $64, $65, $66, $66, $67, $67, $68, $69,
$69, $6a, $6a, $6b, $6c, $84, $85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $91, $92,
$93, $94, $96, $97, $98, $99, $9a, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9a, $99, $98, $97, $96, $94,
$93, $92, $91, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84, $84, $84, $84, $84,
$84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84,
$85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $91, $92, $93, $94, $96, $97, $98, $99,
$9a, $9c, $a8, $a9, $aa, $ab, $ac, $ae, $af, $b0, $b1, $b2
.db $b4, $b5, $b6, $b7, $b8, $ba, $bb, $bc, $bd, $be, $c0, $c0, $c0, $c0, $c0, $c0, $c0,
$c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $be, $bd,
$bc, $bb, $ba, $b8, $b7, $b6, $b5, $b4, $b2, $b1, $b0, $af, $ae, $ac, $ab, $aa, $a9, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a9, $aa, $ab, $ac, $ae, $af, $b0, $b1, $b2, $b4, $b5, $b6, $b7, $b8,
$ba, $bb, $bc, $bd, $be, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0,
$c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c0, $c1, $c2, $c3, $c4, $c6, $c7, $c8, $c9,
$ca, $cc, $cd, $ce, $cf, $d0, $d2, $d3, $d4, $d5, $d6, $d8, $d8, $d8, $d8, $d8, $d8, $d8,
$d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $d8, $9c, $9c, $9c,
```

```
$9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$9c, $9d, $9e, $9f, $a0, $a2, $a3, $a4, $a5, $a6, $a8, $a9, $aa, $ab, $ac, $ae, $af, $b0,
$b1, $b2, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4
.db $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b2, $b1, $b0, $af, $ae,
$ac, $ab, $aa, $a9, $a8, $a6, $a5, $a4, $a3, $a2, $a0, $9f, $9e, $9d, $9c, $84, $84, $84,
$84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84,
$84, $84, $85, $85, $86, $87, $87, $88, $88, $89, $8a, $8a, $8b, $8b, $8c, $8d, $8d, $8e,
$8e, $8f, $90, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84, $82, $81, $80, $7f,
$7e, $7c, $7b, $7a, $79, $78, $6c, $6a, $69, $68, $67, $66, $64, $63, $62, $61, $60, $5e,
$5d, $5c, $5b, $5a, $58, $57, $56, $55, $54, $54, $54, $55, $55, $56, $57, $57, $58, $58,
$59, $5a, $5a, $5b, $5b, $5c, $5d, $5d, $5e, $5e, $5f, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $31, $32, $33, $34, $36, $37, $38, $39, $3a, $3c, $3d, $3e, $3f, $40, $42, $43, $44,
$45, $46, $48
.db $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48,
$48, $48, $48, $48, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $18, $18, $18, $18, $18, $17, $16, $16, $15, $15, $14, $13, $13, $12, $12,
$11, $10, $10, $f, $f, $e, $d, $d, $c, $c, $c, $c, $c, $c, $c, $c, $c, $c, $c, $c, $c, $c,
$c, $c, $c, $c, $c, $c, $c, $c, $c, $c, $d, $e, $f, $10, $12, $13, $14, $15, $16, $18, $19,
$1a, $1b, $1c, $1e, $1f, $20, $21, $22, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $22, $21, $20, $1f, $1e,
$1c, $1b, $1a, $19, $18, $16, $15, $14, $13, $12, $10, $f, $e, $d, $c, $54, $55, $56, $57,
$58, $5a, $5b, $5c, $5d, $5e, $60, $61, $62, $63, $64, $66, $67, $68, $69, $6a, $6c, $6c,
$6b, $6a, $6a, $69, $69, $68, $67, $67, $66, $66, $65, $64, $64, $63, $63, $62, $61, $61,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $78, $78, $78
.db $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b,
$8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84,
$82, $81, $80, $7f, $7e, $7c, $7b, $7a, $79, $78, $a8, $a9, $aa, $ab, $ac, $ae, $af, $b0,
$b1, $b2, $b4, $b5, $b6, $b7, $b8, $ba, $bb, $bc, $bd, $be, $c0, $c0, $bf, $be, $be, $bd,
$bd, $bc, $bb, $bb, $ba, $ba, $b9, $b8, $b8, $b7, $b7, $b6, $b5, $b5, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $9c, $9a, $99, $98, $97, $96, $94, $93, $92, $91, $90, $8e, $8d, $8c, $8b, $8a, $88,
$87, $86, $85, $84, $84, $85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $91, $92, $93,
$94, $96, $97, $98, $99, $9a, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$9c, $9c, $9c, $9c, $9c, $9c
.db $9c, $9c, $9c, $9c, $9c, $9a, $99, $98, $97, $96, $94, $93, $92, $91, $90, $8e, $8d,
$8c, $8b, $8a, $88, $87, $86, $85, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84,
$84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $77, $76, $76,
$75, $75, $74, $73, $73, $72, $72, $71, $70, $70, $6f, $6f, $6e, $6d, $6d, $6c, $6c, $6c,
$6b, $6a, $6a, $69, $69, $68, $67, $67, $66, $66, $65, $64, $64, $63, $63, $62, $61, $61,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $61, $62, $63, $64, $66, $67, $68, $69, $6a,
$6c, $6d, $6e, $6f, $70, $72, $73, $74, $75, $76, $78, $3c, $3d, $3e, $3f, $40, $42, $43,
$44, $45, $46, $48, $49, $4a, $4b, $4c, $4e, $4f, $50, $51, $52, $54, $54, $53, $52, $52,
$51, $51, $50, $4f, $4f, $4e, $4e, $4d, $4c, $4c, $4b, $4b, $4a, $49, $49, $48, $48, $48,
$48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48,
$48, $48, $18, $19, $1a, $1b, $1c, $1e, $1f, $20, $21, $22, $24, $25, $26, $27, $28, $2a,
$2b, $2c, $2d, $2e, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $2e, $2d, $2c, $2b, $2a, $28, $27, $26, $25,
$24, $22, $21, $20, $1f, $1e, $1c, $1b, $1a, $19, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $19, $1a, $1b,
$1c, $1e, $1f, $20, $21, $22, $24, $25, $26, $27, $28, $2a, $2b, $2c, $2d, $2e, $30

