The Microprocessor Course
Constructing an electronic chess game

Hugo Jeanperrin

04.03.15

Abstract

A player versus player chess game was coded on an ATMega128 board using AVR Assembly. Each player in turn moves a piece by entering coordinates on a 4×4 keypad, prompted by messages on a Hitachi HD44780U (4×20 alphanumeric). Move algorithms then verify the validity of the square selected and that of the destination square. The game was displayed on a 160×104 pixels graphic display (EA DOGXL 160-7), using a four-wire SPI interface.

1 Introduction

Chess is famed for its simple, deterministic rules, but also for its difficulty. One only needs to pick up a book in chess theory to appreciate the plethora of strategies and techniques available to the player. It is this duality which has made transferring the game to an electronic format so popular.

In 1997, IBM famously developed *Deep Blue*, a chess playing computer, which defeated Garry Kasparov, the then reigning world champion. More recently, Olivier Poudade coded BootChess in Assembly. This chess simulation is the smallest-sized chess computer program, taking up only 487 bytes in size [1], and was the inspiration behind this project.

The aim of this project was to code a player versus player chess game on the ATMega128 microprocessing board using AVR Assembly and to display the game on a graphic *Liquid Crystal Display* (LCD). For this project, a *Film-compensated Super Twisted Nematic* (FSTN) display was chosen: the EA-DOGXL160S-7.

A liquid crystal exhibits properties halfway between a crystal and a liquid. The molecules present can flow like a liquid, but they are aligned like they would be in a crystal. A liquid crystal has different phases, where the position and the orientation of the molecules differ from one to the next. In the nematic phase, the molecules are oriented along the same axis, but their centre of mass can flow freely. They can be controlled by applying an external electric or magnetic field.

Figure 1 illustrates the two possible states of a pixel. The liquid crystal is inserted between two plates each containing an electrode ($E_{1/2}$ on the Figure), a glass plate ($G_{1/2}$) and a

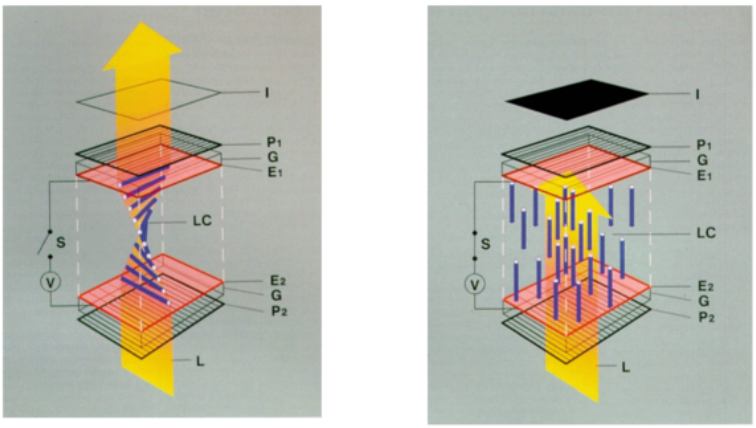


Figure 1: Schematic of the Twisted Nematic field effect. The schematic on the left showcases the helix-like alignment when no voltage is applied. On the right, the molecular structure is blown up when a voltage is applied (image from wikipedia.org).

linear polariser ($P_{1/2}$). The crystal is twisted by 90° and forms a helix-like structure when no voltage is applied to the electrodes. As may be seen in the Figure, light is polarised linearly and is rotated by 90° by the molecules and can therefore pass through the second polariser unimpeded. The pixel then appears transparent. On the other hand, when a voltage is applied to the electrodes, the structure breaks up and the molecules align along the electric field. Therefore the light polarised by P_1 is no longer rotated and is blocked by P_2. As a consequence, the pixel appears opaque.

The discovery of the *Twisted Nematic* (TN) field effect made LCDs practical for the first time. A twisted nematic has the crucial advantage over a normal nematic that it does not require current to flow to influence the liquid crystal. Applying a voltage (typically a few volts) is sufficient to operate the display.

Further improvements were made by *Super Twisted Nematic* displays, improving the contrast ratio and the viewing angle of the displays [2]. These were first built in 1984 and are called "super twisted" because the layer twist angle is three times larger than a traditional TN (270°). Contrast was improved further by inserting film compensators behind the STN display.

The aims of this project were to create a player versus player chess game displayed on a FSTN LCD. Some of the planned features were to enable the user to load the previous game or start a new one, to select and move players according to basic chess rules, and to win the game by taking the opposition king. The graphic LCD should display a chessboard and its coordinates and correctly display individual piece moves on the board. Other propositions included the coding of an interactive selector that uses a pad as an arrow board and a basic AI opponent (eg. using random moves).

2 High level design

2.1 Software Overview

The overall software structure is showcased in Figure 2. At the start of the session, the alphanumeric screen is initialised and the player is prompted to either load a game or start a new one. The program then loads all the pieces in the SRAM in either the starting position or the previously saved one. At this stage the graphic LCD is initialised and draws the board, the coordinates and the pieces to the screen.

A user input routine asks the user to enter the piece that he/she wishes to select and input a destination square. This is done by sending messages on the alphanumeric screen and giving in information using the switches 0-4 on PORTD and a 4×4 keypad as described in Section 2.2.

To avoid completing illegal moves, a Move Verification Algorithm is necessary. It checks if:

- There is a piece present in the selected square and if it is of the right colour
- There is a piece in the destination square and if it can be eaten or not
- The type of piece selected is allowed to perform this particular move. The program points to to a verification subroutine dedicated to this specific piece (eg. a pawn is not allowed to move backwards)
- There are other pieces on the way which render the move impossible. Only knights are capable of jumping over opponents.

As long as the move is invalid, the user is returned to the user input routine, until the move is accepted. A valid move will be drawn on the graphic LCD screen using an output function. The coordinates of the piece are stored and mapped to the corresponding screen position. Upon completion, the player changes and the program returns to the user input routine for the next move.

Note that the user can exit from the program at any time by pressing the exit button on Port D. The game may then be saved and the user can choose to play a demo routine, where the screen loops a ten-move chess game.

For a more detailed version of this flowchart see the Appendix (Figure 14).

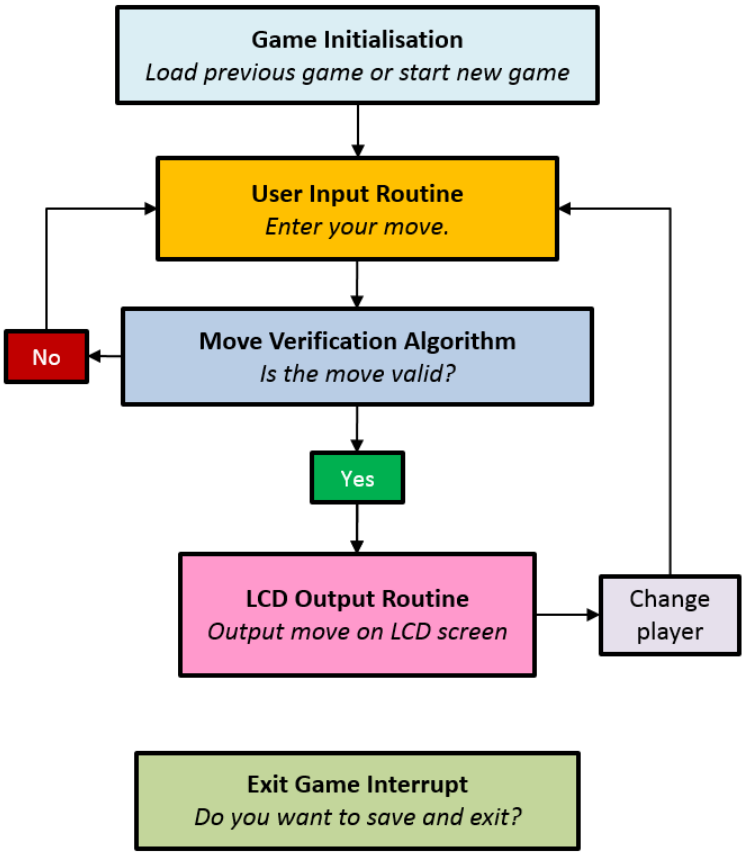


Figure 2: Flowchart of the overall software structure of the project.

2.2 Hardware layout

The hardware layout is illustrated by Figure 3. The program is compiled from AVR Assembly to a .hex file, which is sent to the board via the programming interface [3]. The board itself is powered by a 9-15 V DC supply. User input is executed by entering a coordinate on the keypad, connected to the board on Port E, and the Port D switches (0-4), which let the user accept or reject a command, erase a coordinate or quit the game. The same process was used to decode the keypad as in the lectures [4].

User prompts are displayed on the Hitachi HD44780U LCD, which is driven by the board

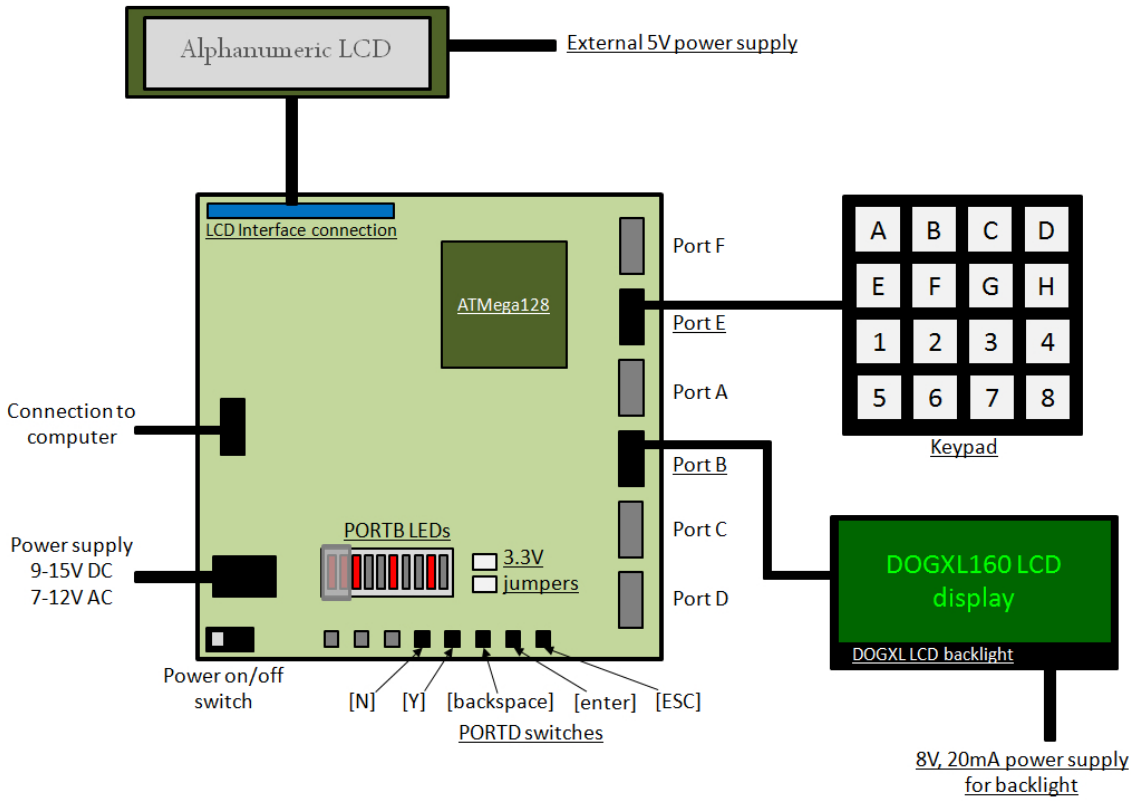


Figure 3: Diagram of the hardware setup of the project, including the ATMega128 Microprocessor board, the Alphanumeric LCD, the Graphic LCD and the 4×4 keypad.

via the LCD interface connection. The board can access the LCD RAM directly as it is detected as external RAM by the microprocessor [5].

Game outputs are sent through Port B to the DOGXL160S-7 LCD screen using a four-wire SPI interface. The display is backlit by a green LED backlight (EA LED78x64-G) connected to an external power supply.

It must be noted that the DOGXL160 LCD operates exclusively at 3.3 V. Therefore, the whole board is run at 3.3 V. This is done by removing the jumpers indicated in the diagram next to the Port B LEDs. However, the alphanumeric screen is configured to function at 5 V, and it was found that the contrast at 3.3 V was insufficient. This was corrected using an external power supply.

3 Software and Hardware design

3.1 Program Structure

The coding was separated into two major components, so that work could be done in parallel. Richard Flint wrote the rules of chess (Section 3.1.2) and I worked on the interface with the graphic LCD screen (Section 3.1.3). Therefore, this report will go into greater detail in the LCD section.

3.1.1 Initialisation sequence

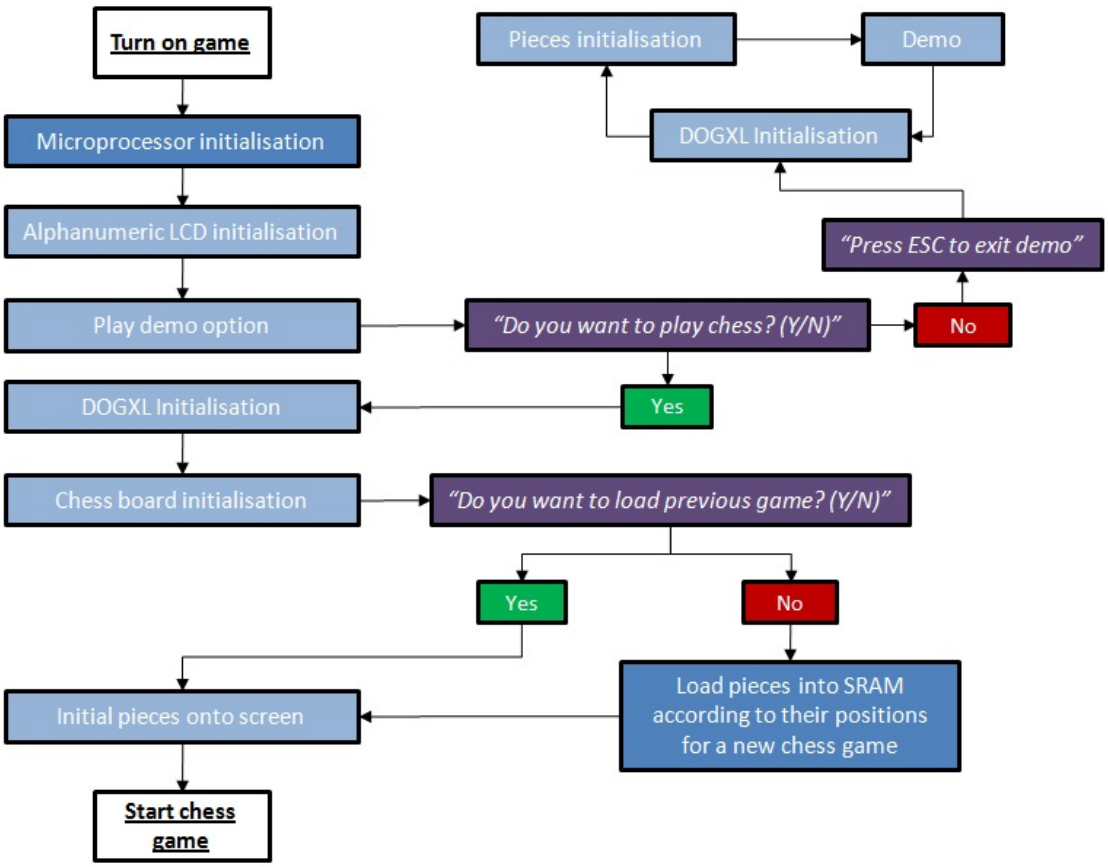


Figure 4: Flowchart of the initialisation sequence of the program.

The initialisation sequence is the first part of the program to be run and constitutes the highest level of the program. When system is turned on, both the ATMega128 and the Alphanumeric LCD screen are initialised. The user is then asked if he/she wants to play chess or run a demo option, which runs in a loop through a ten move game on the DOGXL LCD screen. This sequence can be exited at any time by pressing on the ESC switch, which triggers an interrupt. See section 3.1.3 for more information on the demo sequence.

If the user opts to play chess, pieces are loaded into the SRAM. The graphic display is then initialised the pieces loaded onto the screen. The user can then start playing chess.

3.1.2 The rules of chess

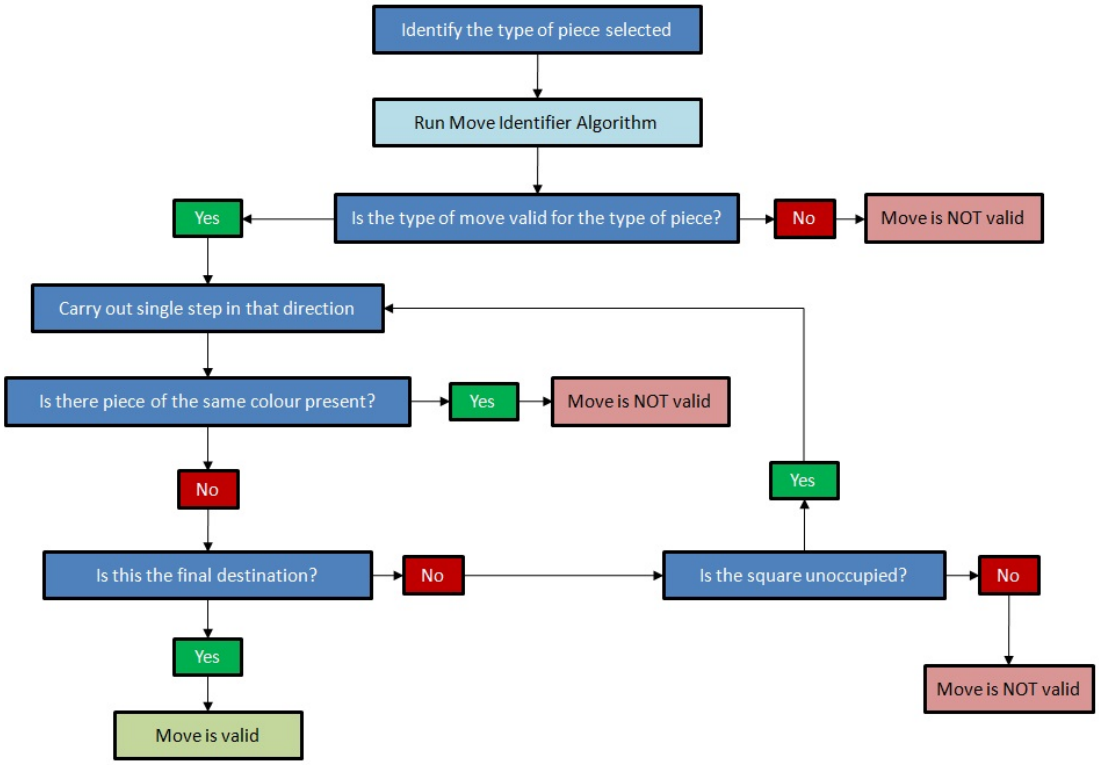


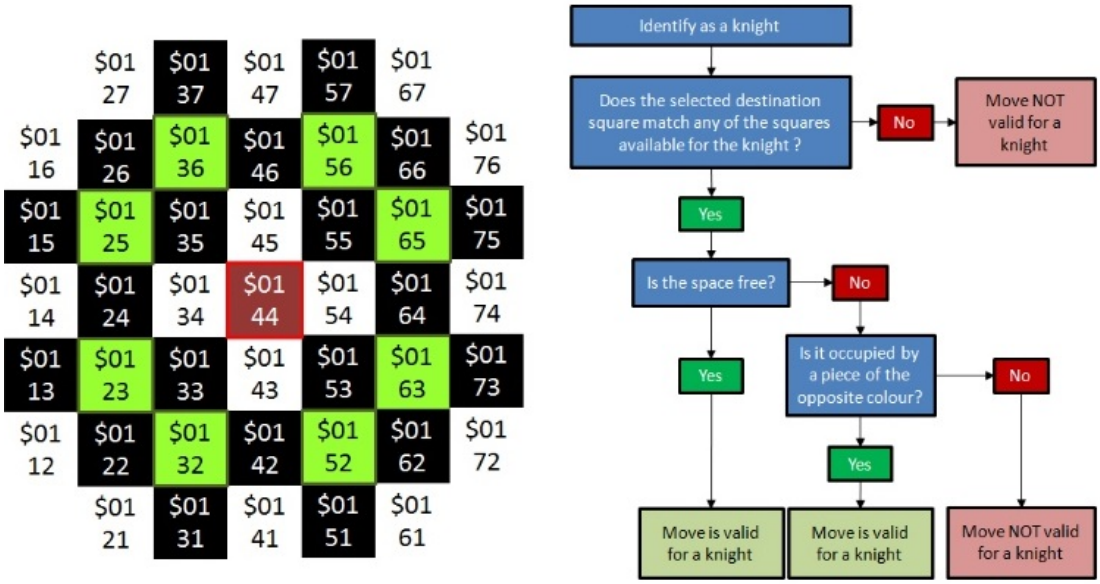
Figure 5: Flowchart of the Move Identifier algorithm to determine the validity of a move.

The general procedure to identify the validity of a move is illustrated in the flow diagram in Figure 5. Firstly, the type of move (eg. diagonal) is checked against the type of piece (eg. bishop). The algorithm then proceeds to carry out single steps in the direction of the destination square, checking at each step that the square doesn't contain a piece unless it is the final destination. If it is the final destination, it is checked whether the square is unoccupied or occupied by a piece of a different colour. Every time a move is declared illegal, the algorithm returns to the user input routine.

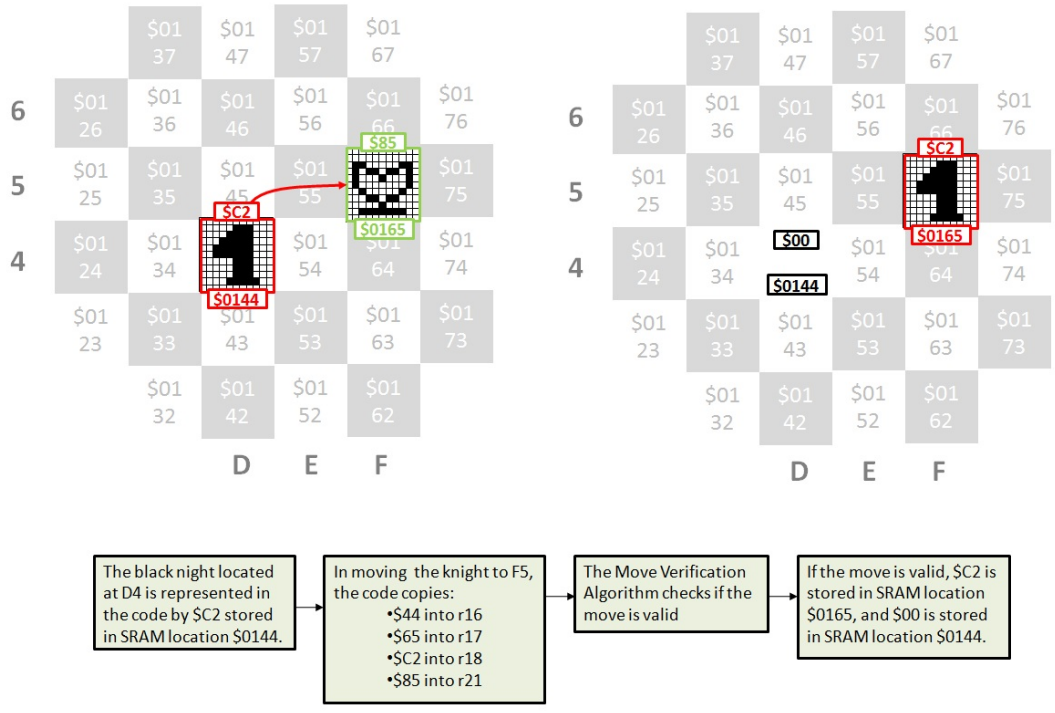
Each individual type has its own Move Identifier Algorithm since they obey different rules. This section will focus on the algorithm for the knight showcased in Figure 6. The algorithm for the knight is the simplest because of the limited number of destination squares and because occupancy of intermediate squares does not matter. The most complicated flowchart is the pawn, because of all the special cases: it can move two steps forward the first time it is moved and can move forward diagonally, but only if an opposing piece is present in the destination square. Flowcharts for the other pieces can be found in the Appendix (Figures 16-19).

The knight always moves in an L-shape, two steps in one direction and one in another. Therefore, there are eight possible destination squares for the knight, highlighted in green in Figure 6a. Moreover, the destination square must be either free or occupied by an opponent piece. If all these conditions are satisfied, the move can take place.

Figure 6b illustrates how the program interprets a knight eating a queen. When the user inputs the coordinates the corresponding position and type values of both the departure and



(a) First and second possible moves for the knight (in green and black respectively). The initial square is highlighted in red.



(b) How the program interprets a knight eating a queen.

Figure 6: Move verification algorithm for the knight.

destination squares are preloaded into the working memory. The algorithm from Figure 6a is then applied to determine if the move is valid. If it is, the code for an empty square ($00) is stored in the departure square, and the value for the black knight ($C2) is loaded to the destination. The value for the white queen ($85) is thereby overwritten. All the coordinate and piece codes can be found in the Appendix (Figures 15a and 15b).

3.1.3 LCD interface

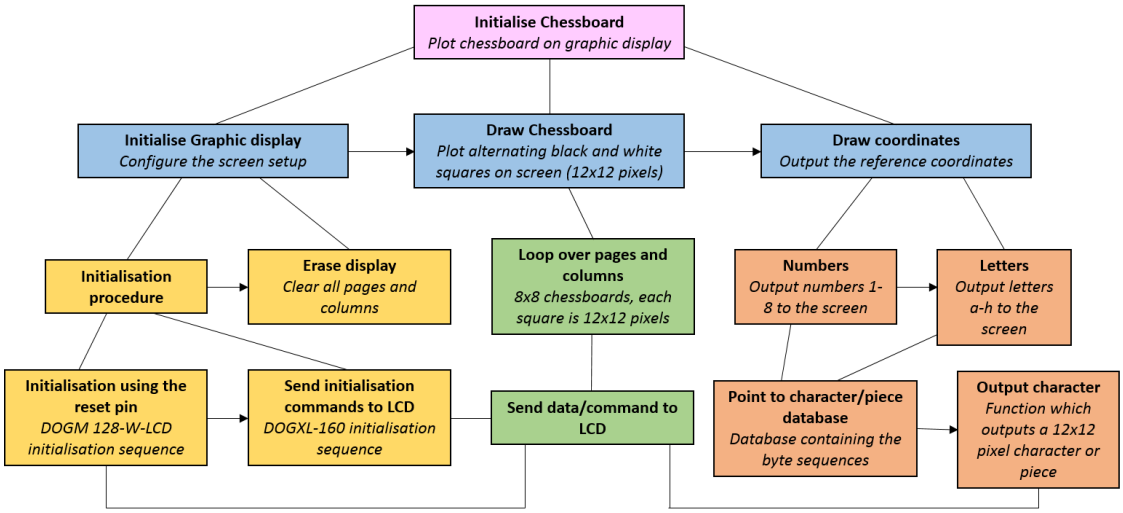


Figure 7: High level design of the DOGXL initialisation procedure. The graphic display initialisation was adapted from a similar function written for the ATMega8 [6].

The initialisation of the graphic screen is called from the main body of code and consists of three components. They are highlighted in blue in Figure 7. They initialise the screen (yellow) and draw the 8×8 chessboard (in green) with each square a block of 12×12 pixels^2. The functions highlighted in red plot the reference coordinates to the screen.

The initialisation procedure is a mixture of the initialisation commands given in the user manual [7] and the initialisation sequence given for another screen, the DOGM 128-W-LCD, using the reset pin. Once the initialisation is complete, the screen is erased by clearing all pages and columns on the LCD.

The chessboard is drawn at the centre of the screen by outputting light gray and dark gray squares to the screen in an alternating fashion. The program loops over the columns and rows and changes the colour when the twelfth column or page is reached. Finally, the reference coordinates are plotted by referencing a database (characters_and_pieces.asm), which contains the pixel mapping for numbers 1-8, letters a-h, and all the pieces needed to output to the screen, as illustrated by Figure 9.

