[redacted]

Programming Project

The remaining assignments for the course will be based around the game of Draughts (also known as Checkers). Programming a full game is a complicated endeavor so it will be broken down into basic functions. Subsequent homework assignment will ask you to implement some sub-function of the game. Depending on time we may integrate the homework assignments at the end into a final program.

Background:

Draughts is a two-player game played on a board of size 8x8. The players are labeled as “red” and “white” with red going first. The game starts with each player having 12 game pieces placed in a fashion as shown in figure 1 on the dark squares.

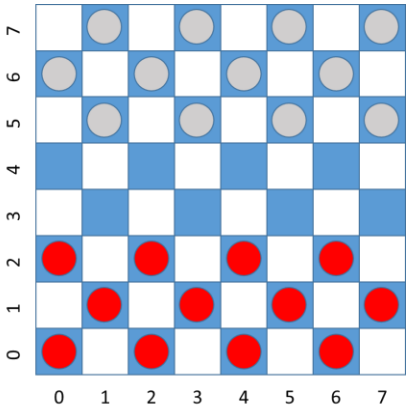

Figure 1

Basic Movement:

A piece may move along the diagonal by 1 space only in the forward direction. The first three moves of a sample game are shown in figure 2. Game pieces may not be moved backwards unless they become a king. If a jump is possible then the jump move must be taken.

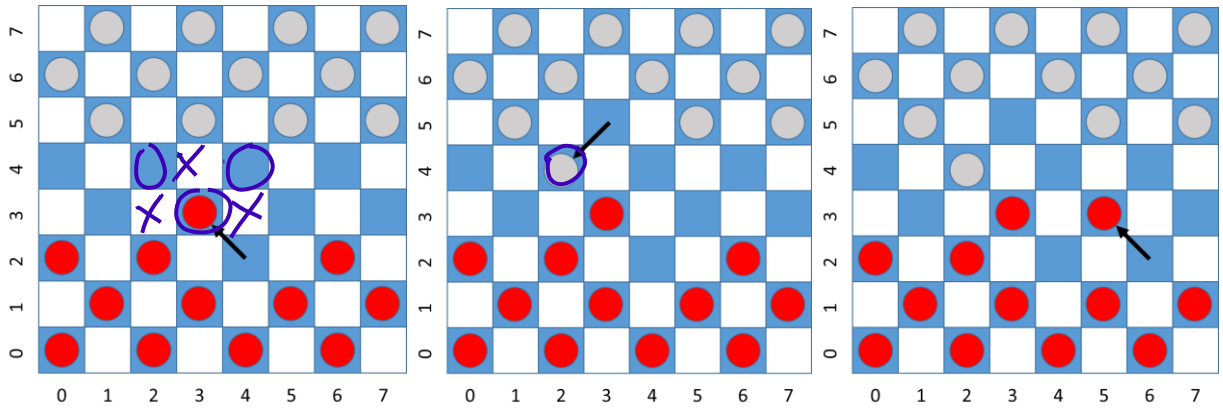

Figure 2

(r+1) (c-1) (c+1)

Jumping:

When the opportunity presents itself, a piece must jump an opposing piece. In the presence of more than one jump possibility, the player may choose which jump to take. A jump possibility is presented to the white player in figure 2 on the right. From this position, it is possible for the white player piece at (2,4) to jump red, and remove the red piece at (3,3) from the board as shown in figure 3.

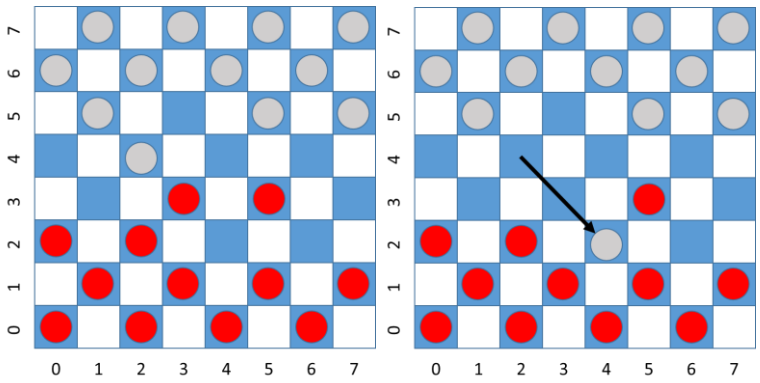

Figure 3

From here, red must use the piece at (5,1) to jump to (3,3) and remove the white piece at board location (4,2).