StartMenu20Y:
.db $b4, $b6, $b8, $bb, $bd, $c0, $c2, $c4, $c7, $c9, $cc, $ce, $d0, $d3, $d5, $d8, $da,
$dc, $df, $e1, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4,
$e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e2, $e1, $e0, $df, $de, $dc, $db, $da, $d9, $d8,
$d6, $d5, $d4, $d3, $d2, $d0, $cf, $ce, $cd, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $b4, $b6, $b8, $bb, $bd,
$c0, $c2, $c4, $c7, $c9, $cc, $ce, $d0, $d3, $d5, $d8, $da, $dc, $df, $e1, $e4, $e4, $e4,
$e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4,
$e4, $e4, $e2, $e1, $e0, $df, $de, $dc, $db, $da, $d9, $d8, $d6, $d5, $d4, $d3, $d2, $d0,
$cf, $ce, $cd, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $ca, $c9, $c8, $c7, $c6, $c4, $c3,
```

```
$c2, $c1, $c0, $be, $bd, $bc, $bb, $ba, $b8, $b7, $b6, $b5, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b6,
$b8, $bb, $bd, $c0, $c2, $c4, $c7, $c9, $cc, $ce, $d0, $d3, $d5, $d8, $da, $dc, $df, $e1,
$e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4,
$e4, $e4, $e4, $e4, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b5, $b6, $b7, $b8, $ba, $bb, $bc,
$bd, $be, $c0, $c1, $c2, $c3, $c4, $c6, $c7, $c8
.db $c9, $ca, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cd, $ce, $cf, $d0, $d2, $d3, $d4, $d5, $d6, $d8,
$d9, $da, $db, $dc, $de, $df, $e0, $e1, $e2, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4,
$e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4
.db $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b5, $b6, $b7, $b8, $ba, $bb,
$bc, $bd, $be, $c0, $c1, $c2, $c3, $c4, $c6, $c7, $c8, $c9, $ca, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cd, $ce, $cf, $d0, $d2, $d3, $d4, $d5, $d6, $d8, $d9, $da, $db, $dc, $de, $df, $e0, $e1,
$e2, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4, $e4,
$e4, $e4, $e4, $e4, $e4, $78, $7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d, $90, $92, $94,
$97, $99, $9c, $9e, $a0, $a3, $a5, $a8, $a8, $a5, $a3, $a0, $9e, $9c, $99, $97, $94, $92,
$90, $8d, $8b, $88, $86, $84, $81, $7f, $7c, $7a, $78, $78, $7a, $7c, $7f, $81, $84, $86,
$88, $8b, $8d, $90, $92, $94, $97, $99, $9c, $9e, $a0, $a3, $a5, $a8, $78, $7a, $7c, $7f,
$81, $84, $86, $88, $8b, $8d, $90, $92, $94, $97, $99, $9c, $9e, $a0, $a3, $a5, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a5, $a3, $a0, $9e, $9c, $99, $97, $94, $92, $90, $8d, $8b, $88, $86, $84,
$81, $7f, $7c, $7a, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d,
$90, $92, $94, $97, $99, $9c, $9e, $a0, $a3, $a5, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8
.db $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a5, $a3, $a0,
$9e, $9c, $99, $97, $94, $92, $90, $8d, $8b, $88, $86, $84, $81, $7f, $7c, $7a, $78, $a8,
$a5, $a3, $a0, $9e, $9c, $99, $97, $94, $92, $90, $8d, $8b, $88, $86, $84, $81, $7f, $7c,
$7a, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78
.db $78, $7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d, $90, $92, $94, $97, $99, $9c, $9e,
$a0, $a3, $a5, $a8, $90, $91, $92, $93, $94, $96, $97, $98, $99, $9a, $9c, $9d, $9e, $9f,
$a0, $a2, $a3, $a4, $a5, $a6, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a5, $a3, $a0, $9e, $9c, $99, $97,
$94, $92, $90, $8d, $8b, $88, $86, $84, $81, $7f, $7c, $7a, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $79,
$7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c,
$6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c,
$6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $69, $67, $64, $62, $60, $5d, $5b,
$58, $56, $54, $51, $4f, $4c, $4a, $48, $45, $43, $40, $3e, $3c, $3c, $3e, $40, $43, $45,
$48, $4a, $4c, $4f, $51, $54, $56, $58, $5b, $5d, $60, $62, $64, $67, $69, $6c, $6c, $6c,
$6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c,
$6c, $6c, $69, $67, $64, $62, $60, $5d, $5b, $58, $56, $54, $51, $4f, $4c, $4a, $48, $45,
$43, $40, $3e, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30
.db $2d, $2b, $28, $26, $24, $21, $1f, $1c, $1a, $18, $15, $13, $10, $e, $c, $9, $7, $4, $2,
$0, $0, $1, $2, $3, $4, $6, $7, $8, $9, $a, $c, $d, $e, $f, $10, $12, $13, $14, $15, $16,
$18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $18, $18, $19, $1a, $1b, $1c, $1e, $1f, $20, $21, $22, $24, $25, $26, $27, $28
.db $2a, $2b, $2c, $2d, $2e, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $2d, $2b, $28, $26, $24, $21, $1f,
$1c, $1a, $18, $15, $13, $10, $e, $c, $9, $7, $4, $2, $0, $0, $1, $2, $3, $4, $6, $7, $8,
$9, $a, $c, $d, $e, $f, $10, $12, $13, $14, $15, $16, $18, $18, $19, $1a, $1b, $1c, $1e,
$1f, $20, $21, $22, $24, $25, $26, $27, $28, $2a, $2b, $2c, $2d, $2e, $30, $30, $2e, $2d,
$2c, $2b, $2a, $28, $27, $26, $25, $24, $22, $21, $20, $1f, $1e, $1c, $1b, $1a, $19, $18,
$18, $16, $15, $14, $13, $12, $10, $f, $e, $d, $c, $a, $9, $8, $7, $6, $4, $3, $2, $1, $0,
$0, $1, $2, $3, $4, $6, $7, $8, $9, $a, $c, $d, $e, $f, $10, $12, $13, $14, $15, $16, $18,
$18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30,
```