Note that nearly all functions refer to the function "Send data/command to LCD", the flowchart of which is in Figure 8. The DOGXL has a size of 160×104 pixels^2 and sends information byte by byte to the UC1610 controller [8]. Since each pixel can take four different shades (white, light grey, dark gray and black), it needs two bits of information to define its state. Consequently, a byte sent to the LCD defines the state of four pixels at once, along four rows (referred to in this report and the documentation as a page) and one column. The screen has 160 columns and 26 pages in the active viewing area.

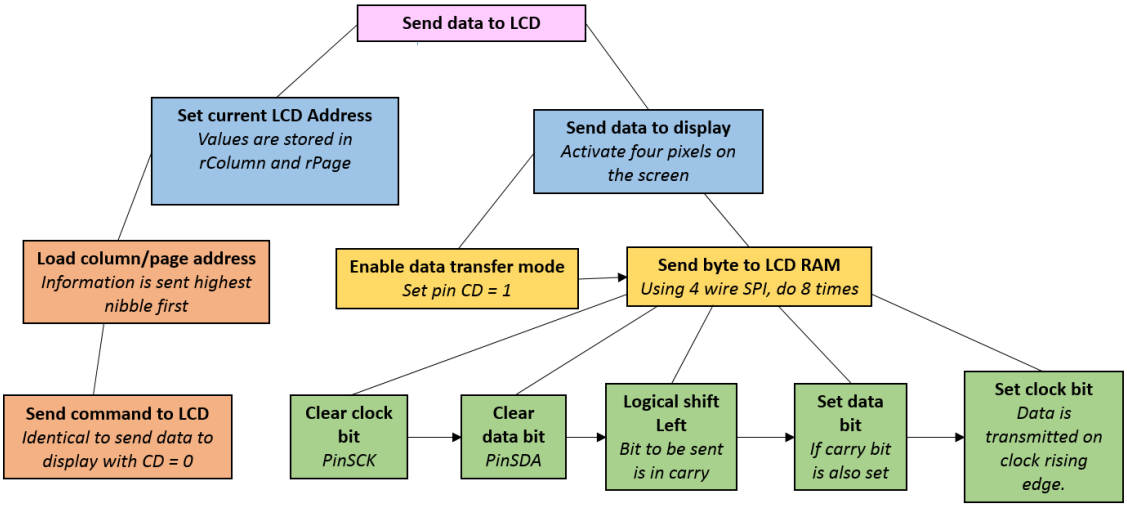


Figure 8: High level design of the function sending data through to the LCD. To send a command, the same procedure is used with PinCD = 0. An 8-bit, four wire SPI interface was used.

Therefore, one must define the current LCD address (a page and a column) to send data to the LCD screen. The column and page are sent to the screen most significant nibble first using the command function.

Sending data to the display is achieved using the four-wire SPI mode going through the Pins on Port B. Four pins are defined in the code to operate in this mode: PinSCK, PinSDA, PinCD and PinCS. They represent the clock, data, command and chip select pins respectively. The byte to be sent is loaded into a register in memory and left-shifted eight times, where PinSDA is set if the carry bit is set and the data is transmitted on the clock rising edge.

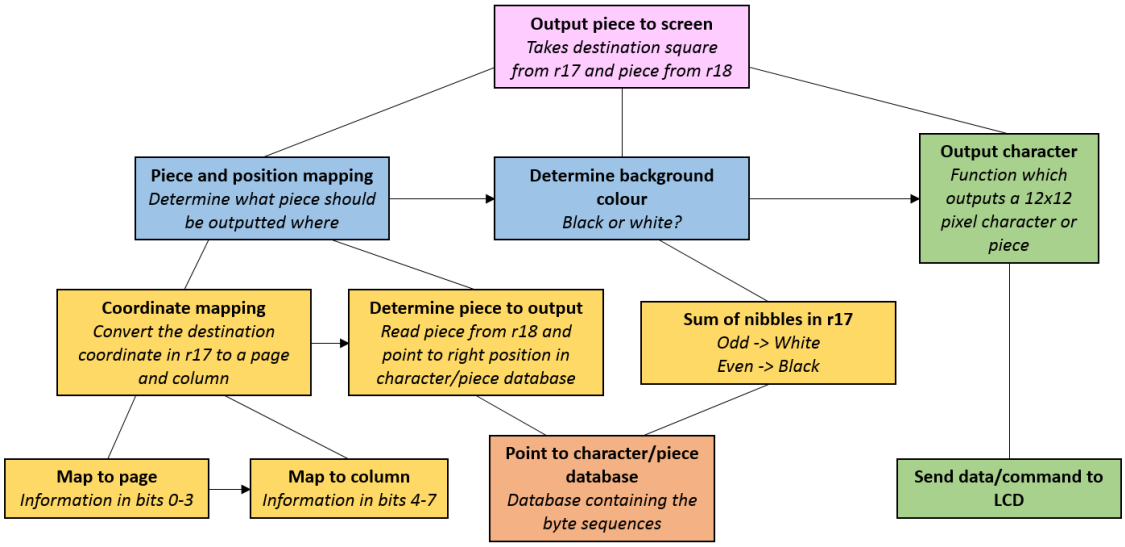


Figure 9: Flowchart of the function called from the main code, which draws the required piece onto the screen.

The LCD interface links to the main program by giving a function which takes the coordinate and value of a piece in the same format as discussed in Section 3.1.2 and showcased in the Appendix (Figures 15a and 15b) as an input. The coordinates given are mapped to the corresponding page and column on the screen. The background colour is determined by checking if the sum of the nibbles in the (unmapped) coordinate register is odd or even, which avoids having to do a direct mapping for all 64 squares. For example, the value received for C1 would be $31 (see Figure 15a). The sum of the two nibbles is $4 which is an even number, indicating that the square is black.

This information is combined with the piece value to point to the right piece in the character and piece database. There are 24 pieces available - the six pieces in black and white and with black and white backgrounds. They each define 144 pixels, which corresponds to 36 byte sequences that must be sent to the LCD screen.

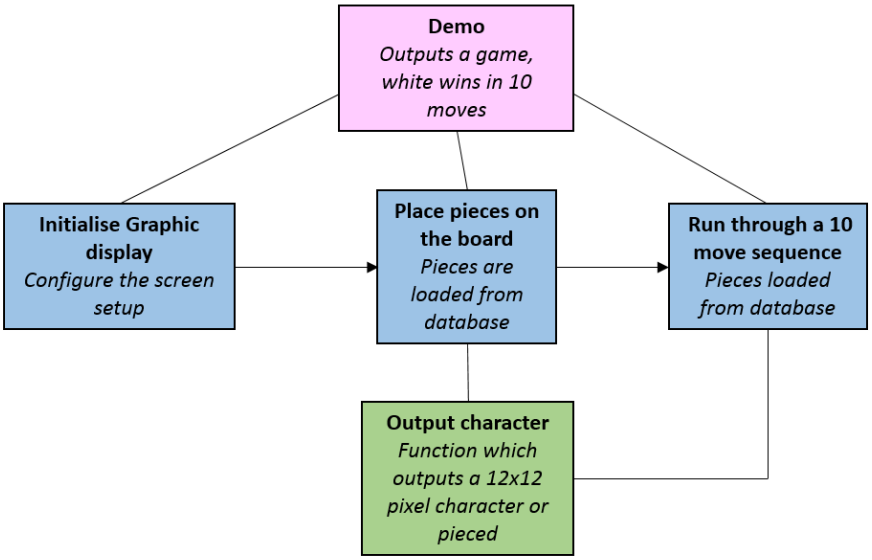


Figure 10: Flowchart of the ten move demonstration, which runs in a loop when a player versus player game isn't being played.

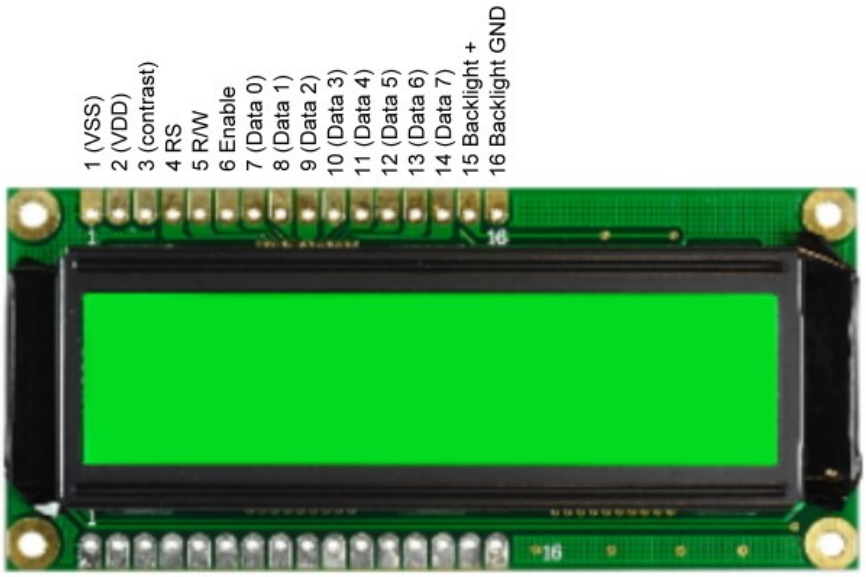
Finally, Figure 10 showcases the demo which runs from the main program if the user opts not to play chess. The function defines the coordinates directly and runs through a ten move sequence after which the black king is in check mate. The function points to the same database as mentioned previously.

3.2 Hardware design

3.2.1 Player input

The player input was achieved using the Hitachi HD44780U alphanumeric screen as a text prompt and the 4×4 keypad introduced in the lectures as an input for the player. Both were used in the same way as in the lectures [5] [6], with slight modifications.

The keypad was modified to show letters a-h and numbers 1-8, as illustrated by Figure 11b. With respect to the screen (Figure 11a), the contrast had to be changed because the screen normally runs on 5 V and the operating voltage had been changed to 3.3 V.



(a) Picture of the Hitachi HD44780U (4×20 alphanumeric) screen and the pin configuration on the top left of the screen.

(b) Schematic of the 4×4 keypad used to enter the coordinates.

Figure 11: Figures of the hardware components used for the user input.

3.2.2 Chessboard display

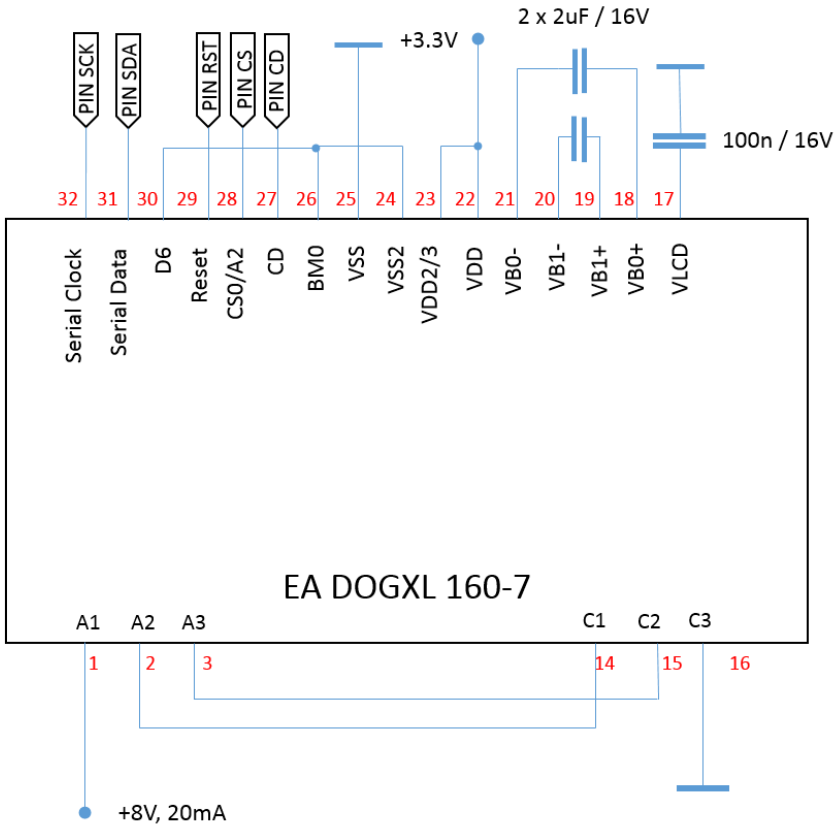


Figure 12: Circuit diagram of the wiring of the DOGXL 160-7 and the EA LED78x64-G backlight in four-wire SPI mode. See Figure 20 in the Appendix for a picture of the LCD wiring.

The chessboard display was achieved using the DOGXL 160-7 manufactured by Electronic Assembly, and the EA LED78x64-G backlight, which clips to the back of the screen. The backlight is powered by an external 8 V DC power supply, which powers the three LEDs wired in series (pins $A_{1/2/3}$-$C_{1/2/3}$).

The display is wired following the instructions of the user manual [7]. The screen is operated using pins (27-29 and 31-32) which are wired to PinCD, PinCS, PinRST, PinSDA and PinSCK and are driven by the microprocessor. Both D6 and BM0, which configure the serial interface, are grounded, indicating that the chip is operating in the 8 bit, four-wire interface. A 3.3 V DC power supply is provided through the board to VDD and VDD2/3, and the screen is grounded through the VSS and VSS2 pins.

The VB pins (18-21) are voltage converters and are wired to two 2 μF capacitors. VLCD is the power LC drive and is observed to be 13.7 V when the display is operating normally. This was used as a test to check if the display had initialised correctly.

Note that the documentation recommended using a 10 MΩ resistor in parallel with the 100 nF capacitor. This was not included in this setup.

4 Results, Performance and Improvements

4.1 Results

Figure 13: Photograph of the final result. The board and pieces have been initialised and the the whites can start playing.

The main objective of the project - to have a working player versus player chess game - was reached. The player can choose between two different modes at the start of the game: a demo mode, which runs through a ten move chess game and a game mode. In the game mode, two players can play against each other, taking turns to use the same keypad. The user can input the departure and destination square and the move is completed on the graphic display if the move is valid. With respect to coding the chess rules, all the main basic rules were transferred:

- The pawn is only allowed to move one square forward (or two if it is the first time it is moved) and can only eat diagonally
- The castle and bishop are only allowed to move along lines or diagonals respectively. The algorithm prevents the move from occurring if there is a piece in the way or if the move type is wrong.
- Queens can move both like a bishop and a castle. The program therefore combines the verification algorithm of the two pieces
- The king can move in any direction but only by one square. The game ends when one of the kings is taken.

With respect to the performance of the graphic LCD screen, the board and coordinate initialisation is done quickly, using the maximum clock speed of the ATMega128L, which is 8 MHz because the board is run at 3.3 V [9]. Moves are executed seamlessly and the algorithm points to the correct piece in the database that needs to be drawn on the screen. The result is a playable and graphically appealing chess game.

However, many technical and software challenges arose during the course of this project. The rules of chess were difficult to code because of the number of different cases that needed to be considered. Some of the software problems encountered were:

- The use of the **rcall** function to call functions which were too far away in program memory. This was solved by using the **call** function instead
- Using the incorrect number of **push** and **pop** functions. For the graphic display screen this would cause the screen to reset and enter infinite loops. In the main program this would also cause bugs. These errors are especially difficult to locate because the debugger doesn't detect them
- Coding branch statements for the rules of chess, because only *relative* branching is possible in AVR Assembly. Functions like **brlo** or **breq** had to refer to another function which would jump to the right place in the code

A number of hardware problems were encountered, in particular for the initialisation of the graphic LCD screen. These problems made the debugging of the move verification algorithm harder, because there was little way of knowing what the cause was. Firstly, the screen and the backlight are sold separately, despite the pins for the backlight being built on the screen. Secondly, the documentation recommends inserting a 10 MΩ resistor in parallel with the capacitor on Pin 17 (see Figure 12), which prevents the screen from initialising properly. Finally, a correct initialisation is only possible when the initialisation routine through the reset pin is completed, which isn't in the documentation. Without these last two points, the screen could not have been initialised.

Moreover, several glitches have been identified with the current setup:

- When the user enters a letter instead of a number or vice versa, the screen interprets the information as an A or a 1 depending on whether a number or a letter is requested
- Sometimes the code requests the information twice. It does not prevent the coordinate from being inputed correctly.
- The alphanumeric screen occasionally freezes, at which point the board must be turned off and on again for the code to run correctly. The reason for this bug is currently unknown, but it may be due to a loss of connection between the board and the screen.
- The keypad sometimes stops working because the connection with Port E sometimes fails. This is due to a faulty port, and a permanent solution would require switching the board. At the moment, the cable must be reinserted for the connection to work again.

Therefore, it can be said that the project was very ambitious within the timeframe available and many modifications and improvements can be proposed.

4.2 Modifications and improvements

4.2.1 More complex rules

Currently some of the more complicated chess rules have not been implemented. Complex moves such as *en passant*, where a pawn moving two squares forward can be retrospectively

eaten, and *rooking*, where the king and castle switch places, are not included. A pawn in chess should also be able to "upgrade" to another piece when it reaches the other side of the board. The user would have the possibility to choose what piece the pawn should upgrade to.

The game also would have to check whether the king is in check or not and signal it to the user. To do this, all the possible next moves of the opposition pieces would have to be checked. Other desirable features include checkmate and stalemate.

4.2.2 Usability

The current setup is not very user-friendly, because there is only one keypad and the switches on Port D feel clunky to the user. Usability would be improved if the user could highlight a square on the screen using a keypad instead of inputting a coordinate. Coding a text output library would also be necessary to do away with the alphanumeric screen and display all the information on the DOGXL screen.

4.2.3 Artificial Intelligence

Possibly the most challenging of the proposed improvements, artificial intelligence is a completely different challenge. A good algorithm would have to not only respond to the moves of the opponent, but also anticipate them. The simplest possible strategies to implement are random moves or an aggressive hunter strategy: when the algorithm sees the possibility of eating a piece it does so.

4.3 Product Specification

- Player versus Player chess game coded in AVR Assembly on an ATMega128 microprocessor board
- Output on a DOGXL160-7 FSTN graphic display interfaced using an 8-bit, four-wire SPI mode illuminated by a green backlight (EA LED78x64-G)
- Coordinate input with a 4×4 keypad and Hitachi HD44780U 4×20 alphanumeric screen
- A demo mode where the screen loops through a ten move game to display the screen's properties
- A game mode including the simplest rules of chess and a move verification algorithm ensuring that the moves given by the user are legal. A game ends when a king is taken.

5 Conclusion

To conclude, a working player versus chess game was coded on an ATMega128 board in AVR assembly. The result was outputted to a graphic LCD screen via an 8 bit SPI interface. The user interface was done using a 4×4 keypad and an alphanumeric screen. Two modes are available to the user: a demonstration sequence loops through a short chess game, and a game mode starts or loads a game. The most important chess rules are included, however more complex rules such as *en passant* and *rooking* are not available. A game ends when the opposition king is taken.

This was an ambitious and challenging project, and there are many possible extensions. These include the coding of more complex rules, such as those mentioned previously and improving the usability of the product. Usability could be improved by coding a text library to have all the text prompts on the graphic LCD and by having a tracker to select squares instead of coordinates. Finally, one could imagine a one player mode, where the user plays against the computer. This would require the inclusion of an artificial intelligence in the code.

References

[1] Coder creates smallest chess game for computers, *www.bbc.co.uk/news/technology-31028787*, last modified 28.01.15

[2] T.J. Scheffer and J.Nehring, *A new highly multiplexable LCD*, Appl. Phys. Lett. vol 45, no 10, pp.1021-1023, November 1984

[3] Atmel, *ATmega Board-Level User Guide*, 1999

[4] Mark Neil, *Decoding and Using a* 4×4 *Keyboard*, Microprocessor Course, 2015

[5] Mark Neil, *Device Drivers: LCD Display*, Microprocessor Course, 2015

[6] Probleme Kontrast DOGXL160, *https://www.mikrocontroller.net/topic/204914*, last modified 18.01.11

[7] Electronic Assembly, *DogXL graphic series user manual*, November 2010

[8] UltraChip, *UC1610* 128×160 *4S STN LCD Controller-Driver*, April 2005

[9] Atmel, *ATMega128 datasheet*, product specification

6 Appendices

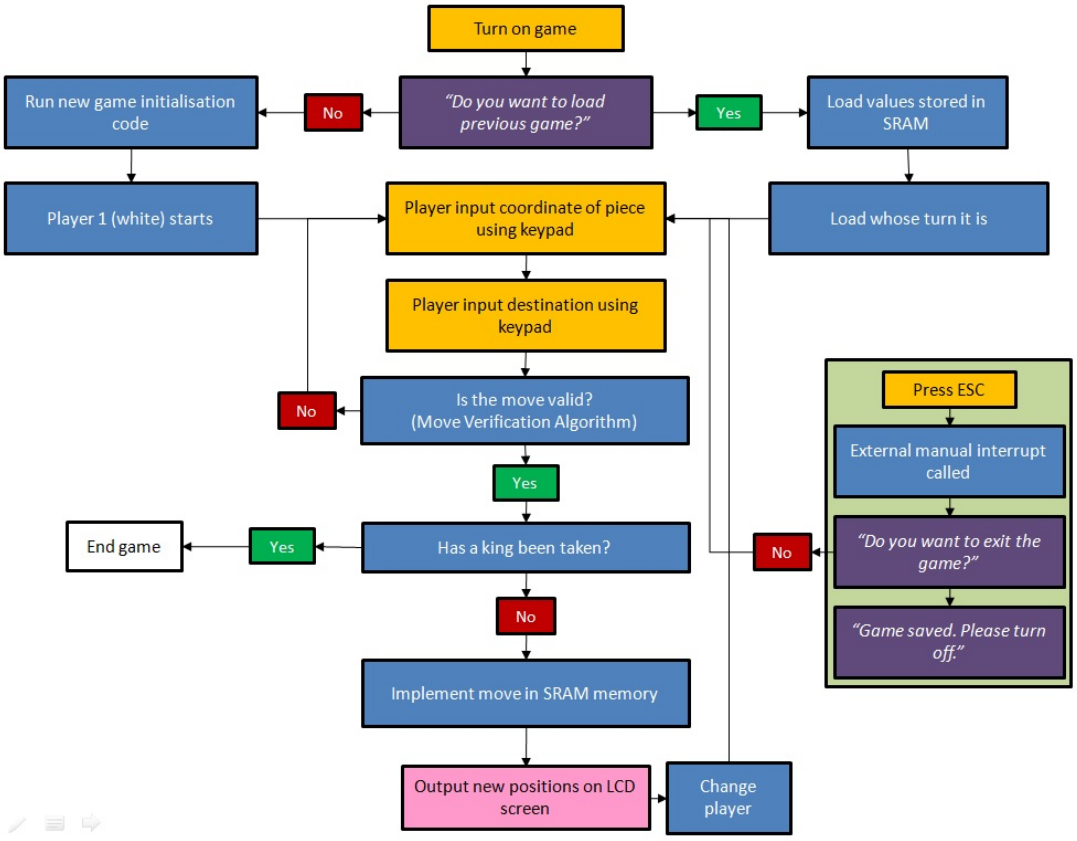
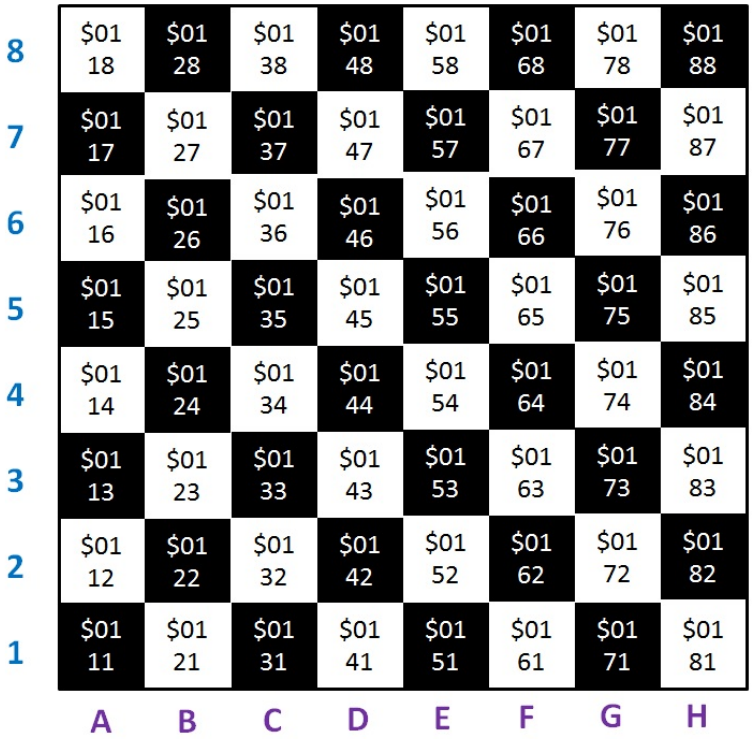
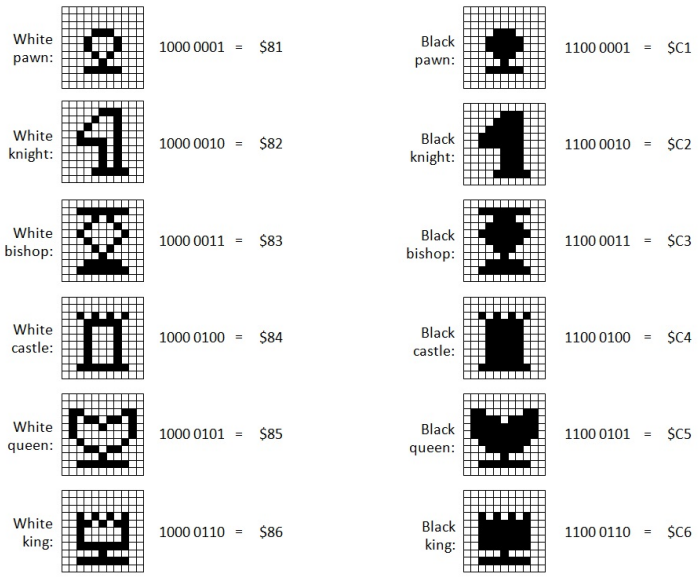


Figure 14: Flowchart of the main program illustrating the overall sequence of the program.



(a) Hex numbers used in the program corresponding to the conventional chess coordinates. The Hex numbers represent the position in memory where the state of the square is stored.



(b) Hex and binary values for each different piece. The highest nibble indicates the colour (black=$8 and white=$C) and the lowest nibble the type ($1-$6).

