Computer Architecture
CPSC 321, Fall Semester 2004
Lab Assignment # 4
Due: One week after your lab session – complete in a team of up to 2 people

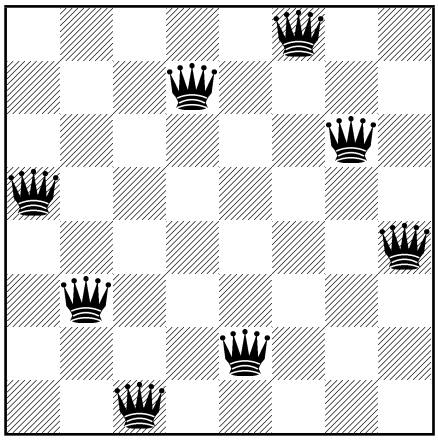
1 Objective

Backtracking is an important algorithmic technique that, basically, explores the solution space by a depth-first search. Backtracking is most naturally formulated in terms of recursive programs. This assignment will help you to further develop your MIPS programming skills. You can work in a team, so that you can gain experience in collaborative software develoment.

2 The n Queens Problem

Suppose that we are given an $n \times n$ chessboard, where n is a positive integer. A queen can attack in horizontal, vertical, and the two diagonal directions. The **n-queens problem** asks to find a configuration of n queens on an $n \times n$ chessboard such that no two attack one another.

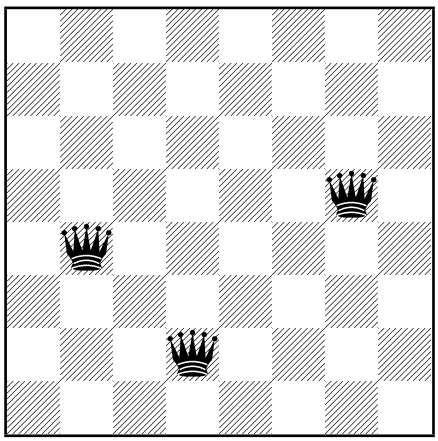
There are no solutions to this problem for $n = 2$ or 3, but it can be shown that a solution exists for each $n \geq 4$. For instance, the following figure shows a solution for the traditional 8×8 chessboard.



Your task is to program a MIPS assembly language program that solves the n-queens problem with backtracking. We have included a solution in a high-level language that can serve as a guide.

3 Prolegomena

Apparently, any solution to the n-queens problem has exactly one queen in each row. We can use an array `row` to store in `row[i]` the position of the queen in the ith row. Since the queens have to be in different columns, it follows that a solution has to satisfy `row[i]`$\neq$`row[j]` for all distinct i and j.



The two queens in the ith and jth row are on the same diagonal if and only if $|i-j|=|$`row[i]` $-$ `row[j]`$|$, as the above example illustrates.

The main idea is to proceed row by row. Suppose that we have already selected the values of `row[k]` for all $0 \leq k < i$. We try to select a value for `row[i]` that is consistent with the requirements that we have outlined above. If that is not possible, then we backtrack and choose a different value for `row[i-1]`.

One possible solution is outlined in the following pseudocode fragment:

```
bool is_admissible(int i) {

  for(int k=0; k<i ; k++) {
    if(row[i] == row[k] or abs(row[i]-row[k]) == i-k)
      return false;
  }
  return true;
}
```

```
void queens(int i) {

  if(is_admissible(i))
    if (i==n-1) {
      print_board();
      exit; // finding one configuration is enough.
    }
    else
      for(int j=0; j<n; j++) {
        row[i+1] = j;
        queens(i+1);
      }
}
```

The call `queens(-1)` determines a valid assignment values to `row[j]`, $0 \leq j < n$, such that the queens at positions (`j`, `row[j]`) do not threaten each other (if such a configuration exists).

4 The Task

Write a MIPS assembly language equivalent of the above code fragment. If your program is executed for instance on the SPIM simulator, then the user should be prompted to provide a size n. If the user enters for instance 6, then your programs should output a valid configuration of 6 queens on the chess board, such as

```
Please provide the size n = 6
-*----
---*--
-----*
*-----
--*---
----*-
```

[**25 points**] Write a procedure `print_board` to print the configuration that is stored in the array `row`. A queen should be represented by a star '`*`' and an empty field by a dash '`-`'. The value of `row[i]` ranges between 0 and $n-1$ and represents the position of the column that contains the queen of the ith row; the rows range from $0 \leq i < n$.

[**20 points**] Write an procedure `is_admissible` that takes an argument `$a0`=i and checks whether the value in `row[i]` is consistent with the values contained in rows `row[k]` with $0 \leq k < i$. The procedure can

assume that the values `row[k]`, with $0 \leq k < i$, represent a valid configuration, meaning that the queens are in distinct columns and no two are on the same diagonal.

[35 points] Write a recursive procedure `queens` that calculates a valid configuration of n queens. Write a main program that prompts the user to input n, calculates and prints a solution, and repeats this cycle.

[20 points] It is important that you plan your task and take advantage of the fact that you work in a team. Please keep a **design notebook** that documents your design process and all the steps in the program development; it should document the progress that you made while writing your software (keep track of date and time). Write the design notebook while you are developing the software, do not write it after the fact.

Your notebook should keep a log of all the errors that you make; it should be handwritten with a pen that is not erasable; printouts documenting your progress should be glued in. The grading of your design notebook will take into account the clarity, regularity, legibility and organization of your annotations.