```
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $2d, $2b, $28, $26, $24, $21, $1f, $1c,
$1a, $18, $15, $13, $10, $e, $c, $9, $7, $4, $2, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0,
$0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $1, $2, $3, $4, $6, $7, $8, $9, $a, $c, $d,
$e, $f, $10, $12, $13, $14, $15, $16, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18,
$18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $18, $19, $1a, $1b, $1c, $1e, $1f,
$20, $21, $22, $24, $25, $26, $27, $28, $2a, $2b, $2c, $2d, $2e, $30, $30, $30, $30, $30,
$30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30, $30

enemy_random_hits:
.db 2, 5, 1, 5, 7, 1, 5, 1, 4, 7, 3, 0, 3, 5, 8, 3, 8, 8, 0, 6, 2, 1, 8, 6, 4, 0, 4, 4, 0,
6, 1, 4, 8, 8, 4, 8, 0, 8, 0, 0, 7, 7, 7, 6, 5, 7, 3, 7, 5, 3, 4, 4, 8, 4, 3, 5, 1, 8, 2, 1,
8, 0, 5, 5, 1, 8, 4, 2, 2, 4, 5, 2, 1, 2, 2, 8, 2, 3, 5, 7, 8, 5, 4, 2, 6, 6, 8, 6, 0, 0, 7,
5, 5, 1, 3, 6, 5, 8, 2, 2, 3, 7, 1, 3, 2, 3, 5, 8, 1, 1, 5, 8, 6, 2, 1, 7, 1, 7, 5, 2, 0, 8,
4, 0, 3, 5, 2, 4, 7, 2, 7, 8, 4, 3, 4, 4, 3, 3, 1, 4, 4, 5, 7, 2, 2, 5, 5, 1, 4, 5, 3, 8, 0,
4, 4, 2, 6, 1, 4, 4, 1, 8, 7, 1, 0, 7, 5, 4, 2, 4, 0, 1, 8, 3, 8, 8, 0, 5, 3, 5, 8, 7, 0, 2,
8, 7, 8, 4, 2, 3, 5, 1, 4, 6, 4, 2, 1, 5, 6, 6, 0, 2, 8, 2, 3, 7, 1, 5, 5, 7, 8, 2, 5, 0, 2,
0, 4, 5, 8, 2, 5, 3, 5, 2, 2, 1, 2, 6, 0, 3, 0, 0, 8, 0, 6, 4, 7, 3, 6, 2, 8, 3, 1, 2, 2, 1,
4, 0, 0, 7, 8, 0, 4

YourScoreX:
.db $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3b, $3a, $3a, $39, $39, $38, $37, $37, $36, $36, $35, $34, $34, $33,
$33, $32, $31, $31, $30, $30, $30, $30, $31, $31, $32, $33, $33, $34, $34, $35, $36, $36,
$37, $37, $38, $39, $39, $3a, $3a, $3b, $3c, $3c, $3c, $3d, $3d, $3e, $3f, $3f, $40, $40,
$41, $42, $42, $43, $43, $44, $45, $45, $46, $46, $47, $48, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $55, $56,
$57, $58, $5a, $5b, $5c, $5d, $5e, $60, $61, $62, $63, $64, $66, $67, $68, $69, $6a, $6c,
$6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c,
$6c, $6c, $6c, $6c, $6a, $69, $68, $67, $66, $64, $63, $62, $61, $60, $5e, $5d, $5c, $5b,
$5a, $58, $57, $56, $55, $54, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81,
$82, $84, $85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $9c, $9c, $9c,
$9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$9c, $9d, $9e, $9f, $a0, $a2, $a3, $a4, $a5, $a6, $a8, $a9, $aa, $ab, $ac, $ae, $af, $b0,
$b1, $b2, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b2, $b1, $b0, $af, $ae, $ac, $ab, $aa, $a9, $a8, $a6,
$a5, $a4, $a3, $a2, $a0, $9f, $9e, $9d, $9c, $9c
.db $9d, $9d, $9e, $9f, $9f, $a0, $a0, $a1, $a2, $a2, $a3, $a3, $a4, $a5, $a5, $a6, $a6,
$a7, $a8, $a8, $a8, $a9, $a9, $aa, $ab, $ab, $ac, $ac, $ad, $ae, $ae, $af, $af, $b0, $b1,
$b1, $b2, $b2, $b3, $b4, $cc, $ca, $c9, $c8, $c7, $c6, $c4, $c3, $c2, $c1, $c0, $be, $bd,
$bc, $bb, $ba, $b8, $b7, $b6, $b5, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b5, $b6, $b7, $b8, $ba, $bb,
$bc, $bd, $be, $c0
.db $c2, $c3, $c4, $c6, $c7, $c8, $c9, $ca, $cc, $b4, $b4, $b5, $b5, $b6, $b7, $b7, $b8,
$b8, $b9, $ba, $ba, $bb, $bb, $bc
.db  $bd, $be, $be, $bf, $c0, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $91, $92, $93, $94, $96, $97, $98, $99,
$9a, $9c, $9d, $9e, $9f, $a0, $a2, $a3, $a4, $a5, $a6, $a8, $a8, $a8, $a8, $a8, $a8, $a8,
$a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a8, $a6, $a5,
$a4, $a3, $a2, $a0, $9f, $9e, $9d, $9c, $9a, $99, $98, $97, $96, $94, $93, $92, $91, $90,
$90, $90, $91, $91, $92, $93, $93, $94, $94, $95, $96, $96, $97, $97, $98, $99, $99, $9a,
$9a, $9b, $9c, $9c, $9c, $9d, $9d, $9e, $9f, $9f, $a0, $a0, $a1, $a2, $a2, $a3, $a3, $a4,
$a5, $a5, $a6, $a6, $a7, $a8, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c,
$6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6c, $6d, $6e, $6f, $70, $72, $73, $74, $75,
$76, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $84, $84, $84, $84, $84, $84,
$84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $82, $81,
$80, $7f, $7e, $7c, $7b, $7a, $79, $78, $76, $75, $74, $73, $72, $70, $6f, $6e, $6d, $6c,
$60, $5e, $5d, $5c, $5b, $5a, $58, $57, $56, $55, $54, $52, $51, $50, $4f, $4e, $4c, $4b,
$4a, $49, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48, $48,
$48, $48, $48, $48, $48, $48, $48, $49, $4a, $4b, $4c, $4e, $4f, $50, $51, $52, $54, $55,
$56, $57, $58, $5a, $5b, $5c, $5d, $5e, $60, $24, $25, $26, $27, $28, $2a, $2b, $2c, $2d,
$2e, $30, $31, $32, $33, $34, $36, $37, $38, $39, $3a, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3a, $39,
$38, $37, $36, $34, $33, $32, $31, $30, $2e, $2d, $2c, $2b, $2a, $28, $27, $26, $25, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $25, $26, $27, $28, $2a, $2b, $2c, $2d, $2e, $30, $31, $32, $33, $34,
$36, $37, $38, $39, $3a, $3c