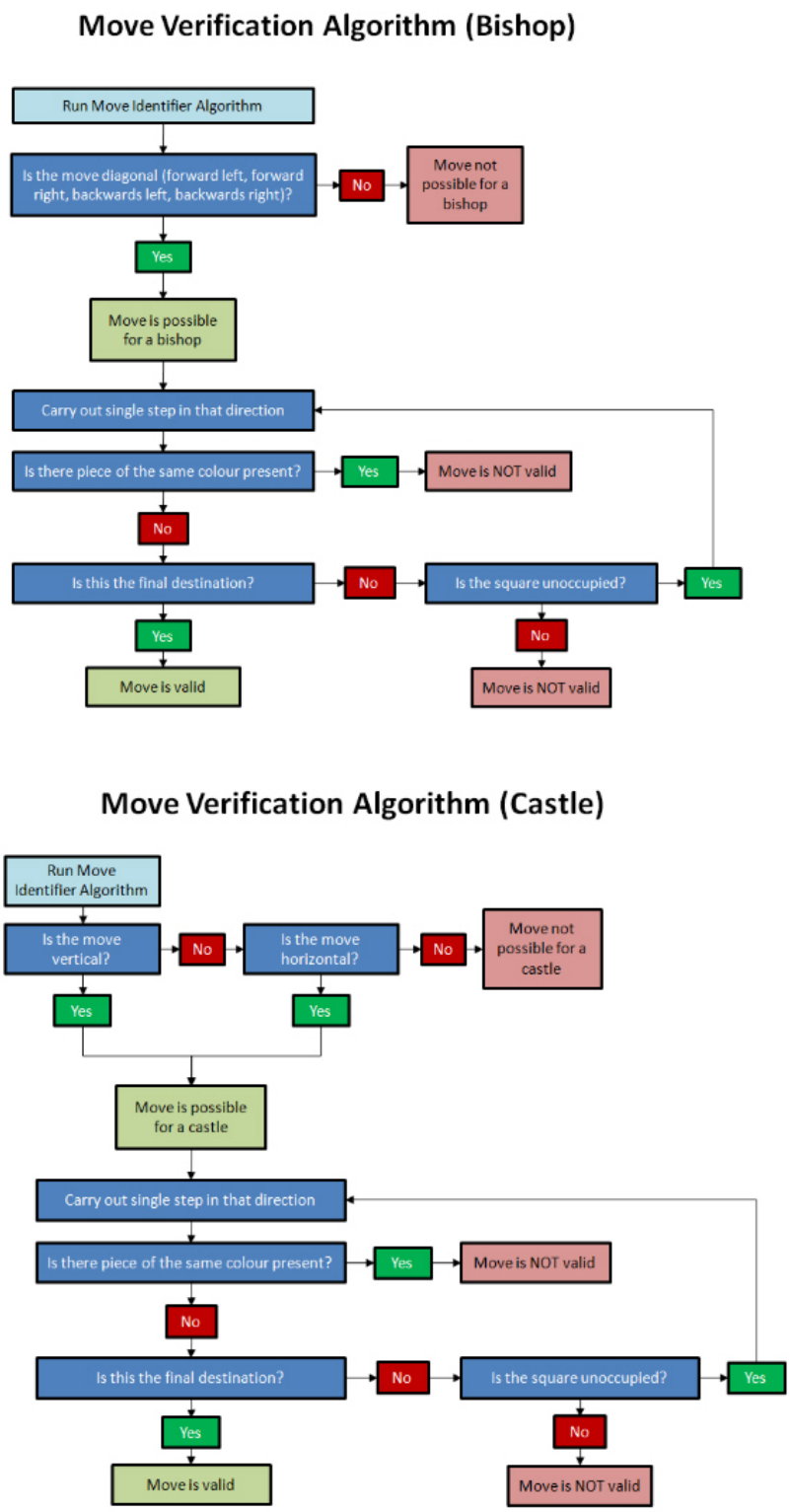


Figure 16: Move verification algorithms for the bishop and the castle.

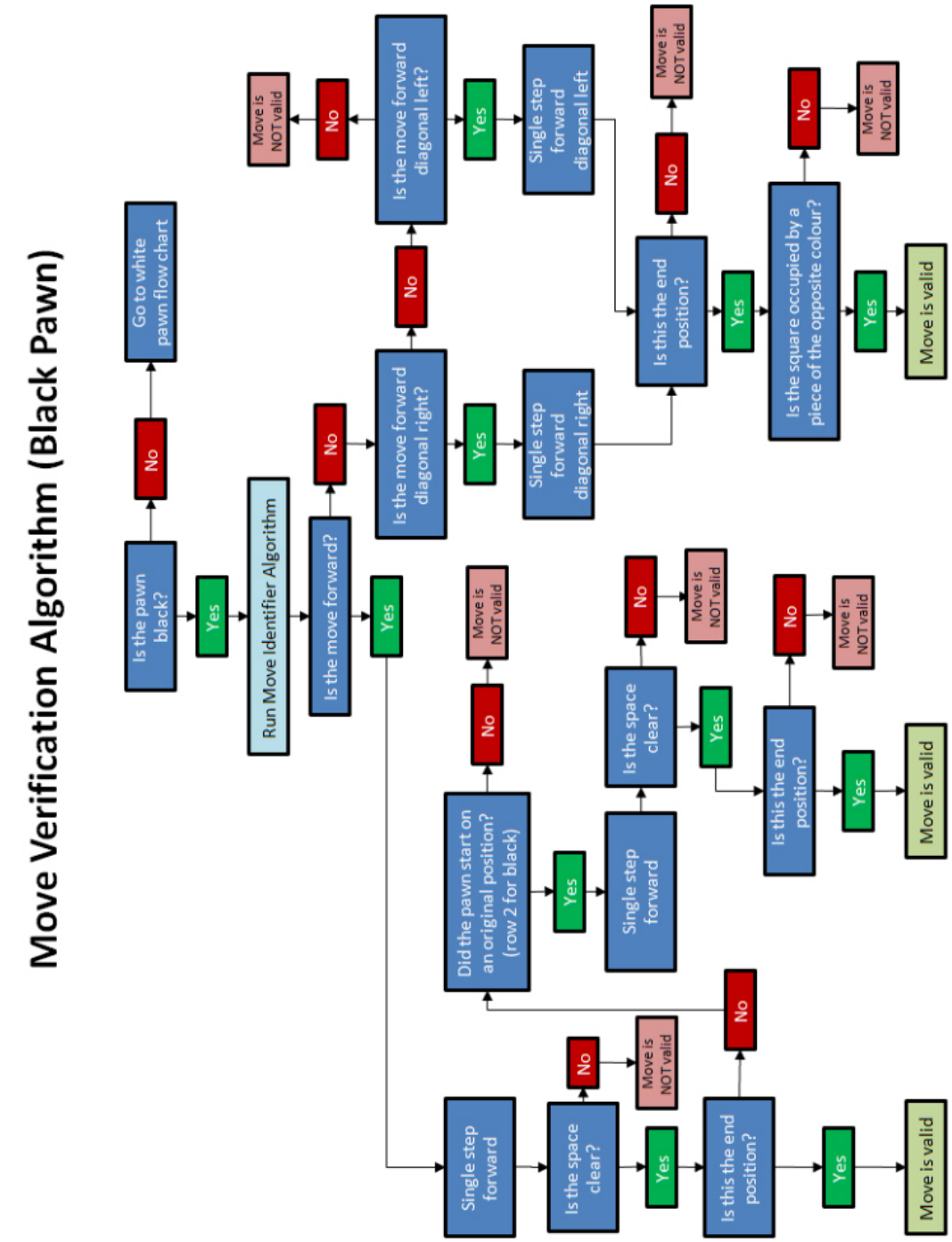


Figure 17: Move verification algorithms for the black pawn.

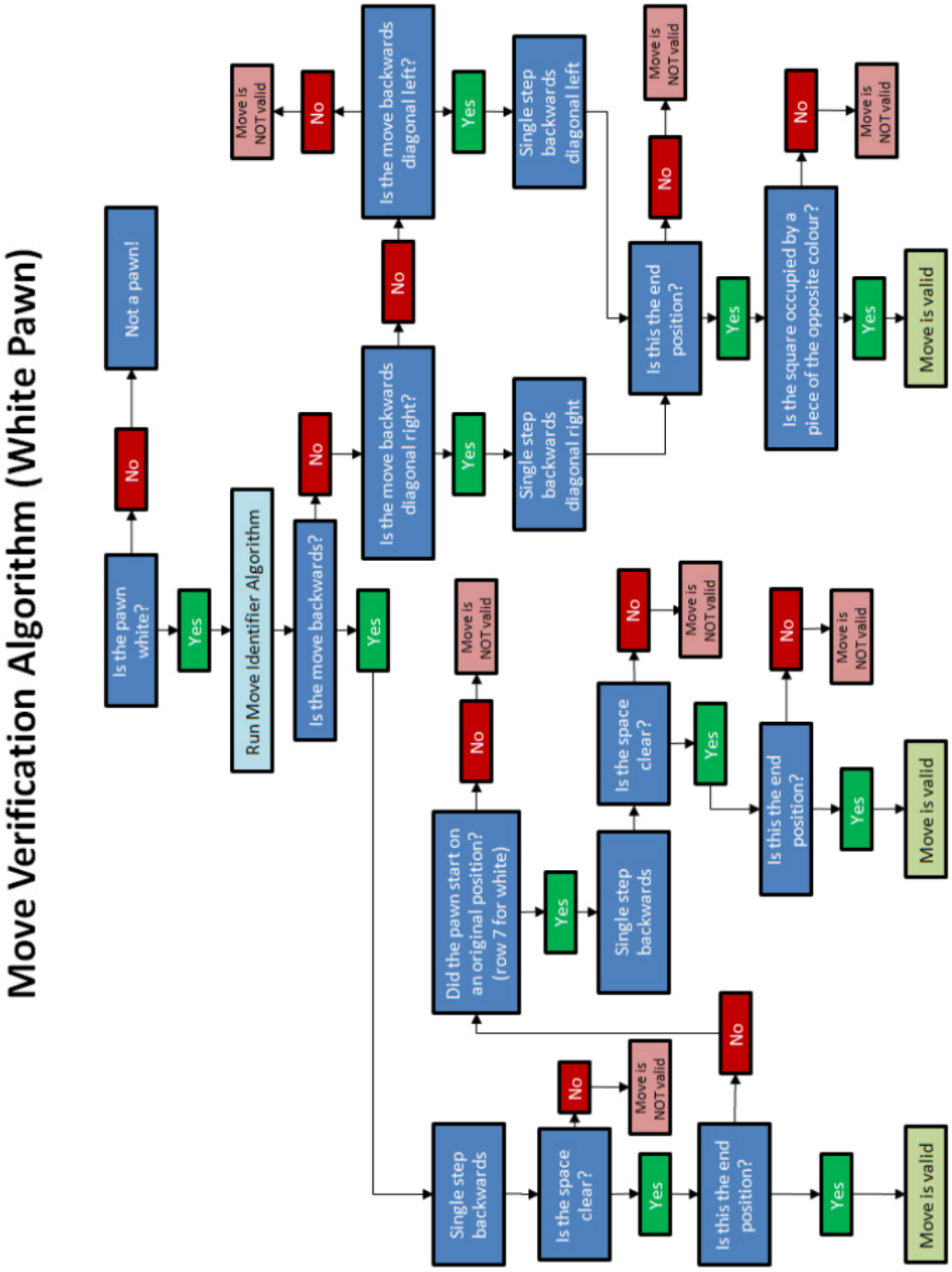


Figure 18: Move verification algorithms for the white pawn.

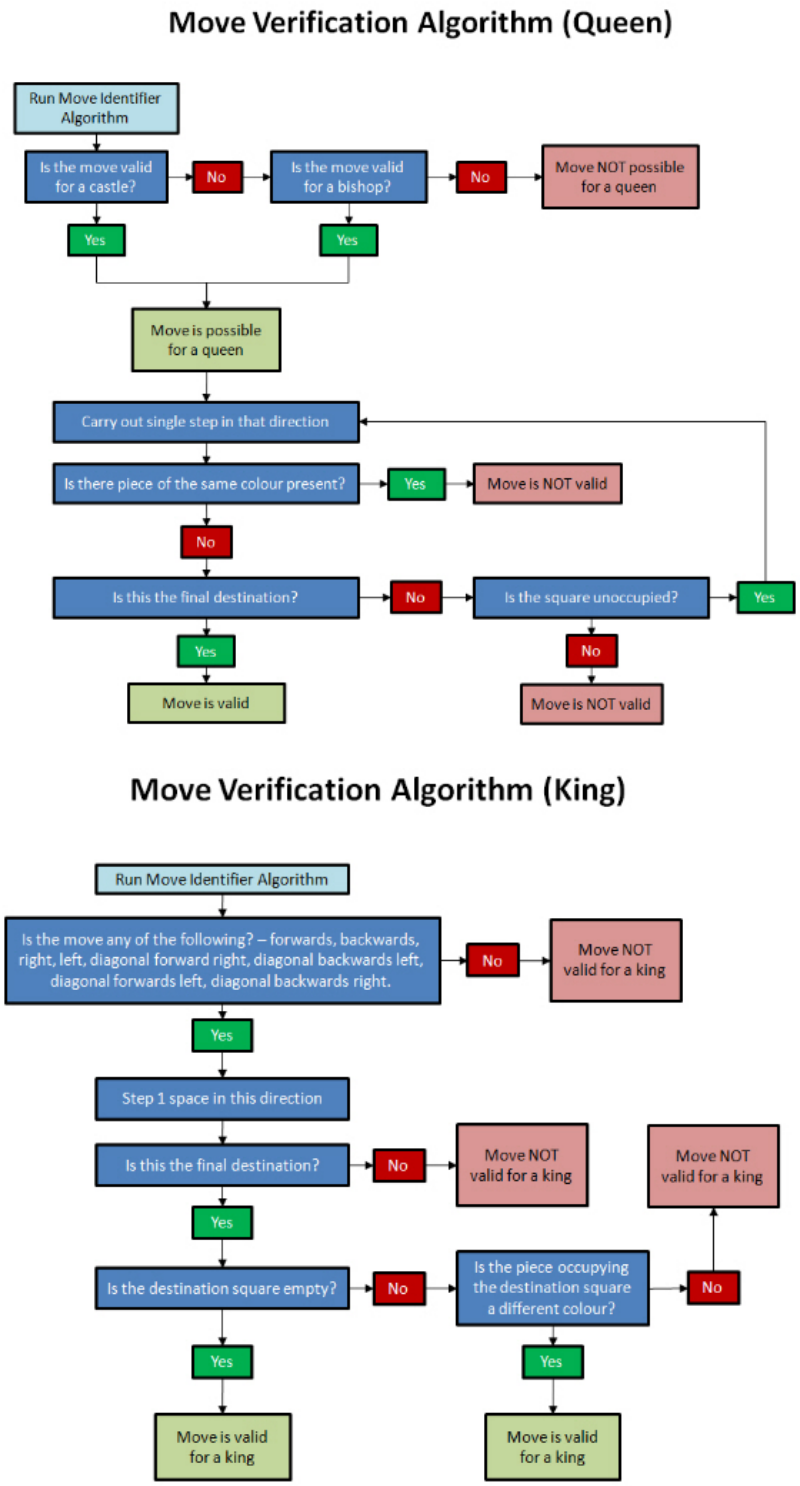


Figure 19: Move verification algorithms for the King and the Queen.