Multiple Jumps:

It is possible to use the same checker to jump, and remove, multiple checkers of the opposing player. Consider figure 4.

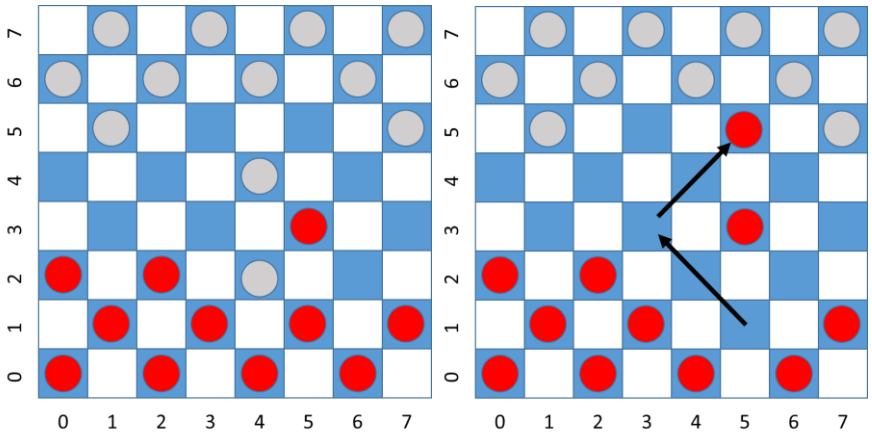

Figure 4

Here we see the red piece at location (5,1) jumps twice in a single turn and takes two white pieces from locations (4,2) and (4,4). If it is possible to jump multiple times, then the piece must make multiple jumps.

Kings:

If a piece makes it to the last row (row 0 for white, row 7 for red) then that piece turns into a king and may move backwards. The king must jump backwards if the opportunity presents itself. Multiple jump rules also apply.

Practical details:

In MIPS assembly, there are a limited number of color choices. We will use black for the illegal squares and white (spacebar) for the legal squares where game pieces are placed. The letter 'r' is used for red game pieces while the letter 'w' is used for white game pieces. The following list of characters can be used with the $print_char system call to denote pieces in individual squares (see appendix B of the textbook):

- ASCII 32 (spacebar): empty square.
- ASCII 114 ('r'): regular red game piece.
- ASCII 82 ('R'): red king.
- ASCII 119 ('w'): regular white game piece.
- ASCII 87 ('W'): white king.
- ASCII 219: an empty square of the opposite color.

We will need to define the board data structure. The board requires 64 squares in an 8x8 grid. At each grid square, we have several possibilities: a) a checker can be on the square or not, b) the checker could be a king, or not, c) the checker could be either red or white. Since multidimensional arrays are not possible in MIPS32, we define the board as a single array of integers:

```
unsigned int board[64];
```

Each integer represents a square on the board. Given a *row* and *column*, the location on the board can be referenced to assign a value *<val>* as follows:

```
board[row*8+column]=<val>;
```

You can multiply the row number "*row*" by 8 and add the column number "*column*" to get the information about the square located at (row,column).

If a board location has no game piece, its value is "0". If the board value is non-zero, there is a game piece. The bits of each board location are defined as follows:

X	X	X	X	X	K	C	E

- E: if 1, there is a checker at this board location, 0 if no checker.
- C: If 1, the color is white, 0 if the color is black (red).
- K: If 1, the checker is a king, 0 if the checker is normal.

W: 0x0111 = 7
w: 0x0011 = 3
r: 0x0001 = 1
R: 0x0101 = 5

Some example values for board locations and their values:

- board[7*8+1]=0x00 ➔ No checker at location (7,1)

- board[4*8+2]=0x05 ➔ A black king is located at (4,2) with bits 0000 0101
- board[2*8+6]=0x03 ➔ A white checker is located at (2,6) with bits 0000 0011
- board[r*8+c]=0x01 ➔ A black checker is located at (r,c) with bits 0000 0001

Notice that the top bits are always ignored so you can safely declare these to be zero to simplify coding. **Also, notice that the codes stored in the array of 36 integers are very different than the codes used to display the board—for instance, do not store code 32 in the array, store the code 0x0 instead**.