YourScoreY:
.db $9c, $9d, $9e, $9f, $a0, $a2, $a3, $a4, $a5, $a6, $a8, $a9, $aa, $ab, $ac, $ae, $af,
```

```
$b0, $b1, $b2, $b4, $b4, $b5, $b6
.db  $b8, $ba, $bb, $bc, $bd, $be, $c0, $c1, $c2, $c3, $c4, $c6, $c7, $c8, $c9, $ca, $cc,
$cc, $ca, $c9, $c8, $c7, $c6, $c4, $c3, $c2, $c1, $c0, $be, $bd, $bc, $bb, $ba, $b8, $b7,
$b6, $b5, $b4, $b4, $b5, $b6, $b7, $b8, $ba, $bb, $bc, $bd, $be, $c0, $c1, $c2, $c3, $c4,
$c6, $c7, $c8, $c9, $ca, $cc, $9c, $9e, $a0, $a3, $a5, $a8, $aa, $ac, $af, $b1, $b4, $b6,
$b8, $bb, $bd, $c0, $c2, $c4, $c7, $c9, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $c9, $c7, $c4, $c2, $c0,
$bd, $bb, $b8, $b6, $b4, $b1, $af, $ac, $aa, $a8, $a5, $a3, $a0, $9e, $9c, $9c, $9c, $9c,
$9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$cc, $c9, $c7, $c4, $c2, $c0, $bd, $bb, $b8, $b6, $b4, $b1, $af, $ac, $aa, $a8, $a5, $a3,
$a0, $9e, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c, $9c,
$9c, $9c, $9c, $9c, $9c, $9c, $9c, $9e, $a0, $a3, $a5, $a8, $aa, $ac, $af, $b1, $b4, $b6,
$b8, $bb, $bd, $c0, $c2, $c4, $c7, $c9, $cc, $9c, $9e, $a0, $a3, $a5, $a8, $aa, $ac, $af,
$b1, $b4, $b6, $b8, $bb, $bd, $c0, $c2, $c4, $c7, $c9, $cc, $cc, $cc, $cc, $cc, $cc, $cc,
$cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $cc, $ca, $c9,
$c8, $c7, $c6, $c4, $c3, $c2, $c1, $c0, $be, $bd, $bc, $bb, $ba, $b8, $b7, $b6, $b5, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4, $b4,
$b4, $b4, $b4, $b4, $b4, $b4, $b4, $b2, $b1, $b0, $af, $ae, $ac, $ab, $aa, $a9, $a8, $a6,
$a5, $a4, $a3, $a2, $a0, $9f, $9e, $9d, $9c, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $62, $64, $67, $69, $6c,
$6e, $70, $73, $75, $78, $7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90
.db  $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78
.db  $78, $78, $78, $78, $78, $78, $60, $62, $64, $67, $69, $6c, $6e, $70, $73, $75, $78,
$7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $8e, $8d, $8c, $8b,
$8a, $88, $87, $86, $85, $84, $82, $81, $80, $7f, $7e, $7c, $7b, $7a, $79, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $76, $75, $74, $73, $72, $70, $6f, $6e, $6d, $6c, $6a, $69, $68,
$67, $66, $64, $63, $62, $61, $60, $60, $62, $64, $67, $69, $6c, $6e, $70, $73, $75, $78,
$7a, $7c, $7f, $81, $84, $86, $88, $8b, $8d, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $8d, $8b, $88, $86,
$84, $81, $7f, $7c, $7a, $78, $75, $73, $70, $6e, $6c, $69, $67, $64, $62, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $62, $64, $67, $69, $6c, $6e, $70, $73, $75, $78, $7a, $7c, $7f,
$81, $84, $86, $88, $8b, $8d, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $60, $60, $60, $60, $60, $60, $60, $60,
$60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $60, $61, $62, $63, $64,
$66, $67, $68, $69, $6a, $6c, $6d, $6e, $6f, $70, $72, $73, $74, $75, $76, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b,
$8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90

score0x:
.db $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74,
$74, $74, $74, $74, $74, $75, $76, $77, $78, $7a, $7b, $7c, $7d, $7e, $80, $81, $82, $83,
$84, $86, $87, $88, $89, $8a, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c,
$8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8a, $89, $88, $87, $86, $84, $83,
$82, $81, $80, $7e, $7d, $7c, $7b, $7a, $78, $77, $76, $75, $74