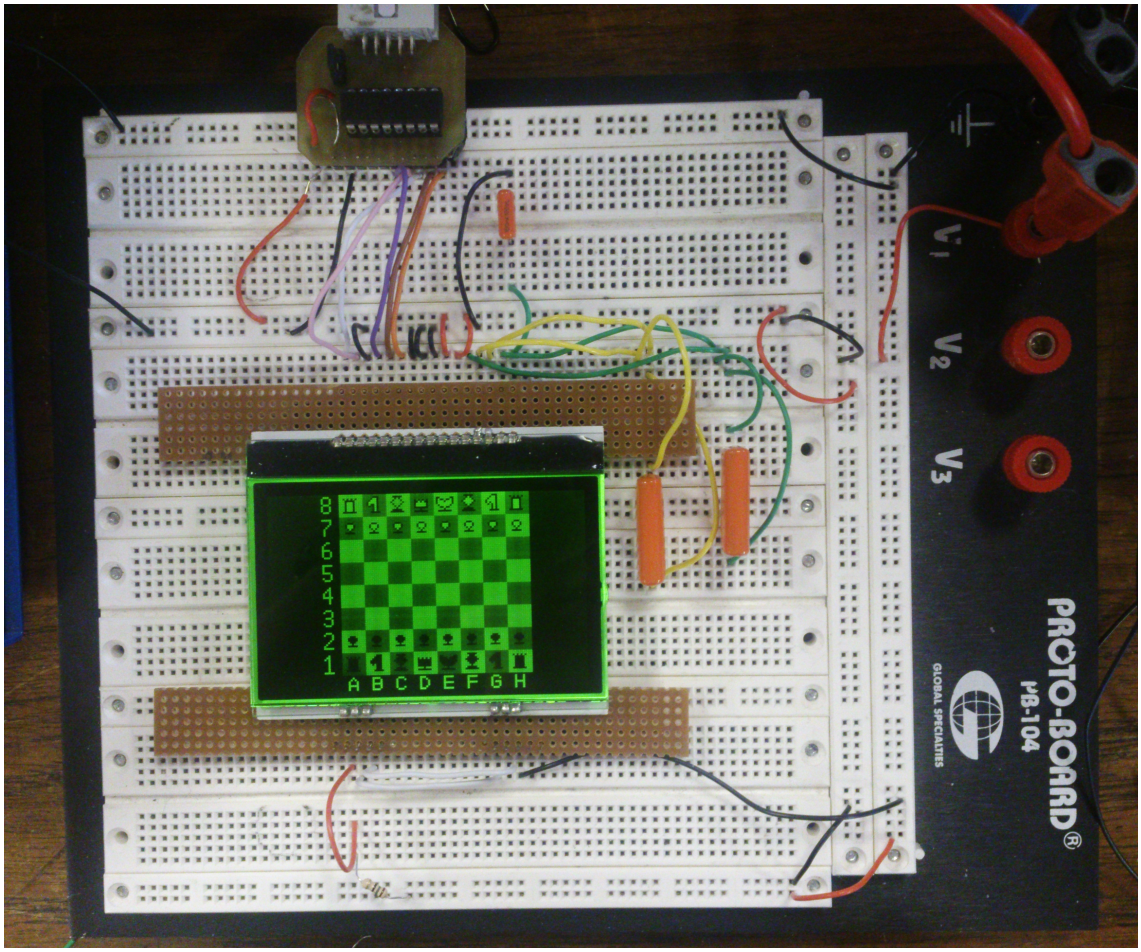


Figure 20: Picture of the wiring of the Graphic DOGXL LCD screen.

```
;
;*****************************************************************************
;*****************************************************************************
;
;           Hugo and Rich AKA dogxlcrew present:
;                 a chess game
;
;*****************************************************************************
;*****************************************************************************
;
;
.DEVICE ATmega128
.include "m128def.inc"
;
.org  $0
  jmp Init     ; jmp is 2 word instruction to set correct vector
  jmp EXT_INT0; External 0 interrupt vector
;
Init:
    ;  Setup the Stack Pointer to point at the end of SRAM
    ;  Put $0FFF in the 1 word SPH:SPL register pair
    ;
    ldi r16, $0F     ; Stack Pointer Setup
    out SPH,r16      ; Stack Pointer High Byte
    ldi r16, $FF     ; Stack Pointer Setup
    out SPL,r16      ; Stack Pointer Low Byte
      ;
    ; RAMPZ Setup Code
    ; Setup the RAMPZ so we are accessing the lower 64K words of program
    memory
    ;
    ldi  r16, $00    ; 1 = EPLM acts on upper 64K
    out RAMPZ, r16     ; 0 = EPLM acts on lower 64K
    ;
    ; ******* Sleep Mode And SRAM  *******
    ;      ; tell it we want read and write activity on RE WR
    ;
    ldi r16, $C0     ; Idle Mode - SE bit in MCUCR not set
    out MCUCR, r16     ; External SRAM Enable Wait State Enabled
      ;
    ; Comparator Setup Code
    ; set the Comparator Setup Registor to Disable Input capture and the
    comparator
```

```
    ;
43  ldi r16,$80      ; Comparator Disabled, Input Capture Disabled
    out ACSR, r16
45    ;
    ; ******* Port A Setup Code ****
47  ldi r16, $FF     ; $FF is output
    out DDRA, r16    ; Port A Direction Register
49  ldi r16, $FF     ; Init value
    out PORTA, r16     ; Port A value
51
    ; ******* Port B Setup Code ****
53  ldi r16, $FF     ; will set to outputs so i can use Leds for debugging
    out DDRB , r16     ; Port B Direction Register
55  ldi r16, $FF     ; Who cares what is it....
    out PORTB, r16     ; Port B value
57
    ; ******* Port C Setup Code ****
59  ldi r16, $FF     ; $FF is output
    out DDRC, r16    ; Port A Direction Register
61  ldi r16, $FF     ; Init value
    out PORTC, r16     ; Port A value
63
    ; ******* Port D Setup Code ****
65  ; Setup PORTD (the switches on the STK300) as inputs by setting the
    direction register
    ; bits to $00.  Set the initial value to $FF
67  ;
    ldi r16, $00     ; I/O:
69  out DDRD, r16    ; Port D Direction Register
    ldi r16, $FF     ; Init value
71  out PORTD, r16     ; Port D value
;
73  ;
    ; ******* Port F Setup Code ****
75  ;ldi r16, $FF    ; $FF is output
    ;out DDRF, r16     ; Port A Direction Register
77  ;ldi r16, $FF    ; Init value
    ;out PORTF, r16    ; Port A value
79  ;
    ;
81  ; ******* External Interrupt setup code ******
    ;
83  ldi r16,$01
    out EIMSK, r16
85  sei
    ;
87  ;Graphic LCD initialisation
    call initialise_LCD_DOGXL
89  ;
    ;Alphanumeric LCD initialisation
91  call alphanum_lcd_initialisation
    ;
93  ;
    call MessDemoQuestionOut
95  push r16
```

```
demo_question_loop:
in r16,PIND
com r16
cpi r16,$08
breq play_game
cpi r16,$10
brne demo_question_loop
pop r16
play_demo:
call CLRDIS
call MessPressESCOut
call initialise_LCD_DOGXL
push r16
push XL
push XH
ldi XH,$01
ldi XL,$10
ldi r16,$00
st X,r16
pop XH
pop XL
pop r16
demo_loop:
call pieces_initialisation
call demo
call initialise_LCD_DOGXL
rjmp demo_loop
play_game:
pop r16
push r16
push XL
push XH
ldi XH,$01
ldi XL,$10
ldi r16,$01
st X,r16
pop XH
pop XL
pop r16
call CLRDIS
;
push r16
pind_clear_loop:
in r16, PIND
com r16
cpi r16,$00
breq continue
rjmp pind_clear_loop
continue:
pop r16

;
;Place the pieces on the chess board
call chess_board_initialisation
```

```
151     ;call chess_board_initialisation_2  ;this is a debugging
        initialisation
        ;
153     ;Place inital pieces on screen
        call initial_pieces_onto_screen
155 MainMain:
    call user_input_position
157 ;
    call move_verification_algorithm
159 ;
    call valid_or_invalid ;if valid, saves move. if invalid, does not save
        move
161 ;
    call king_check
163 ;
    cpi r22,$01
165 breq output_to_DOGXL_screen
    rjmp MainMain
167 ;**********************
    output_to_DOGXL_screen:
169 push r25
    push r16
171 call output_DOGXL
    pop r16
173 mov r17,r16
    ldi r18,$00
175 call output_DOGXL
    pop r25
177 call change_colour
    rjmp MainMain
179 ;
    ; Include files
181 .include "initial_pieces_onto_screen.asm"
    .include "valid_or_invalid.asm"
183 .include "change_colour.asm"
    .include "king_check.asm"
185 .include "chess_board_initialisation.asm"
    .include "chess_board_initialisation_2.asm"
187 .include "user_input_code.asm"
    .include "move_identifier_algorithm.asm"
189 .include "move_verification_algorithm.asm"
    .include "alphanumeric_lcd_initialisation.asm"
191 .include "delay_routines.asm"
    .include "messages.asm"
193 .include "mapping.asm"
    .include "save_game_interrupt.asm"
195 .include "dogxl_initialisation.asm"
```

```
1
;Program to initialise and control the DOGXL-160W lcd in 4 wire, 8-bit SPI-mode
3 ;Hugo Jeanperrin
;27.02.15
5
;******************Register-Definition*********************
7 .def rPage         = R16 ;Byte containing the page address
.def rColumn        = R23 ;Byte containing the column address
9 .def rByteLCD      = R18 ;Byte sent to the lcd
.def rCount         = R19
11 .def rTemp1        = R20
.def rTemp2         = R21
13 .def rTemp3        = R22

15 ;*******************Port-Definition***********************
.equ PORTDOGLCD     = PORTB   ; SPI output port
17 .equ PINDOGLCD      = PINB
.equ DDRDOGLCD      = DDRB
19 .equ cInitPortDOGLCD  = 0b11111111

21 .equ PinSDA        = 3      ;Data transfer pin
.equ PinSCK         = 2      ;Clock
23 .equ PinCD          = 0      ;Command or data mode pin
.equ PinCS          = 1      ;Chip select
25 .equ PinReset      = 4      ;Is kept HIGH after the initialisation

27 .equ cSetPageAddress  = 0b01100000  ;the five lower bits contain the current page (pins 0-4)
.equ cSetColAddrMSB   = 0b00010000  ;the four lowest bits contain the highest nibble for the column
29 .equ cSetColAddrLSB   = 0b00000000  ;the four lowest bits contain the lowest nibble for the column
.equ cNumColumnsLCD   = 160      ;Number of pixels in the horizontal direction
31 .equ cNumPagesLCD   = 104/4       ;Number of pixels along the vertical axis /4 (number of pages)

33 ;The UC1610-LCD-Kontroller uses two bits to define a pixel
;Pixels: 1 = On, 00 = OFF
35 ;A page and a column therefore represent one byte of information

37 ;Constants setting the delay functions.
.equ cDelay5us      = 1
39 .equ cDelay10us     = 2


41


43 ;*****************************************************************
;Interrupt-Table
45 ;*****************************************************************
    jmp Initialise_LCD_DOGXL                    ; jmp is 2 word instruction to set correct vector
47     nop     ; Vector Addresses are 2 words apart
    reti      ; External 0 interrupt  Vector
```

```
49      nop     ; Vector Addresses are 2 words apart
        reti      ; External 1 interrupt  Vector
51      nop     ; Vector Addresses are 2 words apart
        reti      ; External 2 interrupt  Vector
53      nop     ; Vector Addresses are 2 words apart
        reti      ; External 3 interrupt  Vector
55      nop     ; Vector Addresses are 2 words apart
        reti      ; External 4 interrupt  Vector
57      nop     ; Vector Addresses are 2 words apart
        reti      ; External 5 interrupt  Vector
59      nop     ; Vector Addresses are 2 words apart
        reti      ; External 6 interrupt  Vector
61      nop     ; Vector Addresses are 2 words apart
        reti      ; External 7 interrupt  Vector
63      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 2 Compare Vector
65      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 2 Overflow Vector
67      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 1 Capture  Vector
69      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer1 CompareA  Vector
71      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 1 CompareB  Vector
73      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 1 Overflow  Vector
75      nop     ; Vector Addresses are 2 words apart
        reti      ; Timer 0 Compare  Vector
77      nop     ; Timer 0 Overflow interrupt  Vector
        nop     ; Vector Addresses are 2 words apart
79      reti      ; SPI  Vector
        nop     ; Vector Addresses are 2 words apart
81      reti      ; UART Receive  Vector
        nop     ; Vector Addresses are 2 words apart
83      reti      ; UDR Empty  Vector
        nop     ; Vector Addresses are 2 words apart
85      reti      ; UART Transmit  Vector
        nop     ; Vector Addresses are 2 words apart
87      reti      ; ADC Conversion Complete Vector
        nop     ; Vector Addresses are 2 words apart
89      reti      ; EEPROM Ready Vector
        nop     ; Vector Addresses are 2 words apart
91      reti      ; Analog Comparator  Vector
;-----------------------------------------------------------------------
93


95 ;*****************************************************************
Initialise_LCD_DOGXL:
97   ; ************* Stack Pointer Setup Code
       ldi rTemp1, $0F
99
       ; Stack Pointer Setup to 0x0FFF
101    ;out SPH,rTemp1    ; Stack Pointer High Byte
       ;ldi r16, $FF    ; Stack Pointer Setup
103    ;out SPL,rTemp1    ; Stack Pointer Low Byte
```

```
105     ; ******* Port B Setup Code ****
        ;ldi r16, $FF   ; will set to outputs to use Leds for debugging
107     ;out DDRB , r16   ; Port B Direction Register
        ;ldi r16, $FF
109     ;out PORTB, r16   ; Port B value

111   ;LCD initialisation
      rcall fcInitGraphik_DOGLCD
113   rcall Main_DOGXL
      rcall coordinates
115   ret
;-----------------------------------------------------------------------------
117

;************************************************************
119 ;Delay-Function 5us
;************************************************************
121 mDelay5us:
      LDI    XH, HIGH(cDelay5us)
123   LDI      XL, LOW(cDelay5us)
LoopDelayLCD5us:
125   ;NOP                ;1 clock cycle (cc)
      SBIW X,1                ;2 cc
127   BRNE   LoopDelayLCD5us ;1 cc
      ret
129 ;-----------------------------------------------------------------------

131 ;************************************************************
;Delay-Function 10us
133 ;************************************************************
mDelay10us:
135   LDI    XH, HIGH(cDelay10us)
      LDI      XL, LOW(cDelay10us)
137 LoopDelayLCD10us:
      ;NOP                ;1 cc
139   SBIW X,1          ;2 cc
      BRNE   LoopDelayLCD10us   ;1 cc
141   ret
;-----------------------------------------------------------------------
143

;************************************************************
145 ;Delay-Makro 1ms
;************************************************************
147 mDelay1ms:
      PUSH     XH
149   PUSH     XL
;
151 ; This is a 1 msec delay routine. Each cycle costs
; rcall       -> 3 CC
153 ; ret         -> 4 CC
; 2*LDI       -> 2 CC
155 ; SBIW        -> 2 CC * 2286
; BRNE        -> 1/2 CC * 2286
157 ; Total       -> 8009
```

```
159             LDI XH, HIGH(2286)
                LDI XL, LOW (2286)
161 COUNT1ms:
                SBIW X, 1
163             BRNE COUNT1ms
        POP XH
165     POP XL
                RET
167
;-----------------------------------------------------------------
169
;************************************************************
171 ;Delay-Function 100ms
;************************************************************
173 mDelay100ms:
    PUSH    XH
175 PUSH    XL
    PUSH    rTemp1
177 ;
; This is a 100 msec delay routine.
179 ; Do the 1 msed Function 100 times


181
        LDI rTemp1, 100
183 COUNT100ms:
        LDI XH, HIGH(2286)
185             LDI XL, LOW (2286)

187 COUNT100ms2:
                SBIW X, 1
189             BRNE COUNT100ms2
        SUBI rTemp1,1
191     BRNE COUNT100ms
        POP XH
193     POP XL
        POP rTemp1
195             RET
;-----------------------------------------------------------------
197
;************************************************************
199 ;Delay Function 1s
;************************************************************
201 delay1s:

203 ldi rTemp1,10
count1s:
205   rcall mDelay100ms
  subi rTemp1,1
207 brne count1s
  ret
209
;-----------------------------------------------------------------
211
;*****************************************************************
213 ;Main program
```

```
;*****************************************************************
Main_DOGXL:

  ;Program draws a chessboard with no pieces
  CLR   rPage ; Start at page = 0
  LDI   rTemp1,6

LoopPage:
  ldi rColumn,36  ;Column start for the chessboard
  ldi   rByteLCD, 0b10101010 ; Four light gray pixels
  RCALL fcSetCurrentLCDAddress
  LDI   rTemp2,12 ; Each square is 12 pixels wide
  CPI   rTemp1,4  ; Alternate the colour every 3 pages (12 pixels)
  BRLO  LoopChessY
  DEC   rTemp1
  ldi   rByteLCD, 0b10101010 ;Four dark gray pixels

LoopCol:

  CPI   rTemp2,0
  BREQ    LoopChessX
  DEC   rTemp2
  RCALL fcSendDataToLCD
  INC   rColumn
  CPI   rColumn, 132 ;Column coordinate end for the chessboard
  BRNE  LoopCol


  INC   rPage
  CPI   rPage, 24 ; Page coordinate end for the chessboard
  BRLO  LoopPage

ret



EndMain:
  rjmp EndMain

LoopChessX:
  com rByteLCD
  ldi rTemp2,12
  rjmp LoopCol

LoopChessY:
  LDI   rByteLCD, 0b01010101 ;Four dark gray pixels
  DEC   rTemp1
  cpi rTemp1,0
  breq LoopChessy2
  rjmp LoopCol

LoopChessy2:
  ldi rTemp1,6
  rjmp LoopCol
;---------------------------------------------------------------------
```

```
;*********************************************************************
; Output coordinates to the screen
;*********************************************************************

coordinates:

  LDI ZH, HIGH(2*Num8)
  LDI ZL, LOW(2*Num8)
  ldi rPage,0
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num7)
  LDI ZL, LOW(2*Num7)
  ldi rPage,3
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num6)
  LDI ZL, LOW(2*Num6)
  ldi rPage,6
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num5)
  LDI ZL, LOW(2*Num5)
  ldi rPage,9
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num4)
  LDI ZL, LOW(2*Num4)
  ldi rPage,12
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num3)
  LDI ZL, LOW(2*Num3)
  ldi rPage,15
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num2)
  LDI ZL, LOW(2*Num2)
  ldi rPage,18
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Num1)
  LDI ZL, LOW(2*Num1)
  ldi rPage,21
  ldi rColumn,24
  rcall CharOut

  LDI ZH, HIGH(2*Numa)
```

```
    LDI ZL, LOW(2*Numa)
325 ldi rPage,24
    ldi rColumn,36
327 rcall CharOut

329 LDI ZH, HIGH(2*Numb)
    LDI ZL, LOW(2*Numb)
331 ldi rPage,24
    ldi rColumn,48
333 rcall CharOut

335 LDI ZH, HIGH(2*Numc)
    LDI ZL, LOW(2*Numc)
337 ldi rPage,24
    ldi rColumn,60
339 rcall CharOut

341 LDI ZH, HIGH(2*Numd)
    LDI ZL, LOW(2*Numd)
343 ldi rPage,24
    ldi rColumn,72
345 rcall CharOut

347 LDI ZH, HIGH(2*Nume)
    LDI ZL, LOW(2*Nume)
349 ldi rPage,24
    ldi rColumn,84
351 rcall CharOut

353 LDI ZH, HIGH(2*Numf)
    LDI ZL, LOW(2*Numf)
355 ldi rPage,24
    ldi rColumn,96
357 rcall CharOut

359 LDI ZH, HIGH(2*Numg)
    LDI ZL, LOW(2*Numg)
361 ldi rPage,24
    ldi rColumn,108
363 rcall CharOut

365

    LDI ZH, HIGH(2*Numh)
367 LDI ZL, LOW(2*Numh)
    ldi rPage,24
369 ldi rColumn,120
    rcall CharOut
371 ret

373 ;-----------------------------------------------------------------
    ;*****************************************************************
375 ; Output pieces to the screen - this function can be called by the
    ; main program to output a move to the LCD screen. The destination is
377 ; given in r17 and the value of the piece is given by r18.
    ;*****************************************************************
```

```
379
    output_DOGXL:
381 ; Column value conversion
    push r17
383 push r18

385 push r17
    andi r17,0b11110000
387 cpi r17,$10
    breq coordinate_a
389 cpi r17,$20
    breq coordinate_b
391 cpi r17,$30
    breq coordinate_c
393 cpi r17,$40
    breq coordinate_d
395 cpi r17,$50
    breq coordinate_e
397 cpi r17,$60
    breq coordinate_f
399 cpi r17,$70
    breq coordinate_g
401 cpi r17,$80
    breq coordinate_h
403
    coordinate_a:
405 ldi rColumn,36
    rjmp Page_conversion
407 coordinate_b:
    ldi rColumn,48
409 rjmp Page_conversion
    coordinate_c:
411 ldi rColumn,60
    rjmp Page_conversion
413 coordinate_d:
    ldi rColumn,72
415 rjmp Page_conversion
    coordinate_e:
417 ldi rColumn,84
    rjmp Page_conversion
419 coordinate_f:
    ldi rColumn,96
421 rjmp Page_conversion
    coordinate_g:
423 ldi rColumn,108
    rjmp Page_conversion
425 coordinate_h:
    ldi rColumn,120
427 rjmp Page_conversion

429 Page_conversion:
    pop r17
431 push r17
    andi r17,0b00001111
433 cpi r17,$1
```

```
    breq coordinate_1
435 cpi r17,$2
    breq coordinate_2
437 cpi r17,$3
    breq coordinate_3
439 cpi r17,$4
    breq coordinate_4
441 cpi r17,$5
    breq coordinate_5
443 cpi r17,$6
    breq coordinate_6
445 cpi r17,$7
    breq coordinate_7
447 cpi r17,$8
    breq coordinate_8
449
    coordinate_1:
451 ldi rPage,21
    rjmp Piece_conversion
453 coordinate_2:
    ldi rPage,18
455 rjmp Piece_conversion
    coordinate_3:
457 ldi rPage,15
    rjmp Piece_conversion
459 coordinate_4:
    ldi rPage,12
461 rjmp Piece_conversion
    coordinate_5:
463 ldi rPage,9
    rjmp Piece_conversion
465 coordinate_6:
    ldi rPage,6
467 rjmp Piece_conversion
    coordinate_7:
469 ldi rPage,3
    rjmp Piece_conversion
471 coordinate_8:
    ldi rPage,0
473 rjmp Piece_conversion

475 Piece_conversion:
    pop r17
477 rcall odd_or_even
    add r18,r17
479
    cpi r18,$81
481 breq white_pawn_black_background
    cpi r18,$84
483 breq white_castle_black_background
    cpi r18,$82
485 breq white_knight_black_background
    cpi r18,$83
487 breq white_bishop_black_background
    cpi r18,$85
```

```
489 breq white_queen_black_background
    cpi r18,$86
491 breq white_king_black_background
    rjmp white_white
493

495 ;*************************

497 white_pawn_black_background:
    ldi ZH, HIGH(2*white_pawn_black_square)
499 ldi ZL, LOW(2*white_pawn_black_square)
    rjmp piece_output
501 white_castle_black_background:
    ldi ZH, HIGH(2*white_tower_black_square)
503 ldi ZL, LOW(2*white_tower_black_square)
    rjmp piece_output
505 white_knight_black_background:
    ldi ZH, HIGH(2*white_knight_black_square)
507 ldi ZL, LOW(2*white_knight_black_square)
    rjmp piece_output
509 white_bishop_black_background:
    ldi ZH, HIGH(2*white_bishop_black_square)
511 ldi ZL, LOW(2*white_bishop_black_square)
    rjmp piece_output
513 white_queen_black_background:
    ldi ZH, HIGH(2*white_queen_black_square)
515 ldi ZL, LOW(2*white_queen_black_square)
    rjmp piece_output
517 white_king_black_background:
    ldi ZH, HIGH(2*white_king_black_square)
519 ldi ZL, LOW(2*white_king_black_square)
    rjmp piece_output
521

523 white_white:
    cpi r18,$91
525 breq white_pawn_white_background
    cpi r18,$94
527 breq white_castle_white_background
    cpi r18,$92
529 breq white_knight_white_background
    cpi r18,$93
531 breq white_bishop_white_background
    cpi r18,$95
533 breq white_queen_white_background
    cpi r18,$96
535 breq white_king_white_background
    rjmp black_black
537


539
    white_pawn_white_background:
541 ldi ZH, HIGH(2*white_pawn_white_square)
    ldi ZL, LOW(2*white_pawn_white_square)
543 rjmp piece_output
```

```
white_castle_white_background:
ldi ZH, HIGH(2*white_tower_white_square)
ldi ZL, LOW(2*white_tower_white_square)
rjmp piece_output
white_knight_white_background:
ldi ZH, HIGH(2*white_knight_white_square)
ldi ZL, LOW(2*white_knight_white_square)
rjmp piece_output
white_bishop_white_background:
ldi ZH, HIGH(2*white_bishop_white_square)
ldi ZL, LOW(2*white_bishop_white_square)
rjmp piece_output
white_queen_white_background:
ldi ZH, HIGH(2*white_queen_white_square)
ldi ZL, LOW(2*white_queen_white_square)
rjmp piece_output
white_king_white_background:
ldi ZH, HIGH(2*white_king_white_square)
ldi ZL, LOW(2*white_king_white_square)
rjmp piece_output

black_black:

cpi r18,$C1
breq black_pawn_black_background
cpi r18,$C4
breq black_castle_black_background
cpi r18,$C2
breq black_knight_black_background
cpi r18,$C3
breq black_bishop_black_background
cpi r18,$C5
breq black_queen_black_background
cpi r18,$C6
breq black_king_black_background
rjmp black_white


black_pawn_black_background:
ldi ZH, HIGH(2*black_pawn_black_square)
ldi ZL, LOW(2*black_pawn_black_square)
rjmp piece_output
black_castle_black_background:
ldi ZH, HIGH(2*black_tower_black_square)
ldi ZL, LOW(2*black_tower_black_square)
rjmp piece_output
black_knight_black_background:
ldi ZH, HIGH(2*black_knight_black_square)
ldi ZL, LOW(2*black_knight_black_square)
rjmp piece_output
black_bishop_black_background:
ldi ZH, HIGH(2*black_bishop_black_square)
ldi ZL, LOW(2*black_bishop_black_square)
rjmp piece_output
black_queen_black_background:
```

```
599 ldi ZH, HIGH(2*black_queen_black_square)
    ldi ZL, LOW(2*black_queen_black_square)
601 rjmp piece_output
    black_king_black_background:
603 ldi ZH, HIGH(2*black_king_black_square)
    ldi ZL, LOW(2*black_king_black_square)
605 rjmp piece_output

607 black_white:

609 cpi r18,$D1
    breq black_pawn_white_background
611 cpi r18,$D4
    breq black_castle_white_background
613 cpi r18,$D2
    breq black_knight_white_background
615 cpi r18,$D3
    breq black_bishop_white_background
617 cpi r18,$D5
    breq black_queen_white_background
619 cpi r18,$D6
    breq black_king_white_background
621 cpi r18,$00
    breq black_empty_square
623 cpi r18,$10
    breq white_empty_square
625

627 black_pawn_white_background:
    ldi ZH, HIGH(2*black_pawn_white_square)
629 ldi ZL, LOW(2*black_pawn_white_square)
    rjmp piece_output
631 black_castle_white_background:
    ldi ZH, HIGH(2*black_tower_white_square)
633 ldi ZL, LOW(2*black_tower_white_square)
    rjmp piece_output
635 black_knight_white_background:
    ldi ZH, HIGH(2*black_knight_white_square)
637 ldi ZL, LOW(2*black_knight_white_square)
    rjmp piece_output
639 black_bishop_white_background:
    ldi ZH, HIGH(2*black_bishop_white_square)
641 ldi ZL, LOW(2*black_bishop_white_square)
    rjmp piece_output
643 black_queen_white_background:
    ldi ZH, HIGH(2*black_queen_white_square)
645 ldi ZL, LOW(2*black_queen_white_square)
    rjmp piece_output
647 black_king_white_background:
    ldi ZH, HIGH(2*black_king_white_square)
649 ldi ZL, LOW(2*black_king_white_square)
    rjmp piece_output
651 black_empty_square:
    ldi ZH, HIGH(2*empty_black_square)
653 ldi ZL, LOW(2*empty_black_square)
```

```
    rjmp piece_output
655 white_empty_square:
    ldi ZH, HIGH(2*empty_white_square)
657 ldi ZL, LOW(2*empty_white_square)
    rjmp piece_output
659
    piece_output:
661 rcall CharOut

663 pop r17
    pop r18
665


667 ret
    ;-----------------------------------------------------------------------
669 ;***********************************************************************
    ; Initialise the board by outputting the pieces in the correct place
671 ;***********************************************************************
    pieces_initialisation:
673
      LDI ZH, HIGH(2*white_pawn_black_square)
675   LDI ZL, LOW(2*white_pawn_black_square)
      ldi rPage,3
677   ldi rColumn,36
      rcall CharOut
679
      LDI ZH, HIGH(2*white_pawn_black_square)
681   LDI ZL, LOW(2*white_pawn_black_square)
      ldi rPage,3
683   ldi rColumn,60
      rcall CharOut
685
      LDI ZH, HIGH(2*white_pawn_black_square)
687   LDI ZL, LOW(2*white_pawn_black_square)
      ldi rPage,3
689   ldi rColumn,84
      rcall CharOut
691
      LDI ZH, HIGH(2*white_pawn_black_square)
693   LDI ZL, LOW(2*white_pawn_black_square)
      ldi rPage,3
695   ldi rColumn,108
      rcall CharOut
697
      LDI ZH, HIGH(2*white_pawn_white_square)
699   LDI ZL, LOW(2*white_pawn_white_square)
      ldi rPage,3
701   ldi rColumn,48
      rcall CharOut
703
      LDI ZH, HIGH(2*white_pawn_white_square)
705   LDI ZL, LOW(2*white_pawn_white_square)
      ldi rPage,3
707   ldi rColumn,72
      rcall CharOut
```

```
709
711     LDI ZH, HIGH(2*white_pawn_white_square)
        LDI ZL, LOW(2*white_pawn_white_square)
713     ldi rPage,3
        ldi rColumn,96
715     rcall CharOut

717     LDI ZH, HIGH(2*white_pawn_white_square)
        LDI ZL, LOW(2*white_pawn_white_square)
719     ldi rPage,3
        ldi rColumn,120
721     rcall CharOut

723     LDI ZH, HIGH(2*black_pawn_black_square)
        LDI ZL, LOW(2*black_pawn_black_square)
725     ldi rPage,18
        ldi rColumn,48
727     rcall CharOut

729     LDI ZH, HIGH(2*black_pawn_black_square)
        LDI ZL, LOW(2*black_pawn_black_square)
731     ldi rPage,18
        ldi rColumn,72
733     rcall CharOut

735     LDI ZH, HIGH(2*black_pawn_black_square)
        LDI ZL, LOW(2*black_pawn_black_square)
737     ldi rPage,18
        ldi rColumn,96
739     rcall CharOut

741     LDI ZH, HIGH(2*black_pawn_black_square)
        LDI ZL, LOW(2*black_pawn_black_square)
743     ldi rPage,18
        ldi rColumn,120
745     rcall CharOut

747     LDI ZH, HIGH(2*black_pawn_white_square)
        LDI ZL, LOW(2*black_pawn_white_square)
749     ldi rPage,18
        ldi rColumn,36
751     rcall CharOut

753     LDI ZH, HIGH(2*black_pawn_white_square)
        LDI ZL, LOW(2*black_pawn_white_square)
755     ldi rPage,18
        ldi rColumn,60
757     rcall CharOut

759     LDI ZH, HIGH(2*black_pawn_white_square)
        LDI ZL, LOW(2*black_pawn_white_square)
761     ldi rPage,18
        ldi rColumn,84
763     rcall CharOut
```

```
765     LDI ZH, HIGH(2*black_pawn_white_square)
        LDI ZL, LOW(2*black_pawn_white_square)
767     ldi rPage,18
        ldi rColumn,108
769     rcall CharOut

771     LDI ZH, HIGH(2*white_bishop_white_square)
        LDI ZL, LOW(2*white_bishop_white_square)
773     ldi rPage,0
        ldi rColumn,60
775     rcall CharOut

777     LDI ZH, HIGH(2*black_bishop_white_square)
        LDI ZL, LOW(2*black_bishop_white_square)
779     ldi rPage,21
        ldi rColumn,96
781     rcall CharOut

783     LDI ZH, HIGH(2*black_bishop_black_square)
        LDI ZL, LOW(2*black_bishop_black_square)
785     ldi rPage,21
        ldi rColumn,60
787     rcall CharOut

789     LDI ZH, HIGH(2*white_bishop_black_square)
        LDI ZL, LOW(2*white_bishop_black_square)
791     ldi rPage,0
        ldi rColumn,96
793     rcall CharOut

795     LDI ZH, HIGH(2*white_knight_black_square)
        LDI ZL, LOW(2*white_knight_black_square)
797     ldi rPage,0
        ldi rColumn,48
799     rcall CharOut

801     LDI ZH, HIGH(2*black_knight_black_square)
        LDI ZL, LOW(2*black_knight_black_square)
803     ldi rPage,21
        ldi rColumn,108
805     rcall CharOut

807     LDI ZH, HIGH(2*black_knight_white_square)
        LDI ZL, LOW(2*black_knight_white_square)
809     ldi rPage,21
        ldi rColumn,48
811     rcall CharOut

813     LDI ZH, HIGH(2*white_knight_white_square)
        LDI ZL, LOW(2*white_knight_white_square)
815     ldi rPage,0
        ldi rColumn,108
817     rcall CharOut
```

```
819     LDI ZH, HIGH(2*white_tower_white_square)
        LDI ZL, LOW(2*white_tower_white_square)
821     ldi rPage,0
        ldi rColumn,36
823     rcall CharOut

825     LDI ZH, HIGH(2*black_tower_white_square)
        LDI ZL, LOW(2*black_tower_white_square)
827     ldi rPage,21
        ldi rColumn,120
829     rcall CharOut

831     LDI ZH, HIGH(2*black_tower_black_square)
        LDI ZL, LOW(2*black_tower_black_square)
833     ldi rPage,21
        ldi rColumn,36
835     rcall CharOut

837     LDI ZH, HIGH(2*white_tower_black_square)
        LDI ZL, LOW(2*white_tower_black_square)
839     ldi rPage,0
        ldi rColumn,120
841     rcall CharOut

843     LDI ZH, HIGH(2*white_queen_white_square)
        LDI ZL, LOW(2*white_queen_white_square)
845     ldi rPage,0
        ldi rColumn,84
847     rcall CharOut

849     LDI ZH, HIGH(2*black_queen_black_square)
        LDI ZL, LOW(2*black_queen_black_square)
851     ldi rPage,21
        ldi rColumn,84
853     rcall CharOut

855     LDI ZH, HIGH(2*white_king_black_square)
        LDI ZL, LOW(2*white_king_black_square)
857     ldi rPage,0
        ldi rColumn,72
859     rcall CharOut

861     LDI ZH, HIGH(2*black_king_white_square)
        LDI ZL, LOW(2*black_king_white_square)
863     ldi rPage,21
        ldi rColumn,72
865     rcall CharOut

867


869
    ret
871 ;*****************************************************************
    ; Checkmate in 10 moves
873 ;*****************************************************************
```

```
demo:
875
    rcall delay1s
877
    LDI ZH, HIGH(2*empty_black_square)
879 LDI ZL, LOW(2*empty_black_square)
    ldi rPage,3
881 ldi rColumn,84
    rcall CharOut
883
    LDI ZH, HIGH(2*white_pawn_black_square)
885 LDI ZL, LOW(2*white_pawn_black_square)
    ldi rPage,9
887 ldi rColumn,84
    rcall CharOut
889
    rcall delay1s
891
    LDI ZH, HIGH(2*empty_white_square)
893 LDI ZL, LOW(2*empty_white_square)
    ldi rPage,18
895 ldi rColumn,84
    rcall CharOut
897
    LDI ZH, HIGH(2*black_pawn_white_square)
899 LDI ZL, LOW(2*black_pawn_white_square)
    ldi rPage,12
901 ldi rColumn,84
    rcall CharOut
903
    rcall delay1s
905
    LDI ZH, HIGH(2*empty_black_square)
907 LDI ZL, LOW(2*empty_black_square)
    ldi rPage,0
909 ldi rColumn,48
    rcall CharOut
911
    LDI ZH, HIGH(2*white_knight_white_square)
913 LDI ZL, LOW(2*white_knight_white_square)
    ldi rPage,6
915 ldi rColumn,60
    rcall CharOut
917
    rcall delay1s
919
    LDI ZH, HIGH(2*empty_white_square)
921 LDI ZL, LOW(2*empty_white_square)
    ldi rPage,21
923 ldi rColumn,96
    rcall CharOut
925
    LDI ZH, HIGH(2*black_bishop_white_square)
927 LDI ZL, LOW(2*black_bishop_white_square)
    ldi rPage,9
```

```
929     ldi rColumn,48
        rcall CharOut
931
        rcall delay1s
933
        LDI ZH, HIGH(2*empty_white_square)
935     LDI ZL, LOW(2*empty_white_square)
        ldi rPage,6
937     ldi rColumn,60
        rcall CharOut
939
        LDI ZH, HIGH(2*white_knight_black_square)
941     LDI ZL, LOW(2*white_knight_black_square)
        ldi rPage,12
943     ldi rColumn,72
        rcall CharOut
945
        rcall delay1s
947
        LDI ZH, HIGH(2*empty_white_square)
949     LDI ZL, LOW(2*empty_white_square)
        ldi rPage,18
951     ldi rColumn,36
        rcall CharOut
953
        LDI ZH, HIGH(2*black_pawn_white_square)
955     LDI ZL, LOW(2*black_pawn_white_square)
        ldi rPage,12
957     ldi rColumn,36
        rcall CharOut
959
        rcall delay1s
961
        LDI ZH, HIGH(2*empty_white_square)
963     LDI ZL, LOW(2*empty_white_square)
        ldi rPage,0
965     ldi rColumn,84
        rcall CharOut
967
        LDI ZH, HIGH(2*white_queen_white_square)
969     LDI ZL, LOW(2*white_queen_white_square)
        ldi rPage,6
971     ldi rColumn,84
        rcall CharOut
973
        rcall delay1s
975
        LDI ZH, HIGH(2*empty_white_square)
977     LDI ZL, LOW(2*empty_white_square)
        ldi rPage,18
979     ldi rColumn,60
        rcall CharOut
981
        LDI ZH, HIGH(2*black_pawn_black_square)
983     LDI ZL, LOW(2*black_pawn_black_square)
```

```
    ldi rPage,15
985 ldi rColumn,60
    rcall CharOut
987
    rcall delay1s
989

991 LDI ZH, HIGH(2*empty_white_square)
    LDI ZL, LOW(2*empty_white_square)
993 ldi rPage,6
    ldi rColumn,84
995 rcall CharOut

997
    LDI ZH, HIGH(2*white_queen_white_square)
999 LDI ZL, LOW(2*white_queen_white_square)
    ldi rPage,15
1001 ldi rColumn,48
    rcall CharOut
1003
    rcall delay1s
1005
    ret
1007
;-----------------------------------------------------------------------
1009
;*********************************************************************
1011 ;Subroutine initialising the DOGXL-LCD
;*********************************************************************
1013 fcInitGraphik_DOGLCD:
  ;Delay to give the screen time to power up
1015 rcall mDelay100ms
  ;Initialisation of the LCD-Port. Reset is LOW
1017 LDI   rTemp1, cInitPortDOGLCD
  OUT   DDRDOGLCD, rTemp1
1019 CBI   PORTDOGLCD, PinReset
  CBI   PORTDOGLCD, PinSDA
1021 CBI   PORTDOGLCD, PinCD
  SBI   PORTDOGLCD, PinSCK
1023 CBI   PORTDOGLCD, PinCS
  rcall mDelay1ms
1025
  ;Initialisation through the reset. This was taken from the DOGM 128-W-
    LCD
1027 ;initialisation sequence (cf. datasheet ST7565R, pg 65). Without this
    sequence
  ; the display doesn't start correctly.
1029 SBI   PORTDOGLCD, PinReset
  rcall mDelay1ms
1031 CBI   PORTDOGLCD, PinReset
  rcall mDelay10us
1033 SBI   PORTDOGLCD, PinReset
  rcall mDelay10us
1035
  ;Initialisierung gem  Tabelle Datenblatt EA DOGXL160, S. 6
```

```
1037    LDI     rByteLCD, 0xF1          ;Set last COM-Electrode to 104-1
        RCALL   fcSendCommandToLCD
1039    LDI     rByteLCD, 0x67          ;
        RCALL   fcSendCommandToLCD
1041    LDI     rByteLCD, 0xC0          ;Set Mapping control
        RCALL   fcSendCommandToLCD
1043    LDI     rByteLCD, 0x40          ;Set Scroll line LSB
        RCALL   fcSendCommandToLCD
1045    LDI     rByteLCD, 0x50          ;Set Scroll line MSB
        RCALL   fcSendCommandToLCD
1047    LDI     rByteLCD, 0x2B          ;Set Panel Loading
        RCALL   fcSendCommandToLCD
1049    LDI     rByteLCD, 0xEB          ;Set LCD Bias Ratio
        RCALL   fcSendCommandToLCD
1051    LDI     rByteLCD, 0x81          ;Set Vbias Potentiometer
        RCALL   fcSendCommandToLCD
1053    LDI     rByteLCD, 0x5f          ;Set Vbias Potentiometer
        RCALL   fcSendCommandToLCD
1055    LDI     rByteLCD, 0x89          ;Set RAM address control = auto increment
        RCALL   fcSendCommandToLCD
1057    LDI     rByteLCD, 0xAF          ;Display enable
        RCALL   fcSendCommandToLCD
1059    rcall mDelay1ms

1061    ;Erase display
        RCALL fcEraseDisplay
1063
        ret
1065 ;
```

```
1067 ;*****************************************************************
     ;Subroutine sends the address to the LCD.
1069 ;The values in the registers rColumn and rPage are given
     ;rPage, rColumn and rByteLCD are sent back unchanged
1071 ;*****************************************************************
     fcSetCurrentLCDAddress:
1073    ;Save registers
       PUSH   rPage
1075   PUSH   rColumn
       PUSH   rByteLCD
1077
       ;rColumn contains column address (0 - 160).
1079   ;rColumn needs to be used twice
       PUSH   rColumn
1081   LDI    rByteLCD, cSetColAddrMSB
       ;Die 4 highest bits must be transferred to the lowest Nibble
1083   ;The four highest bits are set to 00
       SWAP   rColumn
1085   ANDI   rColumn, 0b00001111
       ;Send the highest nibble to the LCD.
1087   ADD    rByteLCD, rColumn
       RCALL fcSendCommandToLCD
1089   ;Pop rColumn and send the 4 lower bits to the LCD
```

```
    POP     rColumn
1091 LDI     rByteLCD, cSetColAddrLSB
    ANDI    rColumn, 0b00001111
1093 ADD     rByteLCD, rColumn
    RCALL fcSendCommandToLCD
1095 ;Send page
    LDI     rByteLCD, cSetPageAddress
1097 ADD     rByteLCD, rPage
    RCALL fcSendCommandToLCD
1099
    POP     rByteLCD
1101 POP     rColumn
    POP     rPage
1103 ret
;-----------------------------------------------------------------------
1105
.include "characters_and_pieces.asm"
1107 ;*********************************************************************
; Subroutine which outputs a 12*12 block to the screen by looking up in
1109 ; a database characters_and_pieces.asm
;*********************************************************************
1111
CharOut:
1113         push rTemp1
             push rTemp2
1115         push rTemp3
             push rPage
1117         push rColumn
             push ZH
1119         push ZL

1121               mov rTemp1,rPage
             subi rTemp1,-3  ; Set maximum to rPage +3 (3*4 pixels)
1123
             mov rTemp2,rColumn
1125         mov rTemp3,rColumn
             subi rTemp3,-12 ; Set maximum to rColumn +12 (12 pixels)
1127
             ; Output the character here
1129         LoopPageChar:
             mov rColumn,rTemp2
1131
               LoopColChar:
1133           lpm rByteLCD , Z+ ; Point to the character that is to be
    outputted
                          ; and increment pointer
1135           rcall fcSetCurrentLCDAddress
               rcall fcSendDataToLCD
1137           inc   rColumn
               cp    rColumn, rTemp3
1139           brne  LoopColChar ; Only leave loop if column maximum is
    reached

1141         inc   rPage
             cp    rPage, rTemp1
```

```
1143          brlo   LoopPageChar ; Only leave loop if page max is reached

1145          pop rTemp1
              pop rTemp2
1147          pop rTemp3
              pop rPage
1149            pop rColumn
                pop ZH
1151            pop ZL

1153          ret
              ;*****************************************************************
1155          ; Subroutine which sends a byte to the LCD from fcSendData/CommandToLCD
              ; starting with the MSB. The data is sent on the rising edge of
1157          ; the clock. The byte sent to the display is contained in rByteLCD.
              ;*****************************************************************
1159          fcByteToLCD:
                PUSH   rByteLCD
1161            PUSH   rCount

1163            LDI    rCount, 8
              LoopBitToLCD:
1165            rcall mDelay5us
                CBI    PORTDOGLCD, PinSCK ; Clear clock bit
1167            CBI    PORTDOGLCD, PinSDA ; Clear data bit
                LSL    rByteLCD
1169            BRCC   LoopBit ; go to loop bit if carry cleared
                SBI    PORTDOGLCD, PinSDA ; else set bit on data pin
1171          LoopBit:
                rcall mDelay10us
1173            SBI    PORTDOGLCD, PinSCK ; send bit to screen by setting clock pin
                DEC    rCount
1175            CPI    rCount, 0
                BRNE   LoopBitToLCD ; do for all 8 bits
1177
              EndByteToLCD:
1179            rcall mDelay5us
                CBI    PORTDOGLCD, PinSCK
1181
                POP    rCount
1183            POP    rByteLCD

1185            RET
              ;---------------------------------------------------------------------
1187
              ;*****************************************************************
1189          ;Subroutine which sends a command to the display
              ;The byte is contained in rByteLCD
1191          ;*****************************************************************
              fcSendCommandToLCD:
1193            CBI    PORTDOGLCD, PinCS    ;Chip select
                rcall mDelay10us
1195            CBI    PORTDOGLCD, PinCD
                rcall mDelay10us
1197            RCALL fcByteToLCD
```

```
    rcall mDelay10us
1199  SBI   PORTDOGLCD, PinCS    ;Chip "deselect"

1201  RET
;-----------------------------------------------------------------------

;*********************************************************************
;Subroutine which sends a data byte to the LCD
;The byte is contained in rByteLCD
;*********************************************************************
fcSendDataToLCD:
1209  CBI   PORTDOGLCD, PinCS    ;Chip select
    rcall mDelay5us
1211  SBI   PORTDOGLCD, PinCD
    rcall mDelay5us
1213  RCALL fcByteToLCD
    rcall mDelay5us
1215  SBI   PORTDOGLCD, PinCS    ;Chip "deselct"

1217  RET
;-----------------------------------------------------------------------
1219


1221 ;*********************************************************************
;Erase Diplay
1223 ;*********************************************************************
fcEraseDisplay:
1225  CLR   rPage

1227 LoopErase_1:
  CLR   rColumn
1229  RCALL fcSetCurrentLCDAddress
LoopErase_2:
1231  CLR   rByteLCD
  RCALL fcSendDataToLCD
1233  INC   rColumn
  CPI   rColumn, cNumColumnsLCD
1235  BRNE  LoopErase_2

1237  INC   rPage
  CPI   rPage, cNumPagesLCD
1239
  BRNE  LoopErase_1
1241
EndEraseDisplay:
1243  RET
;-----------------------------------------------------------------------
1245
;*********************************************************************
1247 ;Check if the square has an odd or even address (sum of the coordinates)
;*********************************************************************
1249
odd_or_even:
1251 push rTemp2
```

```
1253 push r17
     andi r17, 0b00001111
1255 mov rTemp2, r17
     pop r17
1257 lsr r17
     lsr r17
1259 lsr r17
     lsr r17
1261 add r17,rTemp2
     andi r17,$01
1263 cpi r17,1 ;skip if sum is even
     breq end_bin
1265
     ldi r17,$00
1267 pop rTemp2
     ret
1269
     end_bin:
1271 ldi r17,$10
     pop rTemp2
1273 ret
```

```
;Character and pieces
;Hugo Jeanperrin
;27.02.15


;*****************************************************************
; Code for character 1
;*****************************************************************

Num1:
.db 0b00000000, 0b00000000,0b00000000, 0b11000000, 0b00110000, 0b11111100
.db 0b00000000, 0b00000000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000,0b00000000,0b11111111,0b00000000,0b00000000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00110000, 0b00110000, 0b00111111, 0b00110000, 0b00110000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0


;*****************************************************************
; Code for character 2
;*****************************************************************

Num2:
.db 0b00000000, 0b00000000,0b00000000, 0b00110000, 0b00001100, 0b00001100
.db 0b00001100, 0b11110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000,0b11000000,0b00110000,0b00001100,0b00000011
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00001111, 0b00001100, 0b00001100, 0b00001100, 0b00001100
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character 3
;*****************************************************************

Num3:
.db 0b00000000, 0b00000000,0b00000000, 0b00110000, 0b00001100, 0b00001100
.db 0b00001100, 0b11110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000,0b00001100,0b00001100,0b00001100,0b11110011
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000011, 0b00001100, 0b00001100, 0b00001100, 0b00000011
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0


;*****************************************************************
; Code for character 4
;*****************************************************************

Num4:
.db 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b11000000, 0b00110000
.db 0b11111100, 0b00000000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00111111,0b00110000,0b00110000,0b11111111,0b00110000
```

```
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00001111, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character 5
;*****************************************************************

Num5:
.db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00001100, 0b00001100
.db 0b00001100, 0b00001100
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00001111,0b00001100,0b00001100,0b00001100,0b11110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000011, 0b00001100, 0b00001100, 0b00001100, 0b00000011
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character 6
;*****************************************************************

Num6:
.db 0b00000000, 0b00000000,0b00000000, 0b11000000, 0b00110000, 0b00001100
.db 0b00001100, 0b00000000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b11111111,0b00001100,0b00001100,0b00001100,0b11110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000011, 0b00001100, 0b00001100, 0b00001100, 0b00000011
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character 7
;*****************************************************************

Num7:
.db 0b00000000, 0b00000000,0b00000000, 0b00001100, 0b00001100, 0b00001100
.db 0b00001100, 0b11111100
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000,0b11110000,0b00001100,0b00000011,0b00000000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00001111, 0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character 8
;*****************************************************************

Num8:
.db 0b00000000, 0b00000000,0b00000000, 0b11110000, 0b00001100, 0b00001100
.db 0b00001100, 0b11110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b11110011,0b00001100,0b00001100,0b00001100,0b11110011
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000011, 0b00001100, 0b00001100, 0b00001100, 0b00000011
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0
```

```
111

113 ;*****************************************************************
    ; Code for character a
115 ;*****************************************************************

117 Numa:
    .db 0b00000000, 0b00000000,0b00000000, 0b11000000, 0b00110000, 0b00001100
119 .db 0b00110000, 0b11000000
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
121 .db 0b00000000,0b11111111,0b00001100,0b00001100,0b00001100,0b11111111
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
123 .db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
    .db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0
125

127 ;*****************************************************************
    ; Code for character b
129 ;*****************************************************************

131 Numb:
    .db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00001100, 0b00001100
133 .db 0b00001100, 0b11110000
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
135 .db 0b00000000,0b11111111,0b11000011,0b11000011,0b11000011,0b00111111
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
137 .db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
    .db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0
139

    ;*****************************************************************
141 ; Code for character c
    ;*****************************************************************
143

    Numc:
145 .db 0b00000000, 0b00000000,0b00000000, 0b11110000, 0b00001100, 0b00001100
    .db 0b00001100, 0b00110000
147 .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
    .db 0b00000000,0b00111111,0b11000000,0b11000000,0b11000000,0b00110000
149 .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
    .db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
151 .db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

153 ;*****************************************************************
    ; Code for character d
155 ;*****************************************************************

157 Numd:
    .db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00001100, 0b00001100
159 .db 0b00110000, 0b11000000
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
161 .db 0b00000000,0b11111111,0b11000000,0b11000000,0b00110000,0b00001111
    .db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
163 .db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
    .db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0
165
```

```
;*****************************************************************
; Code for character e
;*****************************************************************

Nume:
.db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00001100, 0b00001100
.db 0b00001100, 0b00001100
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b11111111,0b11000011,0b11000011,0b11000011,0b11000011
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character f
;*****************************************************************

Numf:
.db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00001100, 0b00001100
.db 0b00001100, 0b00001100
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b11111111,0b00000011,0b00000011,0b00000011,0b00000011
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character g
;*****************************************************************

Numg:
.db 0b00000000, 0b00000000,0b00000000, 0b11110000, 0b00001100, 0b00001100
.db 0b00001100, 0b00110000
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00111111,0b11000000,0b11000011,0b11000011,0b00111111
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for character h
;*****************************************************************

Numh:
.db 0b00000000, 0b00000000,0b00000000, 0b11111100, 0b00000000, 0b00000000
.db 0b00000000, 0b11111100
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b11111111,0b00000011,0b00000011,0b00000011,0b11111111
.db 0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000,0b00000000
.db 0b00000000,0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
.db 0b00000000,0b00000000, 0b00000000,0b00000000,$02,0

;*****************************************************************
; Code for white pawn black background
;*****************************************************************
```

```
221
    white_pawn_black_square:
223 .db 0b01010101, 0b01010101,0b01010101, 0b01010101, 0b00010101, 0b00010101
    .db 0b00010101, 0b01010101
225 .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
    .db 0b01010101,0b01010000,0b01001111,0b00111111,0b01001111,0b01010000
227 .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
    .db 0b01010101,0b01010100, 0b01010100, 0b01010100, 0b01010100, 0b01010100
229 .db 0b01010101,0b01010101, 0b01010101,0b01010101,$02,0

231
    ;*****************************************************************
233 ; Code for white pawn white background
    ;*****************************************************************
235
    white_pawn_white_square:
237 .db 0b10101010, 0b10101010,0b10101010, 0b10101010, 0b00101010, 0b00101010
    .db 0b00101010, 0b10101010
239 .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
    .db 0b10101010,0b10100000,0b10001111,0b00111111,0b10001111,0b10100000
241 .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
    .db 0b10101010,0b10101000, 0b10101000, 0b10101000, 0b10101000, 0b10101000
243 .db 0b10101010,0b10101010, 0b10101010,0b10101010,$02,0

245 ;*****************************************************************
    ; Code for black pawn black background
247 ;*****************************************************************

249 black_pawn_black_square:
    .db 0b01010101, 0b01010101,0b01010101, 0b01010101, 0b00010101, 0b00010101
251 .db 0b00010101, 0b01010101
    .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
253 .db 0b01010101,0b01010000,0b01000000,0b00000000,0b01000000,0b01010000
    .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
255 .db 0b01010101,0b01010100, 0b01010100, 0b01010100, 0b01010100, 0b01010100
    .db 0b01010101,0b01010101, 0b01010101,0b01010101,$02,0
257

259 ;*****************************************************************
    ; Code for black pawn white background
261 ;*****************************************************************

263 black_pawn_white_square:
    .db 0b10101010, 0b10101010,0b10101010, 0b10101010, 0b00101010, 0b00101010
265 .db 0b00101010, 0b10101010
    .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
267 .db 0b10101010,0b10100000,0b10000000,0b00000000,0b10000000,0b10100000
    .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
269 .db 0b10101010,0b10101000, 0b10101000, 0b10101000, 0b10101000, 0b10101000
    .db 0b10101010,0b10101010, 0b10101010,0b10101010,$02,0
271
    ;*****************************************************************
273 ; Code for white bishop white background
    ;*****************************************************************
275
```

```
white_bishop_white_square:
.db 0b10101010, 0b10101010,0b10101010, 0b00100010, 0b10000010, 0b10100010
.db 0b10000010, 0b00100010
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
.db 0b10101000,0b10100010,0b10001010,0b00101010,0b10001010,0b10100010
.db 0b10101000,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
.db 0b10100010,0b10100000, 0b10100000, 0b10100000, 0b10100000, 0b10100000
.db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0

;*****************************************************************
; Code for black bishop white background
;*****************************************************************

black_bishop_white_square:
.db 0b10101010, 0b10101010,0b10101010, 0b00100010,0b00000010,0b00000010
.db 0b00000010, 0b00100010
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
.db 0b10101000,0b10100000,0b10000000,0b00000000,0b10000000,0b10100000
.db 0b10101000,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
.db 0b10100010,0b10100000, 0b10100000, 0b10100000, 0b10100000, 0b10100000
.db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0

;*****************************************************************
; Code for black bishop black background
;*****************************************************************

black_bishop_black_square:
.db 0b01010101, 0b01010101,0b01010101, 0b00010001,0b00000001,0b00000001
.db 0b00000001, 0b00010001
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
.db 0b01010100,0b01010000,0b01000000,0b00000000,0b01000000,0b01010000
.db 0b01010100,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
.db 0b01010001,0b01010000, 0b01010000, 0b01010000, 0b01010000, 0b01010000
.db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0

;*****************************************************************
; Code for white bishop black background
;*****************************************************************

white_bishop_black_square:
.db 0b01010101, 0b01010101,0b01010101, 0b00010001,0b11000001,0b11110001
.db 0b11000001, 0b00010001
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
.db 0b01010100,0b01010011,0b01001111,0b00111111,0b01001111,0b01010011
.db 0b01010100,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
.db 0b01010001,0b01010000, 0b01010000, 0b01010000, 0b01010000, 0b01010000
.db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0

;*****************************************************************
; Code for white knight black background
;*****************************************************************

white_knight_black_square:
.db 0b01010101, 0b01010101,0b01010101, 0b00010101,0b11000101,0b11110001
.db 0b11110001, 0b00000001
```

```
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
.db 0b01010000,0b01001111,0b001010011,0b00000011,0b11111111,0b00000000
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
.db 0b01010101,0b01010101, 0b01010001, 0b01010000, 0b01010011, 0b01010000
.db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0

;*****************************************************************
; Code for black knight black background
;*****************************************************************

black_knight_black_square:
.db 0b01010101, 0b01010101,0b01010101, 0b00010101,0b00000101,0b00000001
.db 0b00000001, 0b00000001
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
.db 0b01010000,0b01000000,0b001010000,0b00000000,0b00000000,0b00000000
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
.db 0b01010101,0b01010101, 0b01010001, 0b01010000, 0b01010000, 0b01010000
.db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0

;*****************************************************************
; Code for black knight white background
;*****************************************************************

black_knight_white_square:
.db 0b10101010, 0b10101010,0b10101010, 0b00101010,0b00001010,0b00000010
.db 0b00000010, 0b00000010
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
.db 0b10100000,0b10000000,0b010100000,0b00000000,0b00000000,0b00000000
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
.db 0b10101010,0b10101010, 0b10100010, 0b10100000, 0b10100000, 0b10100000
.db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0

;*****************************************************************
; Code for white knight white background
;*****************************************************************

white_knight_white_square:
.db 0b10101010, 0b10101010,0b10101010, 0b00101010,0b11001010,0b11110010
.db 0b11110010, 0b00000010
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
.db 0b10100000,0b10001111,0b10100011,0b00001111,0b11111111,0b00000000
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
.db 0b10101010,0b10101010, 0b10100010, 0b10100000, 0b10100011, 0b10100000
.db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0

;*****************************************************************
; Code for white tower white background
;*****************************************************************

white_tower_white_square:
.db 0b10101010, 0b10101010,0b10001010, 0b00101010,0b00001010,0b00101010
.db 0b00001010, 0b00101010
.db 0b10001010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
.db 0b10101010,0b00000000,0b11111111,0b11111111,0b11111111,0b00000000
.db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
```

```
    .db 0b10100010,0b10100000, 0b10100011, 0b10100011, 0b10100011, 0b10100000
387 .db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0

389 ;*****************************************************************
    ; Code for black tower white background
391 ;*****************************************************************

393 black_tower_white_square:
    .db 0b10101010, 0b10101010,0b10001010, 0b00101010,0b00001010,0b00101010
395 .db 0b00001010, 0b00101010
    .db 0b10001010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
397 .db 0b10101010,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000
    .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010
399 .db 0b10100010,0b10100000, 0b10100000, 0b10100000, 0b10100000, 0b10100000
    .db 0b10100010,0b10101010, 0b10101010,0b10101010,$02,0
401
    ;*****************************************************************
403 ; Code for black tower black background
    ;*****************************************************************
405
    black_tower_black_square:
407 .db 0b01010101, 0b01010101,0b01000101, 0b00010101,0b00000101,0b00010101
    .db 0b00000101, 0b00010101
409 .db 0b01000101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
    .db 0b01010101,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000
411 .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
    .db 0b01010001,0b01010000, 0b01010000, 0b01010000, 0b01010000, 0b01010000
413 .db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0

415 ;*****************************************************************
    ; Code for white tower black background
417 ;*****************************************************************

419 white_tower_black_square:
    .db 0b01010101, 0b01010101,0b01000101, 0b00010101,0b00000101,0b00010101
421 .db 0b00000101, 0b00010101
    .db 0b01000101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
423 .db 0b01010101,0b00000000,0b11111111,0b11111111,0b11111111,0b00000000
    .db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101
425 .db 0b01010001,0b01010000, 0b01010011, 0b01010011, 0b01010011, 0b01010000
    .db 0b01010001,0b01010101, 0b01010101,0b01010101,$02,0
427
    ;*****************************************************************
429 ; Code for white queen white background
    ;*****************************************************************
431
    white_queen_white_square:
433 .db 0b10101010, 0b00001010,0b11001010, 0b00101010,0b00101010,0b10101010
    .db 0b00101010, 0b00101010
435 .db 0b11001010,0b00001010,0b10101010,0b10101010, 0b10101010, 0b10100000
    .db 0b10001111,0b00111111,0b00111111,0b11111100,0b00111111, 0b00111111
437 .db 0b10001111,0b10100000,0b10101010, 0b10101010,0b10101010,0b10101010
    .db 0b10100010, 0b10100010, 0b10100010, 0b10100000, 0b10100010, 0
        b10100010
439 .db 0b10100010, 0b10101010,0b10101010, 0b10101010,$02,0
```

```
441 ;*****************************************************************
    ; Code for black queen white background
443 ;*****************************************************************

445 black_queen_white_square:
    .db 0b10101010, 0b00001010,0b00001010, 0b00101010,0b00101010,0b10101010
447 .db 0b00101010, 0b00101010
    .db 0b00001010,0b00001010,0b10101010,0b10101010, 0b10101010, 0b10100000
449 .db 0b10000000,0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000
    .db 0b10000000,0b10100000,0b10101010, 0b10101010,0b10101010,0b10101010
451 .db 0b10100010, 0b10100010, 0b10100010, 0b10100000, 0b10100010, 0
        b10100010
    .db 0b10100010, 0b10101010,0b10101010, 0b10101010,$02,0
453

    ;*****************************************************************
455 ; Code for black queen black background
    ;*****************************************************************
457

    black_queen_black_square:
459 .db 0b01010101, 0b00000101,0b00000101, 0b00010101,0b00010101,0b01010101
    .db 0b00010101, 0b00010101
461 .db 0b00000101,0b00000101,0b01010101,0b01010101, 0b01010101, 0b01010000
    .db 0b01000000,0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000
463 .db 0b01000000,0b01010000,0b01010101, 0b01010101,0b01010101,0b01010101
    .db 0b01010001, 0b01010001, 0b01010001, 0b01010000, 0b01010001, 0
        b01010001
465 .db 0b01010001, 0b01010101,0b01010101, 0b01010101,$02,0

467 ;*****************************************************************
    ; Code for white queen black background
469 ;*****************************************************************

471 white_queen_black_square:
    .db 0b01010101, 0b00000101,0b11000101, 0b00010101,0b00010101,0b01010101
473 .db 0b00010101, 0b00010101
    .db 0b11000101,0b00000101,0b01010101,0b01010101, 0b01010101, 0b01010000
475 .db 0b01001111,0b00111111,0b00111111,0b11111100,0b00111111, 0b00111111
    .db 0b01001111,0b01010000,0b01010101, 0b01010101,0b01010101,0b01010101
477 .db 0b01010001, 0b01010001, 0b01010001, 0b01010000, 0b01010001, 0
        b01010001
    .db 0b01010001, 0b01010101,0b01010101, 0b01010101,$02,0
479

    ;*****************************************************************
481 ; Code for white king black background
    ;*****************************************************************
483

    white_king_black_square:
485 .db 0b01010101, 0b01010101,0b00010101, 0b01010101,0b00010101,0b01010101
    .db 0b00010101, 0b01010101
487 .db 0b00010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
    .db 0b00000000,0b00111100,0b00111111,0b00111100,0b00111111, 0b00111100
489 .db 0b00000000,0b01010101,0b01010101, 0b01010101,0b01010101,0b01010101
    .db 0b01010001, 0b01010001, 0b01010001, 0b01010000, 0b01010001, 0
        b01010001
```

```
491 .db 0b01010001, 0b01010101,0b01010101, 0b01010101,$02,0

493 ;*****************************************************************
    ; Code for black king black background
495 ;*****************************************************************

497 black_king_black_square:
    .db 0b01010101, 0b01010101,0b00010101, 0b01010101,0b00010101,0b01010101
499 .db 0b00010101, 0b01010101
    .db 0b00010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
501 .db 0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000
    .db 0b00000000,0b01010101,0b01010101, 0b01010101,0b01010101,0b01010101
503 .db 0b01010001, 0b01010001, 0b01010001, 0b01010000, 0b01010001, 0
        b01010001
    .db 0b01010001, 0b01010101,0b01010101, 0b01010101,$02,0
505

    ;*****************************************************************
507 ; Code for black king white background
    ;*****************************************************************
509

    black_king_white_square:
511 .db 0b10101010, 0b10101010,0b00101010, 0b10101010,0b00101010,0b10101010
    .db 0b00101010, 0b10101010
513 .db 0b00101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
    .db 0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, 0b00000000
515 .db 0b00000000,0b10101010,0b10101010, 0b10101010,0b10101010,0b10101010
    .db 0b10100010, 0b10100010, 0b10100010, 0b10100000, 0b10100010, 0
        b10100010
517 .db 0b10100010, 0b10101010,0b10101010, 0b10101010,$02,0

519 ;*****************************************************************
    ; Code for white king white background
521 ;*****************************************************************

523 white_king_white_square:
    .db 0b10101010, 0b10101010,0b00101010, 0b10101010,0b00101010,0b10101010
525 .db 0b00101010, 0b10101010
    .db 0b00101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
527 .db 0b00000000,0b00111100,0b00111111,0b00111100,0b00111111, 0b00111100
    .db 0b00000000,0b10101010,0b10101010, 0b10101010,0b10101010,0b10101010
529 .db 0b10100010, 0b10100010, 0b10100010, 0b10100000, 0b10100010, 0
        b10100010
    .db 0b10100010, 0b10101010,0b10101010, 0b10101010,$02,0
531

    ;*****************************************************************
533 ; Code for an empty white space
    ;*****************************************************************
535

    empty_white_square:
537 .db 0b10101010, 0b10101010,0b10101010, 0b10101010,0b10101010,0b10101010
    .db 0b10101010, 0b10101010
539 .db 0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010, 0b10101010
    .db 0b10101010,0b10101010,0b10101010,0b10101010,0b10101010, 0b10101010
541 .db 0b10101010,0b10101010,0b10101010, 0b10101010,0b10101010,0b10101010
    .db 0b10101010, 0b10101010, 0b10101010, 0b10101010, 0b10101010, 0
```

```
    b10101010
.db 0b10101010, 0b10101010,0b10101010, 0b10101010,$02,0

;*****************************************************************
; Code for an empty black space