score0y:
.db $24, $26, $28, $2b, $2d, $30, $32, $34, $37, $39, $3c, $3e, $40, $43, $45, $48, $4a,
$4c, $4f, $51, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $54, $54, $54, $51, $4f, $4c, $4a, $48, $45, $43, $40, $3e, $3c,
$39, $37, $34, $32, $30, $2d, $2b, $28, $26, $24, $24, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24

score1x:
.db $74, $74, $75, $75, $76, $77, $77, $78, $78, $79, $7a, $7a, $7b, $7b, $7c, $7d, $7d,
$7e, $7e, $7f, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80, $80,
$80, $80, $80, $80, $80, $80, $80

score1y:
.db 'Z','B', 'F', 'G', 'C', 'D', $48, $48, $49, $49, $4a, $4b, $4b, $4c, $4c, $4d, $4e, $4e,
$4f, $4f, $50, $51, $51, $52, $52, $53, $54, $54, $51, $4f, $4c, $4a, $48, $45, $43, $40,
$3e, $3c, $39, $37, $34, $32, $30, $2d, $2b, $28, $26, $24
```

```
score2x:
.db $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b,
$8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84,
$82, $81, $80, $7f, $7e, $7c, $7b, $7a, $79, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $79, $7a, $7b, $7c,
$7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90

score2y:
.db $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $52, $51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44,
$43, $42, $40, $3f, $3e, $3d, $3c, $3c, $3b, $3a, $3a, $39, $39, $38, $37, $37, $36, $36,
$35, $34, $34, $33, $33, $32, $31, $31, $30, $30, $30, $2f, $2e, $2e, $2d, $2d, $2c, $2b,
$2b, $2a, $2a, $29, $28, $28, $27, $27, $26, $25, $25, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24

score3x:
.db $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b,
$8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84,
$82, $81, $80, $7f, $7e, $7c, $7b, $7a, $79, $78, $84, $84, $85, $85, $86, $87, $87, $88,
$88, $89, $8a, $8a, $8b, $8b, $8c, $8d, $8d, $8e, $8e, $8f, $90

score3y:
.db $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $51, $4f, $4c, $4a, $48, $45, $43, $40, $3e, $3c, $39, $37, $34,
$32, $30, $2d, $2b, $28, $26, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c

score4x:
.db $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84, $82, $81, $80,
$7f, $7e, $7c, $7b, $7a, $79, $78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84,
$85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90

score4y:
.db $24, $26, $28, $2b, $2d, $30, $32, $34, $37, $39, $3c, $3e, $40, $43, $45, $48, $4a,
$4c, $4f, $51, $54, $54, $52, $51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44,
$43, $42, $40, $3f, $3e, $3d, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c

score5x:
.db $8c, $8a, $89, $88, $87, $86, $84, $83, $82, $81, $80, $7e, $7d, $7c, $7b, $7a, $78,
$77, $76, $75, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74,
$74, $74, $74, $74, $74, $74, $74, $74, $74, $75, $75, $76, $77, $77, $78, $78, $79, $7a,
$7a, $7b, $7b, $7c, $7d, $7d, $7e, $7e, $7f, $80, $80, $80, $81, $81, $82, $83, $83, $84,
$84, $85, $86, $86, $87, $87, $88, $89, $89, $8a, $8a, $8b, $8c, $8c, $8a, $89, $88, $87,
$86, $84, $83, $82, $81, $80, $7e, $7d, $7c, $7b, $7a, $78, $77, $76, $75, $74

score5y:
.db $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $52, $51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44,
$43, $42, $40, $3f, $3e, $3d, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3a, $39, $38, $37, $36, $34, $33,
$32, $31, $30, $2e, $2d, $2c, $2b, $2a, $28, $27, $26, $25, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24

score6x:
.db $84, $83, $82, $82, $81, $81, $80, $7f, $7f, $7e, $7e, $7d, $7c, $7c, $7b, $7b, $7a,
$79, $79, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84,
$85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $8e, $8d, $8c, $8b,
$8a, $88, $87, $86, $85, $84, $82, $81, $80, $7f, $7e, $7c, $7b, $7a, $79, $78

score6y:
.db $54, $52, $51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44, $43, $42, $40,
$3f, $3e, $3d, $3c, $3c, $3a, $39, $38, $37, $36, $34, $33, $32, $31, $30, $2e, $2d, $2c,
$2b, $2a, $28, $27, $26, $25, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24,
```

```
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $25, $26, $27, $28, $2a, $2b, $2c,
$2d, $2e, $30, $31, $32, $33, $34, $36, $37, $38, $39, $3a, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c

score7x:
.db $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84, $85, $86, $87, $88, $8a, $8b,
$8c, $8d, $8e, $90, $90, $8f, $8e, $8e, $8d, $8d, $8c, $8b, $8b, $8a, $8a, $89, $88, $88,
$87, $87, $86, $85, $85, $84, $84

score7y:
.db $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $51, $4f, $4c, $4a, $48, $45, $43, $40, $3e, $3c, $39, $37, $34,
$32, $30, $2d, $2b, $28, $26, $24

score8x:
.db $90, $8e, $8d, $8c, $8b, $8a, $88, $87, $86, $85, $84, $82, $81, $80, $7f, $7e, $7c,
$7b, $7a, $79, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78, $78,
$78, $78, $78, $78, $78, $78, $78, $78, $79, $7a, $7b, $7c, $7e, $7f, $80, $81, $82, $84,
$85, $86, $87, $88, $8a, $8b, $8c, $8d, $8e, $90, $90, $90, $90, $90, $90, $90, $90, $90,
$90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $90, $8f, $8e, $8e, $8d,
$8d, $8c, $8b, $8b, $8a, $8a, $89, $88, $88, $87, $87, $86, $85, $85, $84, $84, $84, $84,
$84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84, $84

score8y:
.db $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3a, $39, $38, $37, $36, $34, $33, $32, $31, $30, $2e, $2d, $2c,
$2b, $2a, $28, $27, $26, $25, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $24,
$24, $24, $24, $24, $24, $24, $24, $24, $24, $24, $26, $28, $2b, $2d, $30, $32, $34,
$37, $39, $3c, $3e, $40, $43, $45, $48, $4a, $4c, $4f, $51, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $52,
$51, $50, $4f, $4e, $4c, $4b, $4a, $49, $48, $46, $45, $44, $43, $42, $40, $3f, $3e, $3d, $3c

score9x:
.db $8c, $8a, $89, $88, $87, $86, $84, $83, $82, $81, $80, $7e, $7d, $7c, $7b, $7a, $78,
$77, $76, $75, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74, $74,
$74, $74, $74, $74, $74, $74, $74, $74, $75, $76, $77, $78, $7a, $7b, $7c, $7d, $7e, $80,
$81, $82, $83, $84, $86, $87, $88, $89, $8a, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c,
$8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8c, $8b, $8a, $8a, $89,
$89, $88, $87, $87, $86, $86, $85, $84, $84, $83, $83, $82, $81, $81, $80, $80

score9y:
.db $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c, $3c,
$3c, $3c, $3c, $3c, $3c, $3d, $3e, $3f, $40, $42, $43, $44, $45, $46, $48, $49, $4a, $4b,
$4c, $4e, $4f, $50, $51, $52, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $54,
$54, $54, $54, $54, $54, $54, $54, $54, $54, $54, $52, $51, $50, $4f, $4e, $4c, $4b,
$4a, $49, $48, $46, $45, $44, $43, $42, $40, $3f, $3e, $3d, $3c, $3c, $3a, $39, $38, $37,
$36, $34, $33, $32, $31, $30, $2e, $2d, $2c, $2b, $2a, $28, $27, $26, $25, $24

/*

Code for the TWI (TwoWire Interface) I2C connection with the Nunchuck [Incomplete/Not
working]

; ---------------------------------------------------------------------
;                         VARIABLES
; ---------------------------------------------------------------------

; Decoder Key
.equ    DECODER_KEY = $17                               ; Value used to decode Nunchuck data
| decoded = ( encoded ^ $17) + $17

; TWI Slave Write/Read Command
.equ    SLA_W = $A4                                     ; bit7:1 = SLave Address (Nintendo
Nunchuck), bit0 = Write | 0b10100100
.equ    SLA_R = $A5                                     ; bit7:1 = SLave Address (Nintendo
Nunchuck), bit0 = Read  | 0b10100101

; TWStatusRegister status codes for MasterReceiver mode:
.equ    START = $08                                     ; A START condition has been
transmitted
.equ    RSTART = $10                                    ; A repeated START condition has
```

```
been transmitted
.equ    ARB_LOST = $38                                  ; Arbitration lost in SLA+R or
NOTACK bit
.equ    MT_SLA_ACK = $18                                ; SLA+W has been transmitted, ACK
has been received
.equ    MR_SLA_ACK = $40                                ; SLA+R has been transmitted, ACK
has been received
.equ    MR_SLA_NACK = $48                               ; SLA+R has been transmitted, NOT
ACK has been received
.equ    MT_DATA_ACK = $28                               ; Data byte has been sent, ACK has
been received
.equ    MR_DATA_ACK = $50                               ; Data byte has been received, ACK
has been returned
.equ    MR_DATA_NACK = $58                              ; Data byte has been received; NOT
ACK has been returned

; def register we want to use to dump data from TWDR
.def    DATAREGISTER = r24

; SRAM address of start of buffer for Nunchuck data
.equ    BUFFER_START = $0100

; Addresses in SRAM where to Nunchuck data will be found
.equ    JOYSTICK_X = $0100                              ; X-axis data of the joystick
.equ    JOYSTICK_Y = $0101                              ; Y-axis data of the joystick
.equ    ACCELEROMETER_X = $0102                         ; X-axis data of the accellerometer
sensor (bits 9 to 2 for 10bit precision)
.equ    ACCELEROMETER_Y = $0103                         ; Y-axis data of the accellerometer
sensor (bits 9 to 2 for 10bit precision)
.equ    ACCELEROMETER_Z = $0104                         ; Z-axis data of the accellerometer
sensor (bits 9 to 2 for 10bit precision)
.equ    MISC = $0105                                    ; bit0 as Z button status - 0 =
pressed and 1 = release
                                                        ; bit1 as C button status - 0 =
                                                        pressed and 1 = release
                                                        ; bit2-3 : 2 lower bit of X-axis
                                                        data of the accellerometer sensor
                                                        ; bit4-5 : 2 lower bit of Y-axis
                                                        data of the accellerometer sensor
                                                        ; bit6-7 : 2 lower bit of Z-axis
                                                        data of the accellerometer sensor

; ---------------------------------------------------------------------
;                       INITIALISATION
; ---------------------------------------------------------------------

; I2C Bus Frequency - 100kHz - (for 400 kHz choose TWBR=0, TWPS=1 and change value of
pull-up resistors)
    ;ldi        r16, (1<<TWBR6)                          ; loads register with the
    division factor for the bit rate generation (32) ...
    ;sts        TWBR, r16                                ; ... and sends it to the TWI
    Bit Rate register
    ;ldi        r16, (0<<TWPS1)|(1<<TWPS0)               ; loads register with the bit
    rate pre-scaler value (1) ...
    ;sts        TWSR, r16                                ; and sends it to the TWI Status
    Register

; ---------------------------------------------------------------------
;                       INITIALISATION
; ---------------------------------------------------------------------

Main:

    rcall i2c_start                                     ; start TWI(I2C) connection
    ;rcall nunchuck_handshake                           ; initialise Nunchuck, just do it.
    ;rcall i2c_stop

    ;rcall i2c_start
    ;rcall nunchuck_get_data
    ;rcall i2c_stop
    ;nop
```

```
    ;rcall i2c_rstart
    ;rcall nunchuck_pack                                    ; pack sensor data in
    Nunchuck                                ; stop TWI(I2C) connection  (removable?)
    ;rcall i2c_start
    ;rcall nunchuck_get_data                                ; get all data from nunchuck
    and save in SRAM
    ;rcall i2c_stop                                         ; stop TWI(I2C) connection
    (removable?)