;*****************************************************************

empty_black_square:
.db 0b01010101, 0b01010101,0b01010101, 0b01010101,0b01010101,0b01010101
.db 0b01010101, 0b01010101
.db 0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101, 0b01010101
.db 0b01010101,0b01010101,0b01010101,0b01010101,0b01010101, 0b01010101
.db 0b01010101,0b01010101,0b01010101, 0b01010101,0b01010101,0b01010101
.db 0b01010101, 0b01010101, 0b01010101, 0b01010101, 0b01010101, 0
    b01010101
.db 0b01010101, 0b01010101,0b01010101, 0b01010101,$02,0
```

```
;*****************************************************************************
;
;*****************************************************************************
;************************ Alphanumeric Display Initialisation
;*********************************
;*****************************************************************************
;
;*****************************************************************************
; Display Initialization routine
;
; This is just the inialisation routine provided in the Microprocessors
    lecture notes/files
;
;
;*****************************************************************************
alphanum_lcd_initialisation:
    CALL DEL15ms                    ; wait 15ms for things to relax after
    power up
    ldi r16,    $30                 ; Hitachi says do it...
    sts   $8000,r16                   ; so i do it....
    CALL DEL4P1ms                   ; Hitachi says wait 4.1 msec
    sts   $8000,r16                 ; and again I do what I'm told
    call DEL100mus                    ; wait 100 mus
    sts   $8000,r16                 ; here we go again folks
                call busylcd
    ldi r16, $3F                  ; Function Set : 2 lines + 5x7 Font
    sts  $8000,r16
                call busylcd
    ldi r16,  $08                 ;display off
    sts  $8000, r16
                call busylcd
    ldi r16,  $01                 ;display on
    sts  $8000,  r16
                call busylcd
                ldi r16, $38          ;function set
    sts  $8000, r16
    call busylcd
    ldi r16, $0E                  ;display on
    sts  $8000, r16
    call busylcd
    ldi r16, $06                      ;entry mode set increment no shift
    sts  $8000,  r16
                call busylcd
                clr r16
        ret
;
;
;*****************************************************************************
```

```
; This clears the display so we can start all over again
;
CLRDIS:
      push r16
    ldi r16,$01    ; Clear Display and send cursor
    sts $8000,r16    ; to the most left position
    call busylcd
    pop r16
        ret
;
;
    ********************************************************************************

; A routine the probes the display BUSY bit
;

;busylcd:
;
;Revised 1/2/05 according to comment from John Jones and
; Nicolas Osman
;
busylcd:
    push r16
    busylcdloop:
    lds r16, $8000    ;access
        sbrc r16, 7       ;check busy bit  7
        rjmp busylcdloop
        call DEL100mus
    pop r16
    ret               ;return if clear
;
;
;
    ********************************************************************************
```

```
;
    ********************************************************************

;************************ Change colour
    *********************************
;
    ********************************************************************

; Author: Richard Flint
; Date: 27/02/2015
;
    ********************************************************************

; A short routine that changes the colour of the piece to be moved.
; This is used in checking whether the movement is valid.
;
; This is not a stand alone code. It must be included in the main
    program.
;
    ********************************************************************

change_colour:
cpi r25,$01         ;Has white just moved?
breq change_to_black  ;If so, then change to black
ldi r25,$01         ;If not, then change to white
ret
;
change_to_black:
ldi r25,$10         ;Change to black
ret
;
;
    ********************************************************************
```

```
;
    ************************************************************************************************
;
    ************************************************************************************************
;**************************** Chess Board Initialisation
    ***********************************
;
    ************************************************************************************************
;
    ************************************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;
    ************************************************************************************************
; This initialises the chess board either as a new game, or loads the
    positions of previous game
;
    ************************************************************************************************
chess_board_initialisation:
push r20
;
rcall CLRDIS
rcall MessLoadOut           ;"Do you want to load previous game? (Y/N)"
loading_game_loop:            ;Loop until user presses either [Y] or [N]
  in r20,pind
  com r20
  cpi r20,$10             ;If presses $10 = N
  breq chess_piece_positions_new_game ;branch to new game positions
  cpi r20,$08             ;If presses $08 = Y
  brne loading_game_loop        ;branch to load previous game
  jmp load_previous_game

chess_piece_positions_new_game:     ;new game routine places pieces in
    starting positions
ldi r25,$01   ;white to play first
;
        ;black pieces at the bottom
ldi r20,$C1
sts $0112,r20 ;black pawn
sts $0122,r20 ;black pawn
sts $0132,r20 ;black pawn
sts $0142,r20 ;black pawn
sts $0152,r20 ;black pawn
sts $0162,r20 ;black pawn
sts $0172,r20 ;black pawn
sts $0182,r20 ;black pawn
;
ldi r20,$C4
sts $0111,r20 ;black castle
```

```
sts $0181,r20 ;black castle
;
ldi r20,$C2
sts $0121,r20 ;black knight
sts $0171,r20 ;black knight
;
ldi r20,$C3
sts $0131,r20 ;black bishop
sts $0161,r20 ;black bishop
;
ldi r20,$C5
sts $0151,r20 ;black queen
;
ldi r20,$C6
sts $0141,r20 ;black king
;
; White pieces at the top
ldi r20,$81
sts $0117,r20 ;white pawn
sts $0127,r20 ;white pawn
sts $0137,r20 ;white pawn
sts $0147,r20 ;white pawn
sts $0157,r20 ;white pawn
sts $0167,r20 ;white pawn
sts $0177,r20 ;white pawn
sts $0187,r20 ;white pawn
;
ldi r20,$84
sts $0118,r20 ;white castle
sts $0188,r20 ;white castle
;
ldi r20,$82
sts $0128,r20 ;white knight
sts $0178,r20 ;white knight
;
ldi r20,$83
sts $0138,r20 ;white bishop
sts $0168,r20 ;white bishop
;
ldi r20,$85
sts $0158,r20 ;white queen
;
ldi r20,$86
sts $0148,r20 ;white king

; the rest of the spaces clear (this is just to make sure!)
ldi r20,$00
sts $0113,r20
sts $0114,r20
sts $0115,r20
sts $0116,r20
sts $0123,r20
sts $0124,r20
sts $0125,r20
sts $0126,r20
```

```
96  sts $0133,r20
    sts $0134,r20
98  sts $0135,r20
    sts $0136,r20
100 sts $0143,r20
    sts $0144,r20
102 sts $0145,r20
    sts $0146,r20
104 sts $0153,r20
    sts $0154,r20
106 sts $0155,r20
    sts $0156,r20
108 sts $0163,r20
    sts $0164,r20
110 sts $0165,r20
    sts $0166,r20
112 sts $0173,r20
    sts $0174,r20
114 sts $0175,r20
    sts $0176,r20
116 sts $0183,r20
    sts $0184,r20
118 sts $0185,r20
    sts $0186,r20
120 ;
    pop r20
122 rcall CLRDIS
    rcall MessNewGameStartOut    ;"New game.Press enter to continue"
124 ;
    push r20            ;Loop until user presses [enter]
126 startloop:
    in r20,pind
128 com r20
    cpi r20,$02
130 brne startloop
    pop r20
132 ret
    ;
134 load_previous_game:
    pop r20        ;saved game will still be in SRAM so don't need to do
        anything
136 ret
    ;
```

```
;*****************************************************************************
;*****************************************************************************
;**************************** Chess Board Initialisation *********************
;*****************************************************************************
;*****************************************************************************
;
;Initialises the chess board with pieces scattered about. It is used in debugging and checking
;the results of the code
;
;*****************************************************************************
;
chess_board_initialisation_2:
push r20
;
rcall CLRDIS
;
chess_piece_positions_new_game_2:
ldi r20,$00
sts $0111,r20
sts $0112,r20
sts $0113,r20
ldi r20,$81
sts $0114,r20 ;white pawn
ldi r20,$00
sts $0115,r20
sts $0116,r20
ldi r20,$84
sts $0117,r20 ;white castle
ldi r20,$00
sts $0118,r20
;
sts $0121,r20
sts $0122,r20
ldi r20,$C1
sts $0123,r20 ;black pawn
ldi r20,$00
sts $0124,r20
sts $0125,r20
sts $0126,r20
sts $0127,r20
sts $0128,r20
;
sts $0131,r20
```

```
sts $0132,r20
sts $0133,r20
ldi r20,$81
sts $0134,r20 ;white pawn
ldi r20,$00
sts $0135,r20
sts $0136,r20
sts $0137,r20
sts $0138,r20
;
sts $0141,r20
ldi r20,$C6
sts $0142,r20 ;black king
ldi r20,$00
sts $0143,r20
sts $0144,r20
sts $0145,r20
sts $0146,r20
sts $0147,r20
sts $0148,r20
;
sts $0151,r20
ldi r20,$C5
sts $0152,r20 ;black queen
ldi r20,$00
sts $0153,r20
ldi r20,$85
sts $0154,r20
ldi r20,$00
sts $0155,r20
sts $0156,r20
sts $0157,r20
sts $0158,r20
;
ldi r20,$C3
sts $0161,r20 ;black bishop
ldi r20,$00
sts $0162,r20
sts $0163,r20
sts $0164,r20
sts $0165,r20
sts $0166,r20
sts $0167,r20
sts $0168,r20
;
sts $0171,r20
sts $0172,r20
sts $0173,r20
ldi r20,$C2
sts $0174,r20 ;black knight
ldi r20,$00
sts $0175,r20
sts $0176,r20
sts $0177,r20
sts $0178,r20
```

```
99  ;
    sts $0181,r20
101 sts $0182,r20
    sts $0183,r20
103 sts $0184,r20
    sts $0185,r20
105 ldi r20,$C4
    sts $0186,r20 ;black castle
107 ldi r20,$00
    sts $0187,r20
109 sts $0188,r20
    ;
111 pop r20
    ;
113 ret
```

```
;*****************************************************************************
;
;*****************************************************************************
;********************************** DELAY ROUTINES
;*****************************************
;
;*****************************************************************************
;
;*****************************************************************************
;These delay routines come from the sample code for the Microprocessor
    course. It is possible
;that the delay times do not match the time estimation given. For example
    , the routine Del49ms
;may not match a 149ms delay. This is because the delay codes were
    written for a different
;microprossor. There is no requirement for strict timing/delays in the
    coding of the chess rules,
;so any discrepancy is not very important.
;
;*****************************************************************************
BigDEL:
                rcall Del49ms
                rcall Del49ms
                rcall Del49ms
                rcall Del49ms
                rcall Del49ms
                ret
BigBigDEL:
                rcall BigDEL
                rcall BigDEL
        rcall BigDEL
        rcall BigDEL
        rcall BigDEL
        rcall BigDEL
                ret
;
DEL15ms:
       push XH
       push XL

                LDI XH, HIGH(19997)
                LDI XL, LOW (19997)
COUNT:
                SBIW XL, 1
                BRNE COUNT
;
       pop XL
       pop XH
                RET
```

```
41 ;
   DEL4P1ms:
43                 push XH
           push XL
45
           LDI XH, HIGH(5464)
47                 LDI XL, LOW (5464)
   COUNT1:
49                 SBIW XL, 1
                   BRNE COUNT1
51                 ;
           pop XL
53         pop XH
           RET
55 ;
   DEL100mus:
57                 push XH
           push XL
59
           LDI XH, HIGH(131)
61                 LDI XL, LOW (131)
   COUNT2:
63                 SBIW XL, 1
                   BRNE COUNT2
65                 ;
           pop XL
67         pop XH
           RET
69 ;
   DEL49ms:
71                 push XH
           push XL
73         ;
           LDI XH, HIGH(65535)
75                 LDI XL, LOW (65535)
   COUNT3:
77                 SBIW XL, 1
                   BRNE COUNT3
79
           pop XL
81         pop XH
           RET
83         ;
   ;
       ********************************************************************************************
```

```
;
    ************************************************************************************************
;
    ************************************************************************************************
;**************************** Initial pieces onto screen
    ***********************************
;
    ************************************************************************************************
;
    ************************************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;
    ************************************************************************************************
; Once the game has been initialised (chess_board_initialisation.asm),
    this routine loads the
; pieces onto the screen. The pieces can either be for a new game, or a
    loaded game.
;
    ************************************************************************************************
initial_pieces_onto_screen:
push XL
push XH
push r16
push r17
push r18
push r25

ldi XH,$01          ;load location of A1
ldi XL,$11          ;
ldi r16,$08         ;load a loop counter (required because we only go up
    to 8 on hex unit)

initial_pieces_loop:    ;
mov r17,XL          ;r17 is the piece position in output_DOGXL routine
ld r18,X+         ;r18 is the piece type, and this is stored at address X
     in SRAM
;
push r16          ;need to push all registers again before calling
    output_DOGXL routine
push XL           ;as the output_DOGXL routine changes their value
push XH
call output_DOGXL     ;load piece onto screen
pop XH
pop XL
pop r16
;
cpi r16,$01         ;check counter
breq counter_reset      ;if we have looped 8 times, branch to
```

```
    counter_reset
subi r16,$01        ;if not, subtract 1 from the counter
rjmp initial_pieces_loop  ;and go again!

counter_reset:        ;routine resets the counter to 8
ldi r16,$08
adiw X,$08          ;it also changes the square on the board to the
    bottom of the next column
cpi XL,$91          ;there are only 8 columns, so if we have $91, it
    means we have filled all squares
breq initial_pieces_exit
rjmp initial_pieces_loop

initial_pieces_exit:
pop r25
pop r18
pop r17
pop r16
pop XH
pop XL
ret

;
    ********************************************************************************
```

```
;*****************************************************************************
;********************** KING CHECK ******************************************
;*****************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;*****************************************************************************
; This code checks whether both kings are still left on the board (it is easier to check
; this rather than idenfitying check and check-mate).
;*****************************************************************************
king_check:
push XL
push XH
push r16
;
ldi XH,$01              ;start in the bottom left corner
ldi XL,$11
;
white_king_check_loop:      ;first loop looks for a white king
ld r16,X+           ;load the piece information of a square, then post-increment
cpi r16,$86             ;is the piece the white king?
breq black_king_check       ;if yes, then move on to finding the black king
cpi XL,$89              ;if no, then repeat this for the whole board
brne white_king_check_loop      ;if we get to $0189 and there is no white king, it means it has been taken
call initialise_LCD_DOGXL    ;clear the chess board as blacks have won
call MessBlackWinOut        ;The white king has been taken, so blacks win!
call BigBigDEL          ;Leave the message on the screen for a few seconds
call BigBigDEL
call BigBigDEL
pop r16
rjmp start_new_game         ;Jump to ask whether they want to start a new game
;
black_king_check:       ;White king was found. Now search for black king
ldi XH,$01              ;Go back to A1 square
ldi XL,$11
black_king_check_loop:      ;Loop across all squares on the chess board again
ld r16,X+           ;This is done by post-incrementing the location
cpi r16,$C6             ;Check if a black king is on the square
breq both_kings_alive       ;If the black king is found, both kings are alive!
cpi XL,$89              ;Continue checking across the whole board
```

```
brne black_king_check_loop
call initialise_LCD_DOGXL    ;clear the chess board as whites have won
call MessWhiteWinOut        ;If the whole board is check, and no black king, whites win!!!
call BigBigDEL              ;Keep message on the screen for a few seconds
call BigBigDEL
call BigBigDEL
pop r16
rjmp start_new_game         ;Jump to ask whether they want to start a new game
;
both_kings_alive:           ;Both kings are alive, so return to main program to resume game
pop r16
pop XH
pop XL
ret
;
start_new_game:             ;A king was taken, so we need to start a new game
call CLRDIS
call MessStartNewGameOut     ; "Do you want to start a new game? (Y/N)"
push r16
start_new_game_loop:        ;Loop until the user inputs [Y] or [N]
in r16,pind
com r16
cpi r16,$08              ;If they press [Y]=$08 on PIND
breq start_new_game_yes      ;branch to new game routine
cpi r16,$10              ;If they press no,jump to 'no new game' routine
brne start_new_game_loop
pop r16
rjmp start_new_game_no
;
start_new_game_yes:         ;New game just jumps to initialisation of main program
jmp Init
;
start_new_game_no:          ;No new game outputs message then waits in infinite loop
call MessTurnOffOut         ;"Turn game off."
TurnOffLoop:             ;Infinite loop!
rjmp TurnOffLoop
;
;
;*****************************************************************************
```

```
;*****************************************************************************
;*****************************************************************************
;********************************* MAPPING
;*****************************************************************************
;*****************************************************************************
;
;*****************************************************************************
;Author: Richard Flint
;Date: 1/3/2015
;
;
;*****************************************************************************
;*********************    Keyboard to ASCII mapping
;*****************************
;
;*****************************************************************************
;
; This works as a systematic look-up table that takes the 16 bit
; binary number saved from the keyboard input, and converts it into the
    associated
; ASCII values. This is output on the alphanumeric display.
;
;
;*****************************************************************************
keyboard_to_ASCII_letter:
    cpi r20,$EE
    breq letterA
    cpi r20,$ED
    breq letterB
    cpi r20,$EB
    breq letterC
    cpi r20,$E7
    breq letterD
    cpi r20,$DE
    breq letterE
    cpi r20,$DD
    breq letterF
    cpi r20,$DB
    breq letterG
    cpi r20,$D7
    breq letterH
    ;
    letterA:
    ldi r20,$41
    ret
```

```
39      letterB:
        ldi r20,$42
41      ret
        letterC:
43      ldi r20,$43
        ret
45      letterD:
        ldi r20,$44
47      ret
        letterE:
49      ldi r20,$45
        ret
51      letterF:
        ldi r20,$46
53      ret
        letterG:
55      ldi r20,$47
        ret
57      letterH:
        ldi r20,$48
59      ret
        ;
61  keyboard_to_ASCII_number:
        cpi r20,$BE
63      breq number1
        cpi r20,$BD
65      breq number2
        cpi r20,$BB
67      breq number3
        cpi r20,$B7
69      breq number4
        cpi r20,$7E
71      breq number5
        cpi r20,$7D
73      breq number6
        cpi r20,$7B
75      breq number7
        cpi r20,$77
77      breq number8
        ;
79      number1:
        ldi r20,$31
81      ret
        number2:
83      ldi r20,$32
        ret
85      number3:
        ldi r20,$33
87      ret
        number4:
89      ldi r20,$34
        ret
91      number5:
        ldi r20,$35
93      ret
```

```
    number6:
95      ldi r20,$36
        ret
97      number7:
        ldi r20,$37
99      ret
        number8:
101     ldi r20,$38
        ret
103 ;
    ;
105 ;
    ;
        ********************************************************************************

107 ;************************ ASCII to coordinate mapping
        ***********************
    ;
        ********************************************************************************

109 ;
    ;This works as a systematic look-up table that takes the ASCII
        coordinates,
111 ;and converts to the associated coordinate on the chess board. This
        coordinate
    ;system is defined in the report. For example, B3 corresponds to $0123.
113 ;
    ;
        ********************************************************************************

115 ascii_to_coordinate_mapping:
    cpi XH,$41
117 breq A_to_10
    cpi XH,$42
119 breq B_to_20
    cpi XH,$43
121 breq C_to_30
    cpi XH,$44
123 breq D_to_40
    cpi XH,$45
125 breq E_to_50
    cpi XH,$46
127 breq F_to_60
    cpi XH,$47
129 breq G_to_70
    cpi XH,$48
131 breq H_to_80
    ret
133 ;
    A_to_10:
135 ldi XH,$10
    rjmp number_mapping
137 B_to_20:
    ldi XH,$20
139 rjmp number_mapping
```

```
C_to_30:
ldi XH,$30
rjmp number_mapping
D_to_40:
ldi XH,$40
rjmp number_mapping
E_to_50:
ldi XH,$50
rjmp number_mapping
F_to_60:
ldi XH,$60
rjmp number_mapping
G_to_70:
ldi XH,$70
rjmp number_mapping
H_to_80:
ldi XH,$80
rjmp number_mapping
;
number_mapping:
cpi XL,$31
breq one_to_01
cpi XL,$32
breq two_to_02
cpi XL,$33
breq three_to_03
cpi XL,$34
breq four_to_04
cpi XL,$35
breq five_to_05
cpi XL,$36
breq six_to_06
cpi XL,$37
breq seven_to_07
cpi XL,$38
breq eight_to_08
ret
;
one_to_01:
ldi XL,$01
rjmp combine_coordinates
two_to_02:
ldi XL,$02
rjmp combine_coordinates
three_to_03:
ldi XL,$03
rjmp combine_coordinates
four_to_04:
ldi XL,$04
rjmp combine_coordinates
five_to_05:
ldi XL,$05
rjmp combine_coordinates
six_to_06:
ldi XL,$06
```

```
195 rjmp combine_coordinates
    seven_to_07:
197 ldi XL,$07
    rjmp combine_coordinates
199 eight_to_08:
    ldi XL,$08
201 rjmp combine_coordinates
    ;
203 combine_coordinates:
    add XL,XH
205 ldi XH,$00
    ret
207 ;
    ;
        ********************************************************************************
```

```
;*****************************************************************************
;************************ Messages ******************************************
;*****************************************************************************
;
; These messages are used in the the main function and other included routines
;
; ********************** Define messages in program memory ********************************
Mess1:
  .db "Input letter: ",0
Mess2:
  .db "Press enter/delete",0
Mess3:
  .db "Input number: ",0
Mess4:
  .db "Your coordinate: ",0
Mess6:
  .db "Confirm? (Y/N)",0
Mess7:
  .db "Your move has been  made! Next turn...",0
Mess8:
  .db "Are you sure you    want to exit? (Y/N)",0
Mess1spaces:
  .db " ",0
Mess5spaces:
  .db "     ",0
MessError:
  .db "Error!",0
MessGameSaved:
  .db "Exit complete!",0
MessNoPieces:
  .db "No piece selected.  Press enter to retry",0
MessLoad:
  .db "Do you want to load previous game? (Y/N)",0
MessNewGameStart:
  .db "New game.Press enter to continue",0
MessSelectWhitePiece:
  .db "WHITE: Select your  piece...",0
MessSelectBlackPiece:
  .db "BLACK: Select your  piece...",0
MessSelectDestination:
  .db "Select move...",0
MessInvalidMove:
  .db "Move not valid.Press enter to try again",0
MessValidMove:
  .db "Move is valid. Press enter to continue",0
MessWrongColour:
  .db "You selected  wrong colour. Try again.",0
MessTurnOff:
```

```
    .db "Turn game off.",0
MessStartNewGame:
    .db "Do you want to start a new game? (Y/N)",0
MessWhiteWin:
    .db "WHITE WIN!!!!!!",0
MessBlackWin:
    .db "BLACK WIN!!!!!!",0
MessProgess:
    .db "You have got this far!",0
MessButtonError:
    .db "ERROR! Please hold  button longer!",0
MessPressAgain:
    .db "Press again:  ",0
MessPressESC:
    .db "Press ESC to exit   demo",0
MessDemoQuestion:
    .db "Do you want to play    chess? (Y/N)",0
; ****************** Message output routines
    *********************************
Mess1out:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*Mess1)
    LDI ZL, LOW(2*Mess1)
    rjmp MessLoop
;
Mess2out:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*Mess2)
    LDI ZL, LOW(2*Mess2)
    rjmp MessLoop
;
Mess3out:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*Mess3)
    LDI ZL, LOW(2*Mess3)
    rjmp MessLoop
;
Mess4out:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*Mess4)
    LDI ZL, LOW(2*Mess4)
    rjmp MessLoop
    ;
```

```
Mess6out:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*Mess6)
  LDI ZL, LOW(2*Mess6)
  rjmp MessLoop
  ;
Mess7out:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*Mess7)
  LDI ZL, LOW(2*Mess7)
  rjmp MessLoop
;
Mess8out:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*Mess8)
  LDI ZL, LOW(2*Mess8)
  rjmp MessLoop
;
MessNoPiecesOut:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*MessNoPieces)
  LDI ZL, LOW(2*MessNoPieces)
  rjmp MessLoop
;
Mess1spacesOut:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*Mess1spaces)
  LDI ZL, LOW(2*Mess1spaces)
  rjmp MessLoop
  ;
Mess5spacesOut:
  push ZH
  push ZL
  push r17
  ;
  LDI ZH, HIGH(2*Mess5spaces)
  LDI ZL, LOW(2*Mess5spaces)
  rjmp MessLoop
  ;
MessGameSavedOut:
```

```
158     push ZH
        push ZL
160     push r17
        ;
162     LDI ZH, HIGH(2*MessGameSaved)
        LDI ZL, LOW(2*MessGameSaved)
164     rjmp MessLoop
        ;
166 MessLoadOut:
        push ZH
168     push ZL
        push r17
170     ;
        LDI ZH, HIGH(2*MessLoad)
172     LDI ZL, LOW(2*MessLoad)
        rjmp MessLoop
174     ;
    MessNewGameStartOut:
176     push ZH
        push ZL
178     push r17
        ;
180     LDI ZH, HIGH(2*MessNewGameStart)
        LDI ZL, LOW(2*MessNewGameStart)
182     rjmp MessLoop
        ;
184 MessSelectWhitePieceOut:
        push ZH
186     push ZL
        push r17
188     ;
        LDI ZH, HIGH(2*MessSelectWhitePiece)
190     LDI ZL, LOW(2*MessSelectWhitePiece)
        rjmp MessLoop
192     ;
    MessSelectBlackPieceOut:
194     push ZH
        push ZL
196     push r17
        ;
198     LDI ZH, HIGH(2*MessSelectBlackPiece)
        LDI ZL, LOW(2*MessSelectBlackPiece)
200     rjmp MessLoop
        ;
202 MessSelectDestinationOut:
        push ZH
204     push ZL
        push r17
206     ;
        LDI ZH, HIGH(2*MessSelectDestination)
208     LDI ZL, LOW(2*MessSelectDestination)
        rjmp MessLoop
210     ;
    MessInvalidMoveOut:
212     push ZH
```

```
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessInvalidMove)
    LDI ZL, LOW(2*MessInvalidMove)
    rjmp MessLoop
    ;
MessValidMoveOut:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessValidMove)
    LDI ZL, LOW(2*MessValidMove)
    rjmp MessLoop
;
MessWrongColourOut:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessWrongColour)
    LDI ZL, LOW(2*MessWrongColour)
    rjmp MessLoop
;
MessTurnOffOut:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessTurnOff)
    LDI ZL, LOW(2*MessTurnOff)
    rjmp MessLoop
;
MessStartNewGameOut:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessStartNewGame)
    LDI ZL, LOW(2*MessStartNewGame)
    rjmp MessLoop
;
MessWhiteWinOut:
    push ZH
    push ZL
    push r17
    ;
    LDI ZH, HIGH(2*MessWhiteWin)
    LDI ZL, LOW(2*MessWhiteWin)
    rjmp MessLoop
;
MessBlackWinOut:
    push ZH
    push ZL
```

```
268   push r17
      ;
270   LDI ZH, HIGH(2*MessBlackWin)
      LDI ZL, LOW(2*MessBlackWin)
272   rjmp MessLoop
;
274 MessProgessOut:
      push ZH
276   push ZL
      push r17
278   ;
      LDI ZH, HIGH(2*MessProgess)
280   LDI ZL, LOW(2*MessProgess)
      rjmp MessLoop
282 ;
    MessButtonErrorOut:
284   push ZH
      push ZL
286   push r17
      ;
288   LDI ZH, HIGH(2*MessButtonError)
      LDI ZL, LOW(2*MessButtonError)
290   rjmp MessLoop
      ;
292 ;
    MessPressAgainOut:
294   push ZH
      push ZL
296   push r17
      ;
298   LDI ZH, HIGH(2*MessPressAgain)
      LDI ZL, LOW(2*MessPressAgain)
300   rjmp MessLoop
;
302 MessPressESCOut:
      push ZH
304   push ZL
      push r17
306   ;
      LDI ZH, HIGH(2*MessPressESC)
308   LDI ZL, LOW(2*MessPressESC)
      rjmp MessLoop
310 ;
    MessDemoQuestionOut:
312   push ZH
      push ZL
314   push r17
      ;
316   LDI ZH, HIGH(2*MessDemoQuestion)
      LDI ZL, LOW(2*MessDemoQuestion)
318   rjmp MessLoop
;; ********************** Message output loop
     ************************************
320 ;
    MessLoop:       ; Loops until message has been outputted
```

```
322     LPM r17 , Z+
        cpi r17, $00
324     breq MessEnd
        sts $C000, r17
326     push r16
        call busylcd
328     pop r16
        RJMP MessLoop
330 MessEnd:
        pop r17
332     pop ZL
        pop ZH
334     ret
    ;
336 ;
        ********************************************************************************
```

```
;
    ********************************************************************************
;
    ********************************************************************************
; ************************* MOVE IDENTIFIER ALGORITHM
    **********************
;
    ********************************************************************************
;
    ********************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;
; This code takes the input position (r16) and final position (r17), and
; identifies the type of move. The output of this code is then used in
    the
; Move Verification Algorithm.
;
; Routine will output the following:
;
; r19 = $01   forward vertical move
; r19 = $02   backwards vertical move
; r19 = $03   right horizontal move
; r19 = $04   left horizontal move
; r19 = $05   forward right diagonal move
; r19 = $06   backwards left diagonal move
; r19 = $07   forwards left diagonal move
; r19 = $08   backwards right diagonal move

; This code is written such that it can be included in the main program.
    It will
; not compile in isolation.
;
; Routine can be called in the following way:
;     * call move_identifier_algorithm will run the whole program
;
;********************* Registers for reference
    ******************************
; r16 = initial position     (INPUT)
; r17 = final position       (INPUT)
; r18 = piece information    (INPUT)
; r19 = move type           (OUTPUT)
;
    ********************************************************************************
;
;
move_identifier_algorithm:
push r18            ;use r18 as a comparison to see if move has been
    identified yet
ldi r18,$00
```

```
;
rcall horizontal_move_check   ;routine for checking whether input is horizontal move
cpse r19,r18          ;if it isn't horizontal, skip to the next check
rjmp move_type_output     ;if it is horizontal, jump to exit routine
rcall vertical_move_check   ;routine for checking whether input is vertical move
cpse r19,r18          ;if it isn't vertical, skip to next check
rjmp move_type_output     ;if it is vertical, jump to exit routine
rcall diagonal_move_check   ;routine for checking whether input is a diagonal move
cpse r19,r18
rjmp move_type_output     ;if isn't diagonal, no other options (knight not considered here), so jump to exit routine
;
move_type_output:
pop r18
ret
;
;
; ********************** Horizontal Move Check ***************************
; Checks whether the move is a horizontal move
;
horizontal_move_check:
push r16
push r17
push XL
push XH
;
mov XL,r16              ; store start and end positions
mov XH,r17
;
lsl XL                ; shift left to isolate lowest hex unit of start position
lsl XL
lsl XL
lsl XL
;
lsl XH                ; shift left to isolate lowest hex unit of end position
lsl XH
lsl XH
lsl XH
;
cp XL,XH              ;compare lowest hex numbers
breq horizontal_direction_check   ;lowest hex unit are equal if move is horizontal.
;
ldi r19,$00             ; if not equal, move is not horizontal. Identify this via r19=$00
rjmp horizontal_pop         ; jump to horizonal pop to pop back all registers that were pushed
;
  horizontal_direction_check:   ;if it is a horizontal move, we need to
```

```
    identify which direction
86  cp r17,r16            ;checks if horizontal move is left or right
    brlo horizontal_left      ;if end position is less than start position,
     move is left
88  ;               ;if end position is greater than start position, move
    right
    ldi r19,$03           ; move is horizontal right
90  rjmp horizontal_pop
;
92    horizontal_left:
      ldi r19,$04         ;move is horizontal left
94    rjmp horizontal_pop
;
96  ;
horizontal_pop:           ;pops back all variables pushed at the
    beginning of routine
98  pop XH
pop XL
100 pop r17
pop r16
102 ret
;
104 ;
; ********************** Vertical Move Check
    ***************************
106 ; Checks whether the move is vertical
;
108 vertical_move_check:
push r16
110 push r17
push XL
112 push XH
;
114 mov XL,r16              ;move start position into XL so not lost
mov XH,r17              ;move end posiition into XH so not lost
116 ;
lsr XL                ;isolate 2nd unit in HEX number for start position
118 lsr XL
lsr XL
120 lsr XL
;
122 lsr XH                ;isolate 2nd unit in HEX number for end position
lsr XH
124 lsr XH
lsr XH
126 ;
cp XH,XL              ;if move is vertical, then 2nd unit in HEX should
    be equal
128 breq vertical_direction_check   ;if it is equal, branch to routine that
    identifies direction
;
130 ldi r19,$00             ;if it is not equal, move is not vertical
rjmp vertical_pop         ;jump to pop everything back before exiting
132 ;
  vertical_direction_check:   ;if the move is identified as vertical, we
```

```
    still need to determine direction
134 cp r17,r16           ;if end position is greater than start position,
    vertical forwards move
    brlo vertical_backwards     ;if end position is less that start
    position, vertical backwards move
136 ;
    ldi r19,$01          ; move is vertical forwards
138 rjmp vertical_pop
    ;
140   vertical_backwards:
      ldi r19,$02        ; move is vertical backwards
142   rjmp vertical_pop
;
144 vertical_pop:
pop XH
146 pop XL
pop r17
148 pop r16
ret
150 ;
;
    ********************************************************************

152 ; ********************** Diagonal Move Check
    ***************************
;
    ********************************************************************

154 ; Checks if the move is diagonal, and identifies which direction
diagonal_move_check:
156 push r16
push r17
158 push r20
push r21
160 push r22
push r23
162 push r24
push r25
164 push XL
push XH
166 push YL
push YH
168 ;
mov XL,r16          ;save start position into XL so not lost
170 mov XH,r16          ;save start position into XL so not lost
mov YL,r17          ;save end position into XL so not lost
172 mov YH,r17          ;save end position into XL so not lost
;
174 lsl XL            ;isolate lowest hex unit of start position
lsl XL
176 lsl XL
lsl XL
178 lsr XL
lsr XL
180 lsr XL
```

```
    lsr XL
182 ;
    lsl YL              ;isolate lowest hex unit of end position
184 lsl YL
    lsl YL
186 lsl YL
    lsr YL
188 lsr YL
    lsr YL
190 lsr YL
    ;
192 lsr XH              ; isolate second lowest hex unit of start position
    lsr XH
194 lsr XH
    lsr XH
196 ;
    lsr YH              ; isolate second lowest hex unit of end position
198 lsr YH
    lsr YH
200 lsr YH
    ;
202 ; r16 = start position
    ; r17 = end position
204 ; r16 = XH + XL = upper hex value + lower hex value
    ; r17 = YH + YL = upper hex value + lower hex value
206 ;
    ; ************** Is it a forward diagonal right or backwards diagonal
        left? ****************************
208 ;
    diagonal_check_part_1:
210 ;
    mov r20,XL              ; we need to maintain X and Y values, so transfer
        to new registers
212 mov r21,XH
    mov r22,YL
214 mov r23,YH
    ;
216 cp r21,r20              ; compare HEX units for start position
    brlo inverse_subtract      ; identify which is biggest
218 sub r21,r20             ; if second hex unit is biggest, subtract first
        from second
    ldi r24,$01             ; if second hex unit is greater than first hex unit
        , set to $01 as marker
220 rjmp diagonal_check_part_2
    ;
222   inverse_subtract:
      sub r20,r21           ; if first hex unit is largest, subtract second
        unit from first hex unit
224   mov r21,r20           ; copy into r21 (simplifies code when doing the
        comparison)
      ldi r24,$00           ; Set r24=$00 as marker
226   rjmp diagonal_check_part_2
    ;
228 diagonal_check_part_2:
    cp r23,r22              ; repeat same proceedure as diagonal_check_part_1
```

```
    with HEX units from end position
brlo inverse_subtract_2
sub r23,r22
ldi r25,$01           ; if upper hex is greater than lower hex, set to
    $01
rjmp diagonal_compare
;
  inverse_subtract_2:
  sub r22,r23
  mov r23,r22
  ldi r25,$00         ;if lower hex is greater than upper hex, set to $00
  rjmp diagonal_compare
;
diagonal_compare:         ; r21 = difference between hex units for start
    position
                  ; r23 = difference between hex units for end position
cp r21,r23              ; is their difference the same?
brne diagonal_check_part_3      ; if no, then the move is not this sort
    of diagonal. Skip on!
cp r24,r25              ; if yes, then was the subtraction direction the
    same?
brne diagonal_check_part_3      ; if no, then the move is not this sort
    of diagonal. Skip on!
;
cp r17,r16              ; if yes, we just need to determine the direction
brlo diagonal_backwards_left_move
ldi r19,$05             ; If end position > start position, it is a
    forward right diagonal move
rjmp diagonal_pop
;
  diagonal_backwards_left_move: ; If end position < start position, it is
     a backwards left diagonal move
  ldi r19,$06
  rjmp diagonal_pop
;
; ************** Is it a forward diagonal left or backwards diagonal
    right? ****************************
;
diagonal_check_part_3:
;
mov r20,XL              ; restore registers
mov r21,XH
mov r22,YL
mov r23,YH
;
cp r22,r20              ; compare first HEX unit between start and end
    position
brlo inverse_subtract_3     ; establish which one is greater
sub r22,r20             ; subtract smaller HEX unit from larger HEX unit
ldi r24,$01             ; set marker
rjmp diagonal_check_part_4
;
  inverse_subtract_3:
  sub r20,r22           ; subtract smaller HEX unit from larger HEX unit
  mov r22,r20           ; transfer to r22 as it makes the comparison easier
```

```
      in a bit
    ldi r24,$00          ; set marker
    rjmp diagonal_check_part_4
;
diagonal_check_part_4:       ; do the same again with the 2nd HEX unit for
    the start and end position
cp r23,r21               ; establish which one is greater
brlo inverse_subtract_4
sub r23,r21              ; find the difference
ldi r25,$01              ; set as marker
rjmp diagonal_compare_2
;
  inverse_subtract_4:
  sub r21,r23            ; find the difference
  mov r23,r21            ; save in r23
  ldi r25,$00            ; set as marker
  rjmp diagonal_compare_2
;
diagonal_compare_2:        ; compare whether the order of subtraction was
    different
cp r24,r25
breq not_diagonal_move       ; if no, then not a diagonal move
cp r23,r22               ; if yes, compare magnitude of differences
    calculated
brne not_diagonal_move       ; if not equal, then not a diagonal move
;
cp r17,r16               ; compare start/end position to determine direction
     of movement
brlo diagonal_forward_left_move ; if end position is less than start
    position, branch to forward diagonal left
ldi r19,$08              ; if not, then it is a backwards diagonal right
rjmp diagonal_pop
;
  diagonal_forward_left_move:
  ldi r19,$07            ;save as forward diagonal left move
  rjmp diagonal_pop
;
not_diagonal_move:
ldi r19,$00
rjmp diagonal_pop
;
diagonal_pop:
pop YH
pop YL
pop XH
pop XL
pop r25
pop r24
pop r23
pop r22
pop r21
pop r20
pop r17
pop r16
ret
```

```
324 ;
    ;
        ********************************************************************************
```

```
;*****************************************************************************
;*****************************************************************************
;************************ Move Verification Algorithm ************************
;*****************************************************************************
;*****************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;*****************************************************************************
; This program checks whether the pieces present at the chosen start point can make a valid move
; to the destination point chosen.
;
;The program does not run in isolation. It is included in the main_program.asm,
;and is called as a routine (call move_verification_algorithm)
;********************* Input registers for reference *************************
; r16 = initial position
; r17 = final position
; r18 = piece information
; r19 = move type           $00 = no move
; r20 = intermediate step
; r21 = intermediate piece
; r22 = valid move?         $00 = not valid   $01 = valid
; r23 = is the space occupied?    $00 = unoccuped   $01 = white occupied
       $10 = black occupied
; r24 = colour of chosen piece (piece we are moving)    $01 = white
        $10 = black
; r25 = colour that the piece should be (i.e. white or black turn?) $01 =
     white   $10 = black
;
;*****************************************************************************
;
;****************** r18 piece information for reference **********************
; r18 = 0xxx xxxx   unoccupied
; r18 = 1xxx xxxx occupied
; r18 = x0xx xxxx white
; r18 = x1xx xxxx black
; r18 = xxxx 0001 pawn
; r18 = xxxx 0010
; r18 = xxxx 0011
; r18 = xxxx 0100
```

```
; r18 = xxxx 0101 queen
; r18 = xxxx 0110 king
;
    ********************************************************************************
;
;
;
    ********************************************************************************
;********************** Identify which piece is moving
    **********************
;
    ********************************************************************************
move_verification_algorithm:
;
  mov r20,r16 ;set intermediate move register as r20
  ;
  ;Identify which type of piece is being moved:
  cpi r18,$81
  breq white_pawn_jump
  cpi r18,$C1
  breq black_pawn_jump
  cpi r18,$82
  breq knight_jump
  cpi r18,$C2
  breq knight_jump
  cpi r18,$83
  breq bishop_jump
  cpi r18,$C3
  breq bishop_jump
  cpi r18,$84
  breq castle_jump
  cpi r18,$C4
  breq castle_jump
  cpi r18,$85
  breq queen_jump
  cpi r18,$C5
  breq queen_jump
  cpi r18,$86
  breq king_jump
  cpi r18,$C6
  breq king_jump
  ;
  ldi r22,$00     ;if it is none of these, then the move is not valid
  ret         ;this is marked by setting r22=$00
;
  white_pawn_jump:  ;relative branch is out of range, so jumps must be
    used*
  jmp white_pawn
  black_pawn_jump:
  jmp black_pawn
  knight_jump:
  jmp knight
```

```
83   bishop_jump:
     jmp bishop
85   castle_jump:
     jmp castle
87   queen_jump:
     jmp queen
89   king_jump:
     jmp king
91 ;
   ;*Note: because the program is quite long, the breq command (and other
       relative commands) sometimes
93 ;are insufficient by themselves because the routine is too far away. If
       this is the case,
   ;then the relative command redirects to a jmp statement, which then takes
        program to the
95 ;correct routine
   ;
97 ;
       ********************************************************************************

   ;************** MOVE VERIFICATION ALGORITH FOR EACH PIECE
       ***********************
99 ;
       ********************************************************************************

   ;
101 ;
   ;
       ********************************************************************************

103 ;************************       Knight
       *********************************
   ;
       ********************************************************************************

105 ;
   knight:          ;move verification for a knight
107 push r20
   ;
109 cp r17,r16             ;compare start and end positions
   brlo knight_left         ;if end is smaller than start, knight is moving
       right overall
111 ;knight moving right       ;if end is greater than start, knight is moving
        left overall
   mov r20,r17             ;temporarily set r20 as destination square to
       preserve r17
113 sub r20,r16             ;find the difference between destination and start
       square
   cpi r20,$12 ;forward right 1  ;the difference can only have a few set
       values for the knight move
115 breq knight_moving         ;if it is one of these, then branch to move the
        knight
   cpi r20,$21 ;forward right 2      ;
117 breq knight_moving
   cpi r20,$1F ;back right 1
```

```
119 breq knight_moving
    cpi r20,$E  ;back right 2
121 breq knight_moving
    ;
123 ldi r22,$00           ;if none of these are correct, then move is not
        valid
    pop r20
125 ret               ;this is marked by setting r22=$00
    ;
127 ;knight moving left
    knight_left:          ;knight can only move left into a few different
        squares. Check each of these.
129 mov r20,r16           ;temporarily set r20 as destination square to
        preserve r17
    sub r20,r17           ;the difference can only have a few set values for
        the knight move
131 cpi r20,$E  ;forward left 1   ;if it is one of these, then branch to move
         the knight
    breq knight_moving
133 cpi r20,$1F ;forward left 2
    breq knight_moving
135 cpi r20,$21 ;back left 1
    breq knight_moving
137 cpi r20,$12 ;back left 2
    breq knight_moving
139 ;
    ldi r22,$00           ;if none of these are correct, then move is not
        valid
141 pop r20
    ret               ;this is marked by setting r22=$00
143 ;
    knight_moving:      ;if destination is valid, still need to check that it
         is not occupied by same colour piece
145 pop r20
    mov r20,r17       ;set r20 (intermediate step) to be the destination
147 call is_the_square_occupied   ;check if there is a piece in the
        destination square
    cp r23,r24        ;r23 gives colour of piece in destination square. r24
        gives colour of piece moving
149 brne knight_move_valid  ;destination square must either be free of have
        different colour piece
    jmp moving_error    ;if it doesn't then the move is not valid
151 knight_move_valid:
    ldi r22,$01       ;if it does, then the move is valid. This is saved by
        setting r22=$01
153 ret
    ;
155 ;
    ;
157 ;
    ;
        ********************************************************************************

159 ;*************************       King
        ************************************
```

```
;
    ********************************************************************************

;
king:               ;move verification for a king
call move_identifier_algorithm    ;identify type of move
cpi r19,$01             ;branch/jump to relevant king check
breq king_forward
cpi r19,$02
breq king_backwards
cpi r19,$03
breq king_right
cpi r19,$04
breq king_left
cpi r19,$05
breq king_forward_diagonal_right_jump
cpi r19,$06
breq king_backwards_diagonal_left_jump
cpi r19,$07
breq king_forward_diagonal_left_jump
cpi r19,$08
breq king_backwards_diagonal_right_jump
ldi r22,$00             ;if none of these are possible, then the move is
    not valid!
ret
;
king_forward_diagonal_right_jump: ;jumps needed because relative branch
    too far
jmp king_forward_diagonal_right
king_backwards_diagonal_left_jump:
jmp king_backwards_diagonal_left
king_forward_diagonal_left_jump:
jmp king_forward_diagonal_left
king_backwards_diagonal_right_jump:
jmp king_backwards_diagonal_right
;
king_forward:           ;check that destination is just 1 square from
    start point
push r17
sub r17,r16
cpi r17,$01
breq king_forward_yes
pop r17
jmp moving_error
king_forward_yes:
pop r17
jmp moving_forward
;
king_backwards:           ;check that destination is just 1 square from
    start point
push r16
sub r16,r17
cpi r16,$01
breq king_backwards_yes
pop r16
```

```
209 jmp moving_error
    king_backwards_yes:
211 pop r16
    jmp moving_backwards
213 ;
    king_right:             ;check that destination is just 1 square from
        start point
215 push r17
    sub r17,r16
217 cpi r17,$10
    breq king_right_yes
219 pop r17
    jmp moving_error
221 king_right_yes:
    pop r17
223 jmp moving_right
    ;
225 king_left:              ;check that destination is just 1 square from
        start point
    push r16
227 sub r16,r17
    cpi r16,$10
229 breq king_left_yes
    pop r16
231 jmp moving_error
    king_left_yes:
233 pop r16
    jmp moving_left
235 ;
    king_forward_diagonal_right:    ;check that destination is just 1 square
        from start point
237 push r17
    sub r17,r16
239 cpi r17,$11
    breq king_forward_diagonal_right_yes
241 pop r17
    jmp moving_error
243 king_forward_diagonal_right_yes:
    pop r17
245 jmp moving_forward_diagonal_right
    ;
247 king_backwards_diagonal_left:   ;check that destination is just 1 square
        from start point
    push r16
249 sub r16,r17
    cpi r16,$11
251 breq king_backwards_diagonal_left_yes
    pop r16
253 jmp moving_error
    king_backwards_diagonal_left_yes:
255 pop r17
    jmp moving_backwards_diagonal_left
257 ;
    king_forward_diagonal_left:     ;check that destination is just 1 square
        from start point
```

```
259 push r16
    sub r16,r17
261 cpi r16,$09
    breq king_forward_diagonal_left_yes
263 pop r16
    jmp moving_error
265 king_forward_diagonal_left_yes:
    pop r17
267 jmp moving_forward_diagonal_left
    ;
269 king_backwards_diagonal_right:    ;check that destination is just 1
        square from start point
    push r17
271 sub r17,r16
    cpi r17,$09
273 breq king_backwards_diagonal_right_yes
    pop r17
275 jmp moving_error
    king_backwards_diagonal_right_yes:
277 pop r17
    jmp moving_backwards_diagonal_right
279 ;
    ;
        ********************************************************************************

281 ;************************       Queen
        ***********************************
    ;
        ********************************************************************************

283 ;       ;move verification for a queen
    queen:      ;queen is just a combination of castle and bishop
285 call castle   ;try castle moves
    cpi r22,$00
287 brne queen_2  ;if castle move produces valid move, then branch to exit
    call bishop   ;if castle move does not produce valid move, then try a
        bishop move
289 queen_2:
    RET
291 ;
    ;
        ********************************************************************************

293 ;************************       Castle
        ***********************************
    ;
        ********************************************************************************

295 ;
    castle:              ;move verification for a castle
297 call move_identifier_algorithm    ;identify type of move
    cpi r19,$01             ;branch to type of move check
299 breq castle_moving_forward
    cpi r19,$02
301 breq castle_moving_backwards
```

```
    cpi r19,$03
303 breq castle_moving_right
    cpi r19,$04
305 breq castle_moving_left
    ldi r22, $00            ;if none of these are applicable, then move is
        not valid
307 RET
    ;
309 castle_moving_forward:        ;need to use jmp command because relative
        branch too far
    jmp moving_forward
311 castle_moving_backwards:
    jmp moving_backwards
313 castle_moving_right:
    jmp moving_right
315 castle_moving_left:
    jmp moving_left
317 ;
    ;
        ********************************************************************************

319 ;************************      Bishop
        **********************************
    ;
        ********************************************************************************

321 ;
    bishop:                   ;move verification for a bishop
323 call move_identifier_algorithm        ;identify type of move
    cpi r19,$05                 ;branch/jmp to appropriate move type
325 breq bishop_moving_forward_diagonal_right
    cpi r19,$06
327 breq bishop_moving_backwards_diagonal_left
    cpi r19,$07
329 breq bishop_moving_forward_diagonal_left
    cpi r19,$08
331 breq bishop_moving_backwards_diagonal_right
    ldi r22, $00                ;if none of these are applicable, then move
        is not valid
333 RET
    ;
335 bishop_moving_forward_diagonal_right:    ;need to use jmp command because
        relative branch too far
    jmp moving_forward_diagonal_right
337 bishop_moving_backwards_diagonal_left:
    jmp moving_backwards_diagonal_left
339 bishop_moving_forward_diagonal_left:
    jmp moving_forward_diagonal_left
341 bishop_moving_backwards_diagonal_right:
    jmp moving_backwards_diagonal_right
343 ;
    ;
345 ;
        ********************************************************************************
```

```
;************************      PAWNS
    ************************************
;
    ********************************************************************************

;
;
    ********************************************************************************

;               White pawn
;
    ********************************************************************************

white_pawn:
call move_identifier_algorithm        ;identify type of move
cpi r19,$02                 ;is the pawn stepping backwards 1 square?
breq white_pawn_backwards          ;if yes, branch to correct routine
cpi r19,$08                 ;is the pawn stepping backwards diagonal
    right?
breq white_pawn_diagonal_backwards_right  ;if yes, then branch to correct
     routine
cpi r19,$06                 ;is the pawn stepping backwards diagonal
    left?
breq white_pawn_diagonal_backwards_left   ;if yes, then branch to correct
     routine
ldi r22,$00                 ;if it is none of these, then the move is ***
    NOT VALID***
RET
;
;
;********************** White pawn diagonal backwards right
    *********************************
;
white_pawn_diagonal_backwards_right:
push r16
ldi r16,$F
add r20,r16
pop r16             ;move forward 1 square backwards diagonal right
cp r20,r17            ;is this the final destination?
brne moving_error_jump_01   ;if no, then the move is not valid. If yes,
    then carry on
call is_the_square_occupied   ;call routine to check if the destination
    square is full
cpi r23,$10           ;check if square is occupied by a black piece
brne moving_error_jump_01   ;if not, then the move is not valid
ldi r22,$01           ;if yes, then the move is valid
RET
;
moving_error_jump_01:     ;need to jump because moving_error routine is
    too far for relative branch
jmp moving_error
;********************** White pawn diagonal backwards left
    *********************************
;
white_pawn_diagonal_backwards_left:
```

```
    subi r20,$11          ;move forward 1 square backwards diagonal left
385 cp r20,r17            ;is this the final destination?
    brne moving_error_jump_02   ;if no, then the move is not valid. If yes,
        then carry on
387 call is_the_square_occupied   ;call routine to check if the destination
        square is full
    cpi r23,$10           ;check if square is occupied by a white piece
389 brne moving_error_jump_02   ;if not, then the move is not valid
    ldi r22,$01           ;if yes, then the move is valid
391 RET
    ;
393 moving_error_jump_02:     ;need to jump because moving_error routine is
        too far for relative branch
    jmp moving_error
395 ;
    ;********************** White pawn backwards
        ***********************************************
397 white_pawn_backwards:
    subi r20,$01            ;move backwards 1
399 call is_the_square_occupied
    cpi r23,$00
401 breq white_pawn_backwards_2     ;if r23 = $00 (space is free), then can
        carry on on
    jmp moving_error          ;if space is not free, then move is not valid
403 ;
    white_pawn_backwards_2:
405 cp r20,r17              ;is it the end position?
    brne white_pawn_double_backwards  ;if it is not the end position, check
        double pawn move
407 ldi r22,$01             ;if it is the end position, then the move is
        valid
    RET                 ;return with valid move
409 ;

411 white_pawn_double_backwards:
    subi r20,$01            ;move pawn backwards another 1 square
413 cp r20,r17
    brne white_pawn_error_jump      ;if this is not the final location, then
        error
415 call is_the_square_occupied     ;call routine to check if the destination
         square is full
    cpi r23,$00             ;check if square is free
417 breq white_pawn_double_backwards_2  ;if square is free, then carry on
    jmp moving_error          ;if square is not free, then the move is not
        vlaid
419 ;
    white_pawn_double_backwards_2:
421 push r16
    ;
423 lsl r16               ;isolate the lowest hex unit in the start position
        of the pawn
    lsl r16
425 lsl r16
    lsl r16
427 ;
```

```
    cpi r16,$70             ;check if pawn's initial position was a start
        game position
429 breq white_pawn_double_backwards_3  ;if valid, branch to confirm valid
        move
    pop r16               ;if r16 is not the pawns position at the start of
        the game, then a double move is not valid
431 jmp moving_error
    ;
433 white_pawn_double_backwards_3:
    ldi r22,$01             ;signify that a double move is by setting r22=$01
435 pop r16
    RET
437 ;
    white_pawn_error_jump:        ;need to jump because moving_error routine
        is too far for relative branch
439 jmp moving_error
    ;
        ********************************************************************************

441 ;                 Black pawn
    ;
        ********************************************************************************

443 black_pawn:
    call move_identifier_algorithm      ;identify type of move
445 cpi r19,$01               ;is the pawn stepping forward 1 square?
    breq black_pawn_forward         ;if yes, branch to correct routine
447 cpi r19,$05               ;is the pawn stepping forward diagonal right?
    breq black_pawn_diagonal_forward_right  ;if yes, then branch to correct
        routine
449 cpi r19,$07               ;is the pawn stepping forward diagonal left?
    breq black_pawn_diagonal_forward_left   ;if yes, then branch to correct
        routine
451 ldi r22,$00               ;if it is none of these, then the move is not
        valid
    RET
453 ;
    ;
455 ;********************** Black pawn diagonal forward right
        **********************************
    ;
457 black_pawn_diagonal_forward_right:
    push r16
459 ldi r16,$11
    add r20,r16
461 pop r16             ;move forward 1 square forward diagonal right
    cp r20,r17            ;is this the final destination?
463 brne moving_error_jump_03   ;if no, then the move is not valid. If yes,
        then carry on
    call is_the_square_occupied   ;call routine to check if the destination
        square is full
465 cpi r23,$01           ;check if square is occupied by a white piece
    brne moving_error_jump_03   ;if not, then the move is not valid
467 ldi r22,$01           ;if yes, then the move is valid
    RET
```

```
469 ;
    moving_error_jump_03:      ;need to jump because moving_error routine is
        too far for relative branch
471 jmp moving_error
    ;
473 ;********************** Black pawn diagonal forward left
        *********************************
    ;
475 black_pawn_diagonal_forward_left:
    subi r20,$F          ;move forward 1 square forward diagonal right
477 cp r20,r17             ;is this the final destination?
    brne moving_error_jump_04    ;if no, then the move is not valid. If yes,
        then carry on
479 call is_the_square_occupied    ;call routine to check if the destination
        square is full
    cpi r23,$01            ;check if square is occupied by a white piece
481 brne moving_error_jump_04    ;if not, then the move is not valid
    ldi r22,$01            ;if yes, then the move is valid
483 RET
    ;
485 moving_error_jump_04:      ;need to jump because moving_error routine is
        too far for relative branch
    jmp moving_error
487 ;
    ;********************** Black pawn forward
        ***********************************************
489 black_pawn_forward:
    push r16
491 ldi r16,$01
    add r20,r16
493 pop r16              ;move forward 1
    call is_the_square_occupied    ;call routine to check if the destination
        square is full
495 cpi r23,$00
    breq black_pawn_forward_2    ;if r23 = $00 (space is free), then can carry
         on on
497 jmp moving_error         ;if space is not free, then move is not valid
    ;
499 black_pawn_forward_2:
    cp r20,r17             ;is it the end position?
501 brne black_pawn_double_forward  ;if it is not the end position, check
        double pawn move
    ldi r22,$01            ;signal that the move is valid by setting r22=$01
503 RET
    ;
505
    black_pawn_double_forward:
507 push r16
    ldi r16,$01
509 add r20,r16
    pop r16                  ;move forward another 1
511 cp r20,r17
    brne black_pawn_error_jump         ;if this is not the final destination,
        then error
513 call is_the_square_occupied        ;call routine to check if the
```

```
    destination square is full
cpi r23,$00               ;check if square is free
breq black_pawn_double_forward_2    ;if square is free, branch on
jmp moving_error            ;if r23 is not free, move not valid
;
black_pawn_double_forward_2:
push r16
;
lsl r16               ;isolate the lowest hex unit in the start position
    of the pawn
lsl r16
lsl r16
lsl r16
;
cpi r16,$20             ;check if pawn's initial position was a start
    game position
breq black_pawn_double_forward_3  ;if valid, branch to confirm valid move
pop r16               ;if r16 is not the pawns initial position, then a
    double move is not valid
jmp moving_error
;
black_pawn_double_forward_3:
ldi r22,$01             ;signify that a double move is valid by setting
    r22=$01
pop r16
RET
;
black_pawn_error_jump:        ;need to jump because moving_error routine
    is too far for relative branch
jmp moving_error
;
;
;
    ********************************************************************************

;************************ GENERAL MOVEMENT ROUTINES
    ***************************
;
    ********************************************************************************

;These routines are used by the castle, bishop and queen, all of whom can
     move
;large distances across the board. We not only need to check that the
    final destination
;is valid, but also that there are no pieces in between the start and end
     positions.
;
;
    ********************************************************************************

;************************** Moving forward
    ***********************************
;
    ********************************************************************************
```

```
moving_forward:
push r16
ldi r16,$01          ;move forwards one
add r20,r16
pop r16
call is_the_square_occupied ;call routine to check if the square is full
cpi r23,$00
breq moving_forward_2      ;if space is free, branch with no problems
cp r20,r17           ;the space is not free. is this the final position?
breq moving_forward_3    ;if this is the final position, then more tests
    are required, so branch on
jmp moving_error         ;if this is not the final position, then the
    movement is not possible regardless of colour
;
moving_forward_2:      ;space is free
cp r20,r17           ;is it the final position?
brne moving_forward      ;if no, then repeat
ldi r22,$01          ;if this is the final position, then the move is
    valid
ret
;
moving_forward_3:      ;space is not free but is also final destination
cp r23,r24           ;compare colour of pieces
breq moving_error_jump1    ;if the piece you are moving is the same colour
     as the piece in the final square, the move is not valid
ldi r22,$01          ;if the piece you are moving is a different colour in
     the final square, then the move is valid
RET
;
moving_error_jump1:      ;need to jump because relative branch out of
    range
jmp moving_error
;
;
;
    ********************************************************************************

;************************** Moving backwards
    **********************************
;
    ********************************************************************************

;
moving_backwards:
subi r20,$01         ;move backwards one
call is_the_square_occupied ;call routine to check if the square is full
cpi r23,$00
breq moving_backwards_2    ;if space is free, branch with no problems
cp r20,r17           ;the space is not free. is this the final position?
breq moving_backwards_3    ;if this is the final position, then more tests
     are required, so branch on
jmp moving_error         ;if this is not the final position, then the
    movement is not possible regardless of colour