loop:
    ;rcall i2c_start
    ;rcall nunchuck_get_data
    ;rcall i2c_stop
    ;rcall nunchuck_z_button
    ;out PORTB, TEMPREG
    ;rcall i2c_start
    ;rcall BIGDEL
    ;rcall nunchuck_get_data
    ;rcall BIGDEL
    ;rcall i2c_stop
    ;rcall BIGDEL
    ;rcall nunchuck_z_button
    ;out PORTB, TEMPREG
    ;rcall BIGDEL
    ;ldi    TEMPREG, 0b11001100
    ;out PORTB, TEMPREG
    ;rcall BIGDEL
    ;rcall BIGDEL
    ;ldi    TEMPREG, 0b00110011
    ;out PORTB, TEMPREG
    ;rcall i2c_start                                        ; start TWI(I2C) connection
    (removable?)
    ;rcall nunchuck_get_data                                ; get all data from nunchuck
    and save in SRAM
    ;rcall i2c_stop                                         ; stop TWI(I2C) connection

    ;rcall nunchuck_z_button                                ; get status of Z button
    from memory (0 pressed / 1 not pressed)
    ;out PORTB, TEMPREG                                     ; display it on PORTB LEDs

    rjmp Main

;
================================================================================================
================================
;                                                           I2C ROUTINES
;
================================================================================================
================================


i2c_start:                                                  ; START TWI(I2C) CONNECTION:
    ldi     TEMPREG,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)         ; Set-up START: clears TWINT
    flag ¦ sets START ¦ turns on TWI ¦ takes control of PinD0 (SCL) and PinD1 (SDA)
    sts     TWCR, TEMPREG                                   ; Send command to Two-Wire
    interface Control Register
    rcall   wait_for_TWINT_flag                             ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                                    ; Check value of TWI Status
    Register
    out     PORTB, TEMPREG
    andi    TEMPREG, 0xF8                                   ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, START                                  ; If status different from START
    go to ERROR ...
    ;brne   TWSR_error                                      ; ... else continue
    ret

nunchuck_handshake:                                         ; NINTENDO HANDSHAKE, FOLLOW
INSTRUCTIONS:
    ldi     TEMPREG, SLA_W                                  ; Load SLA_W (SLave Address +
```

```
    Write) byte ...
    sts     TWDR, TEMPREG                           ; ... into TWDataRegister
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)         ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                           ; ... to start transmission of
    SLA_W on bus
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                           ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MT_SLA_ACK                     ; If status different from
    MT_SLA_ACK go to ERROR
    brne    TWSR_error                              ; ... else continue

    ldi     TEMPREG, $40                            ; Load $40 byte ...
    sts     TWDR, TEMPREG                           ; ... into TWDataRegister
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)         ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                           ; ... to start transmission of
    byte on bus
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                           ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MT_DATA_ACK                    ; If status different from
    MT_DATA_ACK go to ERROR
    brne    TWSR_error                              ; ... else continue

    ldi     TEMPREG, $00                            ; Load $00 byte ...
    sts     TWDR, TEMPREG                           ; ... into TWDataRegister
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)         ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                           ; ... to start transmission of
    byte on bus
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                           ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MT_DATA_ACK                    ; If status different from
    MT_DATA_ACK go to ERROR
    brne    TWSR_error                              ; ... else continue

    ret

nunchuck_pack:                                      ; TELL NUNCHUK TO STORE ALL
SENSOR DATA IN ITS 6BYTE REGISTER
    ldi     TEMPREG, SLA_W                          ; Load SLA_W (SLave Address +
    Write) byte ...
    sts     TWDR, TEMPREG                           ; ... into TWDataRegister
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)         ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                           ; ... to start transmission of
    SLA_W on bus
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                           ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MT_SLA_ACK                     ; If status different from
    MT_SLA_ACK go to ERROR
    brne    TWSR_error

    ldi     TEMPREG, $00                            ; Load $00 byte ...
    sts     TWDR, TEMPREG                           ; ... into TWDataRegister
```

```
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)                 ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                                   ; ... to start transmission of
    byte on bus
    rcall   wait_for_TWINT_flag                             ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                                    ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                                   ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MT_DATA_ACK                            ; If status different from
    MT_DATA_ACK go to ERROR
    brne    TWSR_error                                      ; ... else continue

    ret

TWSR_error:                                                 ; Routine that handles error if
TWStatusRegister does not have correct status ...
    out     PORTB, TEMPREG                                  ; display status on LEDs (for
    debugging)
    ret                                                     ; (do something useful later on
    when everything else works)

nunchuck_get_data:                                          ; ENTER MASTER RECEIVER MODE,
SET SLAVE, AND GET DATA:
    ldi     TEMPREG, SLA_R                                  ; Load SLA_R (SLave Address +
    Read) byte ...
    sts     TWDR, TEMPREG                                   ; ... into TWDataRegister
    ldi     TEMPREG, (1<<TWINT) | (1<<TWEN)                 ; Now we can clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                                   ; ... to start transmission of
    SLA_R on bus
    rcall   wait_for_TWINT_flag                             ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                                    ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                                   ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MR_SLA_ACK                             ; If status different from
    MR_SLA_ACK go to ERROR
    brne    TWSR_error                                      ; ... else continue

    ldi     ZL, low(BUFFER_START)                           ; Loads pointer to BUFFER_START
    in SRAM ...
    ldi     ZH, high(BUFFER_START)                          ; ... in ZH:ZL registers
    rcall   nunchuck_get_byte                               ; get and save JOYSTICK_X value
    from NUNCHUCK
    rcall   nunchuck_get_byte                               ; get and save JOYSTICK_Y value
    from NUNCHUCK
    rcall   nunchuck_get_byte                               ; get and save ACCELEROMETER_X
    value from NUNCHUCK
    rcall   nunchuck_get_byte                               ; get and save ACCELEROMETER_Y
    value from NUNCHUCK
    rcall   nunchuck_get_byte                               ; get and save ACCELEROMETER_Z
    value from NUNCHUCK
    rcall   nunchuk_get_last_byte                           ; get and save MISC value
    (buttons) from NUNCHUCK

    ret

nunchuck_get_byte:                                          ; READ BYTE
    ldi     TEMPREG, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)         ; Clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                                   ; ... to receive byte from slave
    (will also send ACK)
    rcall   wait_for_TWINT_flag                             ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG, TWSR                                   ; Check value of TWI Status
    Register:
    andi    TEMPREG, 0xF8                                   ; mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
```