;
moving_backwards_2:      ;space is free
```

```
    cp r20,r17          ;is it the final position?
593 brne moving_backwards     ;if no, then repeat loop
    ldi r22,$01         ;if this is the final position, then the move is
        valid
595 ret
    ;
597 moving_backwards_3:     ;space is not free but is also final destination
    cp r23,r24          ;compare colour of pieces
599 breq moving_error_jump2   ;if the piece you are moving is the same colour
         as the piece in the final square, the move is not valid
    ldi r22,$01         ;if the piece you are moving is a different colour in
         the final square, then the move is valid
601 RET
    ;
603 moving_error_jump2:     ;need to jump because relative branch out of
        range
    jmp moving_error
605 ;
    ;
        ********************************************************************************

607 ;************************** Moving right
        *********************************
    ;
        ********************************************************************************

609 ;
    moving_right:
611 push r16
    ldi r16,$10
613 add r20,r16           ;move right one
    pop r16
615 call is_the_square_occupied   ;call routine to check if the square is
        full
    cpi r23,$00
617 breq moving_right_2       ;if space is free, branch with no problems
    cp r20,r17            ;the space is not free. is this the final position?
619 breq moving_right_3       ;if this is the final position, then more tests
         are required, so branch on
    jmp moving_error          ;if this is not the final position, then the
        movement is not possible regardless of colour
621 ;
    moving_right_2:         ;space is free
623 cp r20,r17            ;is it the final position?
    brne moving_right         ;if no, then branch back to move right another
        square
625 ldi r22,$01           ;if this is the final position, then the move is
        valid
    ret
627 ;
    moving_right_3:         ;space is not free but is also final destination
629 cp r23,r24            ;compare colours of pieces
    breq moving_error_jump3     ;if the piece you are moving is the same
        colour as the piece in the final square, the move is not valid
631 ldi r22,$01           ;if the piece you are moving is a different colour
```

```
    in the final square, then the move is valid
RET
;
moving_error_jump3:       ;need to jump because relative branch out of
    range
jmp moving_error
;
;
    ********************************************************************************

;************************** Moving left
    *********************************
;
    ********************************************************************************

;
moving_left:
subi r20,$10          ;move left one
call is_the_square_occupied   ;call routine to check if the square is
    full
cpi r23,$00
breq moving_left_2        ;if space is free, branch with no problems
cp r20,r17            ;the space is not free. is this the final position?
breq moving_left_3        ;if this is the final position, then more tests
     are required, so branch on
jmp moving_error          ;if this is not the final position, then the
    movement is not possible regardless of colour
;
moving_left_2:          ;space is free
cp r20,r17            ;is it the final position?
brne moving_left        ;if no, then repeat loop
ldi r22,$01           ;if this is the final position, then the move is
    valid
ret
;
moving_left_3:          ;space is not free but is also final destination
cp r23,r24            ;compare colour of pieces
breq moving_error_jump4     ;if the piece you are moving is the same
    colour as the piece in the final square, the move is not valid
ldi r22,$01           ;if the piece you are moving is a different colour
    in the final square, then the move is valid
RET
;
moving_error_jump4:       ;need to jump because relative branch out of
    range
jmp moving_error
;
;
;
    ********************************************************************************

;************************ Moving forward diagonal right
    ************************
;
    ********************************************************************************
```

```
669 ;
    moving_forward_diagonal_right:
671 push r16
    ldi r16,$11
673 add r20,r16
    pop r16                 ;move forward diagonal right 1 square
675 call is_the_square_occupied       ;call routine to check if the square is
        full
    cpi r23,$00
677 breq moving_forward_diagonal_right_2    ;if space is free, branch with no
        problems
    cp r20,r17              ;the space is not free. is this the final
        position?
679 breq moving_forward_diagonal_right_3  ;if this is the final position,
        then more tests are required, so branch on
    jmp moving_error            ;if this is not the final position, then
        the movement is not possible regardless of colour
681 ;
    moving_forward_diagonal_right_2:    ;space is free
683 cp r20,r17              ;is it the final position?
    brne moving_forward_diagonal_right    ;if no, then repeat loop
685 ldi r22,$01             ;if this is the final position, then the move
        is valid
    ret
687 ;
    moving_forward_diagonal_right_3:    ;space is not free but is also final
        destination
689 cp r23,r24              ;compare colour of pieces
    breq moving_error_jump5         ;if the piece you are moving is the same
        colour as the piece in the final square, the move is not valid
691 ldi r22,$01             ;if the piece you are moving is a different
        colour in the final square, then the move is valid
    RET
693 ;
    moving_error_jump5:         ;need to jump because relative branch out
        of range
695 jmp moving_error
    ;
697 ;
        ********************************************************************************

    ;************************ Moving backwards diagonal left
        **********************
699 ;
        ********************************************************************************

    ;
701 moving_backwards_diagonal_left:
    subi r20,$11            ;move backwards diagonal left 1 square
703 call is_the_square_occupied       ;call routine to check if the square is
        full
    cpi r23,$00
705 breq moving_backwards_diagonal_left_2   ;if space is free, branch with no
        problems
```

```
    cp r20,r17                ;the space is not free. is this the final
        position?
707 breq moving_backwards_diagonal_left_3 ;if this is the final position,
        then more tests are required, so branch on
    jmp moving_error              ;if this is not the final position, then
        the movement is not possible regardless of colour
709 ;
    moving_backwards_diagonal_left_2:   ;space is free
711 cp r20,r17                ;is it the final position?
    brne moving_backwards_diagonal_left   ;if no, then repeat loop
713 ldi r22,$01               ;if this is the final position, then the move
        is valid
    ret
715 ;
    moving_backwards_diagonal_left_3:   ;space is not free but is also final
        destination
717 cp r23,r24                ;compare colour of pieces
    breq moving_error_jump6         ;if the piece you are moving is the same
        colour as the piece in the final square, the move is not valid
719 ldi r22,$01               ;if the piece you are moving is a different
        colour in the final square, then the move is valid
    RET
721 ;
    moving_error_jump6:           ;need to jump because relative branch out
        of range
723 jmp moving_error
    ;
725 ;
        ********************************************************************************

    ;************************ Moving forward diagonal left
        **********************
727 ;
        ********************************************************************************

    ;
729 moving_forward_diagonal_left:
    subi r20,$F             ;move forward diagonal left 1 square
731 call is_the_square_occupied       ;call routine to check if the square is
         full
    cpi r23,$00
733 breq moving_forward_diagonal_left_2   ;if space is free, branch with no
        problems
    cp r20,r17                ;the space is not free. is this the final
        position?
735 breq moving_forward_diagonal_left_3   ;if this is the final position,
        then more tests are required, so branch on
    jmp moving_error              ;if this is not the final position, then
        the movement is not possible regardless of colour
737 ;
    moving_forward_diagonal_left_2:     ;space is free
739 cp r20,r17                ;is it the final position?
    brne moving_forward_diagonal_left   ;if no, then repeat loop
741 ldi r22,$01               ;if this is the final position, then the move
        is valid
```

```
    ret
743 ;
    moving_forward_diagonal_left_3:     ;space is not free but is also final
        destination
745 cp r23,r24                  ;compare colour of pieces
    breq moving_error_jump7         ;if the piece you are moving is the same
        colour as the piece in the final square, the move is not valid
747 ldi r22,$01                 ;if the piece you are moving is a different
        colour in the final square, then the move is valid
    RET
749 ;
    moving_error_jump7:
751 jmp moving_error
    ;
753 ;
        ********************************************************************************

    ;************************ Moving backwards diagonal right
        **********************
755 ;
        ********************************************************************************

    ;
757 moving_backwards_diagonal_right:
    push r16
759 ldi r16,$F
    add r20,r16
761 pop r16                     ;move backwards diagonal right 1 square
    call is_the_square_occupied       ;call routine to check if the square is
         full
763 cpi r23,$00
    breq moving_backwards_diagonal_right_2  ;if space is free, branch with no
         problems
765 cp r20,r17                  ;the space is not free. is this the final
        position?
    breq moving_backwards_diagonal_right_3  ;if this is the final position,
        then more tests are required, so branch on
767 jmp moving_error                ;if this is not the final position, then
        the movement is not possible regardless of colour
    ;
769 moving_backwards_diagonal_right_2:    ;space is free
    cp r20,r17                  ;is it the final position?
771 brne moving_backwards_diagonal_right    ;if no, then repeat loop
    ldi r22,$01                 ;if this is the final position, then the move
        is valid
773 ret
    ;
775 moving_backwards_diagonal_right_3:    ;space is not free but is also
        final destination
    cp r23,r24                  ;compare colour of pieces
777 breq moving_error_jump8         ;if the piece you are moving is the same
        colour as the piece in the final square, the move is not valid
    ldi r22,$01                 ;if the piece you are moving is a different
        colour in the final square, then the move is valid
779 RET
```

```
;
moving_error_jump8:           ;need to jump because relative branch out
    of range
jmp moving_error
;
;
    ********************************************************************************

;******************************** Movement error
    ***********************************************
;
    ********************************************************************************

moving_error:
ldi r22,$00               ;return with move not valid by setting r22=$00
RET


;
    ***************************************************************************

;************************ Is the square occupied?
    ***************************
;
    ***************************************************************************

;This routine checks whether the intermediate step (r20) square is
    occupied, and
;if it is occupied, by what colour piece.
;
; r23 = is the space occupied?  $00 = unoccuped   $01 = white occupied
    $10 = black occupied
;
is_the_square_occupied:
push r21
push XL
push XH
;
ldi XH,$01
mov XL,r20
ld r21,X                ;load information about piece at intermediate
    step
;
lsr r21                 ;isolate two highest bits
lsr r21
lsr r21
lsr r21
lsr r21
lsr r21
;
cpi r21,$02               ;compare two highest bits to definite values
breq square_is_occupied_by_white    ;if 0000 0010 square is occupied by a
     white piece
cpi r21,$03               ;if 0000 0011 square is occupied by a white
    piece
```

```
819 breq square_is_occupied_by_black
    ;
821 ldi r23,$00               ;if 0000 0000 square is not occupied
    pop XH
823 pop XL
    pop r21
825 ret
    ;
827 square_is_occupied_by_white:
    ldi r23,$01               ;identify white piece by setting r23=$01
829 pop XH
    pop XL
831 pop r21
    ret
833 ;
    square_is_occupied_by_black:
835 ldi r23,$10               ;identify black piece by setting r23=$10
    pop XH
837 pop XL
    pop r21
839 ret
    ;
        ********************************************************************************
```

```
;
    ********************************************************************************

;
    ********************************************************************************

;*******************    Save game interrupt
    ******************************
;
    ********************************************************************************

;
    ********************************************************************************

; Author: Richard Flint
; Date: 27/02/2015
;
    ********************************************************************************************************

; When the user wants to stop the game, they must press ESC on the port 6
    pin.
; This triggers the 0 external interrupt.
; This routine asks whether the player is sure they want to leave. If
    they
; do leave, then their game is saved.
;
; The program does not run in isolation. It is included in the
    main_program.asm,
; and is called as a routine (call move_verification_algorithm).
;
    ********************************************************************************

EXT_INT0:
  in r4,SREG            ; It is advised to store and return SREG, so I do
     it
  ;
  push r20
  push r16
  ;
  rcall CLRDIS          ; Clear LCD screen
  rcall Mess8out          ; "Are you sure you want to exit? (Y/N)",0
  ;
  interrupt_loop:         ; Loop waits for confirmation on whether the
    user wants to exit
    in r16,pind
    com r16
    cpi r16,$08
    breq save_game        ; If user presses [Y], branch to save_game
    routine
    cpi r16,$10
    brne interrupt_loop     ; Loop until user presses [N]
    ;             ; If they do press [N], we need to return back to the
    game
    pop r16
    pop r20
```

```
36      ;
        out SREG, r4
38      ;
        sei             ; return global interrupt to 1 (instead of using reti
        )
40      rjmp return_game        ; jump back to game
;
42 ;
        ********************************************************************************
;*******************    Save game function
        *********************************
44 ;
        ********************************************************************************
;
46 save_game:
rcall CLRDIS
48 rcall MessGameSavedOut      ;"Exit completed!"
call BigBigDEL
50 call BigBigDEL
jmp Init
52
return_game:
54 push r16
push XL
56 push XH
ldi XH,$01
58 ldi XL,$10
ld r16,X
60 cpi r16,$00
breq return_initialisation
62 pop XH
pop XL
64 pop r16
jmp user_input_position
66
return_initialisation:
68 pop XH
pop XL
70 pop r16
jmp play_demo
```

```
;*****************************************************************************
;*****************************************************************************
;************************ USER INPUT POSITION ********************************
;*****************************************************************************
;*****************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;*****************************************************************************
; This code allows the user to input both the coordinates of the piece he/she wants to move (start
; position), and the coordinates of the destination (end position). It then outputs this information,
; along with information of the type of piece being moved.
;
;The program does not run in isolation. It is included in the main_program.asm,
;and is called as a routine (call valid_or_invalid)
;*****************************************************************************
;
; INPUT:  Must have pieces set on the board
; OUTPUT:   r16 gives start location
;     r17 gives destination
;     r18 gives piece information of piece in r16
;     r24 gives the selected piece colour             $01 = white     $10 = black
;     r25 gives the colour that the selected piece should be  $01 = white
          $10 = black
;
; Routine is called using: call user_input_position
;
;*****************************************************************************
;************************ Buttons for reference ******************************
;*****************************************************************************
; PIND = 00000001 ($01) ----> ESC
; PIND = 00000010 ($02) ----> Enter
; PIND = 00000100 ($04) ----> Backspace
; PIND = 00001000 ($08) ----> Y
```

```
; PIND = 00010000 ($10) ---> N
; PIND = 00100000 ($20) --->
; PIND = 01000000 ($40) --->
; PIND = 10000000 ($80) --->
;
    ********************************************************************************

user_input_position:        ;call the entire routine
;
rcall select_colour_piece_message     ;call a brief message prompting "Select piece..."
rcall Main2             ;Main2 is the part of the code that records user input
rcall ascii_to_coordinate_mapping ;map the keypad input to ASCII values
rcall save_initial_position     ;initial position is saved in r16
;
rcall load_piece_information    ;load piece information for piece in r16, and save in r18
rcall check_piece_is_present    ;check that there is a piece in that square
rcall identify_colour       ;identifies the colour of the selected piece, saves in r24
rcall check_piece_colour      ;check that the piece selected is the right colour
;
rcall select_destination_message  ;call a brief message stating "Select destination..."
rcall Main2             ;Main2 is the part of the code that records user input
rcall ascii_to_coordinate_mapping ;map the keypad input to ASCII values
rcall save_final_position     ; final position is saved in r17
;
ret                 ;return to main_program with r16,r17,r18 all recorded
;
;
    ********************************************************************************
     ;
Main2:
  ;letter:
        rcall CLRDIS      ; Clear display
        rcall Mess1out    ; "Input letter"
        rcall keyboard_input  ; Loops until letter entered
        rcall keyboard_to_ASCII_letter  ; Converts input to ASCII value
        sts $C000, r20    ; Immediately output letter. Letter temporarily stored in r20
        rcall busylcd
        ;
        rcall Mess5spacesOut  ; Adds some spaces to display for nicer LCD output
        rcall Mess2out    ; "Press enter/delete"
        rcall enter_button  ; Loops until enter or delete button pressed
        rcall save_letter   ; If enter is pressed, letter will be saved in XH
        ;
  ;number
```

```
73          rcall CLRDIS        ; Clear display
            rcall Mess3out      ; "Input number"
75          rcall keyboard_input  ; Loops until number entered
            rcall keyboard_to_ASCII_number  ; Converts input to ASCII value
77          sts $C000, r20      ; Immediately output letter
            rcall busylcd
79          ;
            rcall Mess5spacesOut  ; Adds some spaces for nicer LCD output
81          rcall Mess2out      ; "Press enter/delete"
            rcall enter_button  ; Loops until enter or delete button pressed
83          rcall save_number   ; If enter is pressed, number will be saved
      in XL
            ;
85    ;display coordinates:
            rcall CLRDIS
87          rcall Mess4out      ; "Your coordinate: "
            rcall display_coordinate ; Outputs coordinate on LCD screen
89          rcall Mess1spacesOut  ; Adds a space for nicer LCD output
            rcall Mess6Out      ; "Confirm? (Y/N)"
91          rcall yes_button    ; If Y button is pressed, player has
      committed to move
            rcall CLRDIS        ; Clear display
93          ;
            ret ;return to where user_input_position was called
95  ;
    ;
97  ;
    ;
99  ;
      ********************************************************************************

    ;*******************   Keyboard input
      ************************************
101 ;
      ********************************************************************************

    ;
103 keyboard_input:
    push r16
105 push r18
    push r19
107 ;
    ; Part 1: Set the initial value to $0F
109 ;
    ldi r16, $F0      ; I/O (determins which pins are inputs and which are
      outputs)
111 out DDRE, r16     ; Port D Direction Register
    ldi r16, $0F      ; Initial value
113 out PORTE, r16      ; Port D value
    ;
115 rcall DEL49ms
    ;
117 keyboard_loop2:
    in r19,PINE       ; Read input from keypad
119 cpi r19,$0F       ; Check if any button has been pressed
```

```
    breq keyboard_loop2 ; If no button pressed, then continue looping. If
      button pressed, carry on
121 ;
    ; Part 2: Set the initial value to $F0
123 ;
    ldi r16, $0F    ; I/O (determins which pins are inputs and which are
      outputs)
125 out DDRE, r16   ; Port D Direction Register
    ldi r16, $F0    ; Initialise value
127 out PORTE, r16    ; Port D value
    ;
129 rcall DEL49ms
    ;
131 in r18,PINE     ; Read input from keypad
    cpi r18,$F0
133 breq keyboard_error ;If they have taken their finger off the button too
       fast, it will register an error
    ;
135 add r19,r18     ;combine coordinates
    mov r20,r19     ;move to r20 for saving
137 ;
    pop r19
139 pop r18
    pop r16
141 ;
    ret         ;returns to Main2 with coordinates temporarily saved in r20
143 ;
    keyboard_error:       ;if the user has pressed they key too fast, it
      returns an error
145 call CLRDIS
    call MessButtonErrorOut   ;message out the error
147 call BigBigDEL        ;wait a bit
    call BigBigDEL
149 call CLRDIS
    call MessPressAgainOut    ;ask them to press again
151 pop r19
    pop r18
153 pop r16
    jmp keyboard_input      ;jump back to the keyboard input
155 ;
    ;
157 ;
    ********************************************************************

;******************   Select pieces messages
    ****************************
159 ;
    ********************************************************************

    ;
161 select_colour_piece_message:
    call CLRDIS
163 cpi r25,$01
    breq select_white_piece_message
165 cpi r25,$10
```

```
    breq select_black_piece_message
167 ret
    ;
169 select_white_piece_message:
    call MessSelectWhitePieceOut
171 call BigBigDEL       ;message is only temporary
    call BigBigDEL
173 ret
    select_black_piece_message:
175 call MessSelectBlackPieceOut
    call BigBigDEL       ;message is only temporary
177 call BigBigDEL
    ret
179 ;
    ;
        ********************************************************************************

181 ;********************    Select destination messages
        *************************
    ;
        ********************************************************************************

183 select_destination_message:    ;a temporary message notifying the stage of
         piece selection
        call CLRDIS
185     call MessSelectDestinationOut ;"Select destination..."
        call BigBigDEL       ;message is only temporary
187     call BigBigDEL
        ret
189 ;
    ;
        ********************************************************************************

191 ;********************    Temporarily save values
        ****************************
    ;
        ********************************************************************************

193 ; Temporarily saves values into X
    save_letter:
195     mov XH, r20
        ret
197 ;
    save_number:
199     mov XL, r20
        ret
201 ;
    ;
        ********************************************************************************

203 ;********************    Display coordinate
        ***********************************
    ;
        ********************************************************************************
```

```
205 ; Displays the coordinates temporarily stored in X
    ;
207 display_coordinate:
      sts $C000, XH
209   rcall busylcd
      sts $C000, XL
211   rcall busylcd
      ret
213 ;
    ;
215 ;
        ***************************************************************************

    ;************************ Save positions
        ***********************************
217 ;
        ***************************************************************************

    ;If move is accepted, then coordinates are transferred to appropriate
        registers
219 ;
    save_initial_position:
221   mov r16,XL          ;saves start position into r16 for use in rest of
        code
      ret
223 save_final_position:
      mov r17,XL          ;saves destination in r17 for use in rest of code
225   ret
    ;
227 ;
        ***************************************************************************

    ;************************ Check piece colour
        *******************************
229 ;
        ***************************************************************************

    ;
231 check_piece_colour:      ;checks whether the square is filled with a piece
    ;             of the correct colour for whose turn it is
233 cp r25,r24
    brne colour_not_ok
235 ret              ;
    ;
237 colour_not_ok:
    call MessWrongColourOut    ;"You selected the wrong colour. Try again."
239 call BigBigDEL
    call BigBigDEL
241 jmp user_input_position    ;Return back to allow user another go.
    ;
243 ;
    ;
        ;***************************************************************************

245 ;************************ Check piece is present
```

```
    ************************
;
    ********************************************************************************

;Checks whether a piece is present in the start square selected
;
check_piece_is_present:
push r18
;
lsr r18             ;isolate most significant bit in piece information
lsr r18
lsr r18
lsr r18
lsr r18
lsr r18
lsr r18
cpi r18,$01           ;if most significant bit = $01, then square is
    occupied.
brne no_piece_present
pop r18
ret               ;if a piece is present, then return back with no
    problems
;
no_piece_present:       ;if there is no piece present, we must
    communicate this
push r20
rcall CLRDIS
rcall MessNoPiecesOut     ;"No piece selected.  Press enter to retry"
;
no_piece_present_loop:      ;loop until enter is pressed
  in r20,pind
  com r20
  cpi r20,$02
  brne no_piece_present_loop
  ;
  pop r20
  pop r18
  jmp user_input_position   ;using jmp is fine, but remember no ret has
    been used
  ret
;
;
    ********************************************************************************

;******************   Enter/delete button
    ********************************
;
    ********************************************************************************

;
enter_button:     ;routine loops until [enter] or [delete] is pressed
  push r23
  enter_button_loop:
    in r23,pind    ; input from pin d buttons on microprocessor board
    com r23      ; com input because pin d is inverted
```

```
289     ;
        cpi r23,$04    ; branch if delete button is pressed
291     breq delete_button
        ;
293     cpi r23,$02    ; loop until enter button is pressed
        brne enter_button_loop
295   pop r23
      ret
297
delete_button:
299   pop r23
      jmp Main2      ;if delete button is pressed, the program returns to the
        beginning
301   ;          ;using jmp is fine, but remember no ret has been used

303 ;
      ********************************************************************************

;********************    Yes/no button  ******************************
305 ;
      ********************************************************************************

;
307 yes_button:        ;routine loops until [Y] or [N] is pressed
      push r23
309   yes_loop:
        in r23,pind    ; input from pin d buttons on microprocessor board
311     com r23      ; com input because pin d is inverted

313     cpi r23,$10    ; branch if N button is pressed
        breq no_button
315
        cpi r23,$08    ; loop until Y button is pressed
317     brne yes_loop
      pop r23
319   ret

321 no_button:
      pop r23
323   jmp Main2      ; if N button is pressed, the program returns to the
        beginning
      ;          ;using jmp is fine, but remember no ret has been used
325 ;
;
      ********************************************************************************

327 ;************************ Load piece information
      ****************************
;
      ********************************************************************************

329 ;
load_piece_information:
331 push XH
    push XL
```

```
333 ;
    mov XL,r16
335 ldi XH,$01    ; piece information is stored at $01XX where XX is the
        coordinate
    ;
337 ld r18,X    ; stored in r18
    ;
339 pop XL
    pop XH
341 ret
    ;
343 ;
        ********************************************************************************

    ;*********************      Identify colour
        ****************************
345 ;
        ********************************************************************************

    identify_colour:
347 mov r24,r18 ;save piece type in r24 to identify colour
    ;
349 lsl r24   ;isolate colour bit
    lsr r24
351 lsr r24
    lsr r24
353 lsr r24
    lsr r24
355 lsr r24
    ;
357 cpi r24,$00
    breq white_piece
359 ldi r24,$10     ;black piece
    ret
361 white_piece:    ;white piece
    ldi r24,$01
363 ret
    ;
```

```
;
    ********************************************************************************
;******************** VALID OR INVALID CHECK
    *******************************
;
    ********************************************************************************
; Author: Richard Flint
; Date: 27/02/2015
;
    ********************************************************************************
;This code applies a simple branch based on whether the Move Verification
;Algorithm determined the move to be valid or invalid.
;
;The program does not run in isolation. It is included in the
    main_program.asm,
;and is called as a routine (call valid_or_invalid)
;******************** Input registers for reference
    ************************
; r22 = valid move?       $00 = not valid   $01 = valid
; r22 is the overall output from the Move Verification Algorithm
;
    ********************************************************************************
;
valid_or_invalid:     ;call the routine
cpi r22,$00          ;compare the output (r22) of the Move Verification
    Algorithm
breq invalid_move     ;if r22=$00, it means the move is not valid
cpi r22,$01
breq valid_move       ;if r22=$01, it means the move is valid
rjmp invalid_move     ;if neither of these, then there has been some
    error...
;
invalid_move:        ;invalide move routine
call CLRDIS
call MessInvalidMoveOut   ;"Move not valid.Press enter to try again"
push r20
invalid_loop:        ;Loop until enter is pressed
in r20,pind
com r20
cpi r20,$02
brne invalid_loop
pop r20
call CLRDIS
ret              ;Once enter is pressed, return to main program without
;             ;any move being made. Player can try again.
;
valid_move:          ;valid move routine
call CLRDIS
call MessValidMoveOut   ;"Move is valid. Press enter to continue"
push r20
valid_loop:          ;Loop until enter is pressed
```

```
in r20,pind
com r20
cpi r20,$02
brne valid_loop
pop r20
call CLRDIS
rcall save_valid_move    ;Once enter is pressed, call routine to save move into the SRAM
ret              ;Return back to main program
;
save_valid_move:       ;routine to save move into the SRAM
push XL
push XH
push r20
;
ldi XH,$01           ;store destination in X
mov XL,r17
st X,r18           ;move piece information into destination square
;
mov XL,r16           ;ensure that the start square is cleared
ldi r20,$00          ;this means that there is no piece in the start square
st X,r20           ;store this in the SRAM
;
pop r20
pop XH
pop XL
ret
;
    ********************************************************************************
```