```
    cpi     TEMPREG, MR_DATA_ACK                    ; if status different from
    MR_DATA_ACK go to ERROR
    brne    TWSR_error                              ; ... else continue
    lds     DATAREGISTER, TWDR                      ; load slave data from
    TWDataRegister
    rcall   nunchuck_decode                         ; decodes slave data (Nunchuck
    sends encrypted data)
    st      Z+,     DATAREGISTER                    ; save data to SRAM and post
    increments pointer to SRAM
    ret

nunchuk_get_last_byte:                              ; READ LAST BYTE
    ldi     TEMPREG, (1<<TWINT)|(1<<TWEN)           ; clear TWINT flag in
    TWControlRegister ...
    sts     TWCR, TEMPREG                           ; ... to receive next byte from
    slave (will send NACK)
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register:
    andi    TEMPREG, 0xF8                           ; mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, MR_DATA_NACK                   ; if status different from
    MR_DATA_NACK go to ERROR
    brne    TWSR_error                              ; ... else continue
    lds     DATAREGISTER, TWDR                      ; Load slave data from
    TWDataRegister
    rcall   nunchuck_decode                         ; decodes slave data (Nunchuck
    sends encrypted data)
    st      Z, DATAREGISTER                         ; save data to SRAM
    ret

i2c_rstart:                                         ; START TWI(I2C) CONNECTION:
    ldi     TEMPREG,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN) ; Set-up START: clears TWINT
    flag ¦ sets START ¦ turns on TWI ¦ takes control of PinD0 (SCL) and PinD1 (SDA)
    sts     TWCR, TEMPREG                           ; Send command to Two-Wire
    interface Control Register
    rcall   wait_for_TWINT_flag                     ; wait for TWINT flag in TWCR to
    be set
    lds     TEMPREG,TWSR                            ; Check value of TWI Status
    Register
    andi    TEMPREG, 0xF8                           ; Mask reserved bit (b2) and
    pre-scaler bits (b1,b0)
    cpi     TEMPREG, RSTART                         ; If status different from START
    go to ERROR ...
    brne    TWSR_error                              ; ... else continue
    ret

i2c_stop:                                           ; STOP: (STOP does not set TWINT
flag so no need to add a wait4 block)
    ldi     TEMPREG,(1<<TWINT)|(1<<TWEN)|(1<<TWSTO) ; Set-up STOP: clears TWINT flag
    ¦ sets STOP ¦ turns on TWI
    sts     TWCR, TEMPREG                           ; Send command to
    TWControlRegister
wait_for_i2c_stop:
    lds     TEMPREG,TWCR                            ; read in TWControlRegister
    sbrc    TEMPREG,TWSTO                           ; if TWSTO flag is clear (stop
    has been executed) skip next instruction ...
    rjmp    wait_for_i2c_stop
    ret

wait_for_TWINT_flag:                                ; WAITS FOR TWINT FLAG TO BE SET:
    lds     TEMPREG,TWCR                            ; read in TWControlRegister
    sbrs    TEMPREG,TWINT                           ; if TWINT flag is set (command
    has been executed) skip next instruction ...
    rjmp    wait_for_TWINT_flag                     ; ... else wait until it's set
    ret

nunchuck_decode:                                    ; Routine that decodes Nunchuck
encrypted data --> decoded = ( encoded XOR $17) + $17
    ldi     TEMPREG, DECODER_KEY                    ; loads TEMPorary REGISTER with
```

```
    the Nintendo decoding key $17
    eor     DATAREGISTER, TEMPREG                   ; XORs Nunchuck data byte with $17
    adc     DATAREGISTER, TEMPREG                   ; adds $17 to XORed Nunchuck data
    ret                                             ; returns with decoded data in
    DATAREGISTER

nunchuck_joy_x:                                     ; ACCESS LAST SAVED VALUE OF
JOYSTICK_X
    lds     TEMPREG, JOYSTICK_X
    ret

nunchuck_joy_y:                                     ; ACCESS LAST SAVED VALUE OF
JOYSTICK_Y
    lds     TEMPREG, JOYSTICK_Y
    ret

nunchuck_accel_x:                                   ; ACCESS LAST SAVED VALUE OF
ACCELEROMETER_X
    lds     TEMPREG, ACCELEROMETER_X
    ret

nunchuck_accel_y:                                   ; ACCESS LAST SAVED VALUE OF
ACCELEROMETER_Y
    lds     TEMPREG, ACCELEROMETER_Y
    ret

nunchuck_accel_z:                                   ; ACCESS LAST SAVED VALUE OF
ACCELEROMETER_Z
    lds     TEMPREG, ACCELEROMETER_Z
    ret

nunchuck_z_button:                                  ; ACCESS LAST SAVED STATUS OF Z
BUTTON
    lds     TEMPREG, MISC
    andi    TEMPREG, 0b00000001                     ; masks all bits except Z BUTTON
    status
    ret

nunchuck_c_button:                                  ; ACCESS LAST SAVED STATUS OF C
BUTTON
    lds     TEMPREG, MISC
    andi    TEMPREG, 0b00000010                     ; masks all bits except C BUTTON
    status
    lsr     TEMPREG                                 ; shifts right to return 0 or 1
    ret

*/
```