ISTANBUL TECHNICAL UNIVERSITY

COMPUTER AND INFORMATICS FACULTY

COMPUTER ENGINEERING

BLG413E

SYSTEM PROGRAMMING

PROJECT 1

REPORTED BY

OZAN ÖZYEĞEN - 150120115

TEAM NUMBER: 17

MEMBERS

Ozan ÖZYEĞEN - 150120115

M. Aziz ULAK - 150130103

Sercan BAYINDIR - 150130104

INSTRUCTOR: TURGUT UYAR

**Description of the assignment 1**

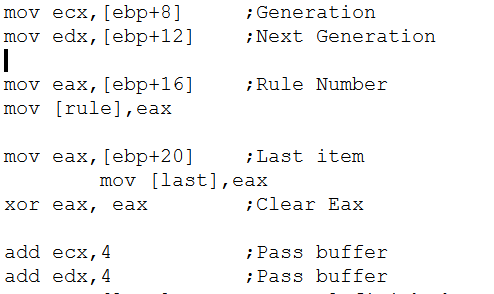
In this project assignment, we were asked to create 1D cellular automaton. Automata must evolve according to a set of rules. The size of rule and number of generations (how many times the program must run) are given by the user on runtime. The size and content of the grid is read from a file whose name must be entered as the first argument by the user.

**Variables and Data Structures**

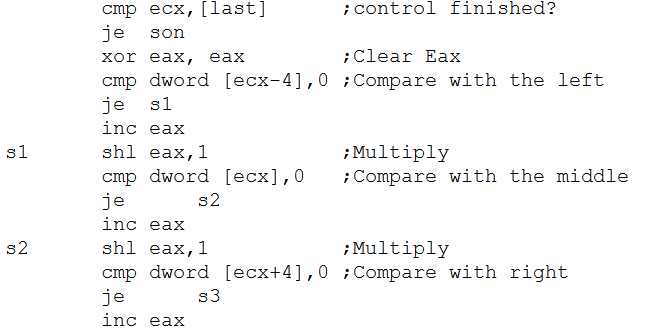
Some of the prominent variables in this program are:

* Gen: It is a dynamically allocated array which holds current generation of the grid. Its size is 2 more than given size value in order to take in consideration of the calculations of the first and last box of the grid.
* Nextgen: It is a dynamically allocated array similar to gen. It holds the new, generated line of the grid.
* Rulenum: Size of the rule
* Gennum: Number of generations the program must ran. For each generation, a new grid is created.

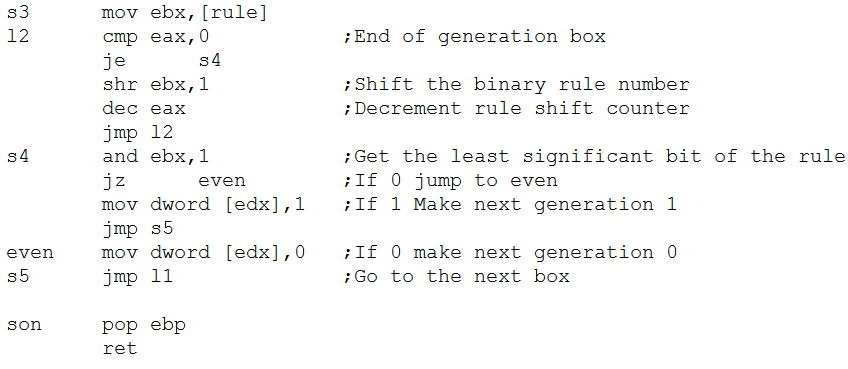
**The brief explanation of the assembly code**



First we reserved areas for rule and the last element. Then, after the prologue, we moved the generation and next generation to ecx and edx. After putting the rule number to reserved space and clearing the eax for future operations, we passed one box in generation and new generation to get the beginning of the grid.



Then we check whether the ecx shows the end of the grid. If it is in the end of the grid, it jumps to the son position. After clearing eax again, we compare the left position of the grid and multiply (shift left) the result with eax. The value of eax helps us to find the position of rule that we will use at each step. Thus, we do the same operations for middle and right boxes.



First we put the rule to ebx. Then, we shift right the rule number for eax number of times. And because of that the lowest significant bit of ebx is now equal to the box in the new generation (next line). In order to get this bit, we made and 1 operation with ebx. And the result is moved into the edx which shows the new generation box.

**Example input and output values**

The size of rule and number of generations (how many times the program must run) are given by the user on runtime. The size and content of the grid is read from a file whose name must be entered as the first argument by the user. Example file read:

8

0 1 0 1 1 0 1 0

If we run the program and enter the rule number as 5 and iterations as 3, the output of the code is shown below.

1.Iteration

0 1 0 0 0 0 1 0

2.Iteration

0 1 0 1 1 0 1 0

3.Iteration

0 1 0 0 0 0 1 0

**Description of the assignment 2**

In this project assignment, we were asked to create 2D cellular automaton. Automata must evolve according to a set of rules. The number of generations (how many times the program must run) is given by the user on runtime. The size and content of the rule and grid are read from a file whose name must be entered as arguments by the user.

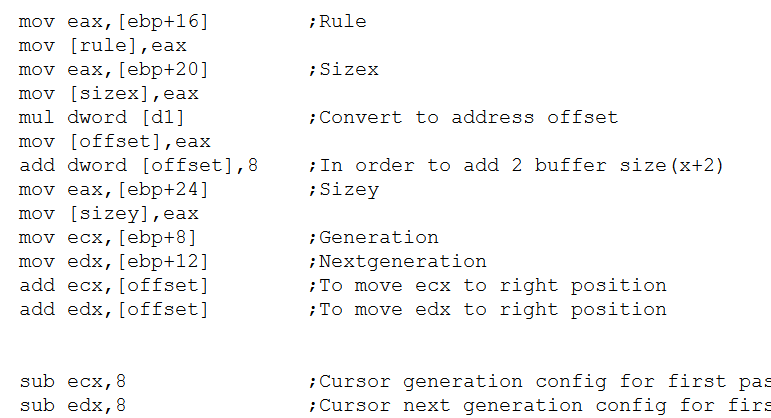
In the C code, we created a function called copythematrix which copies the source matrix to destination matrix. We use it to put next generation to generation matrix for the next generation.

**Variables and Data Structures**

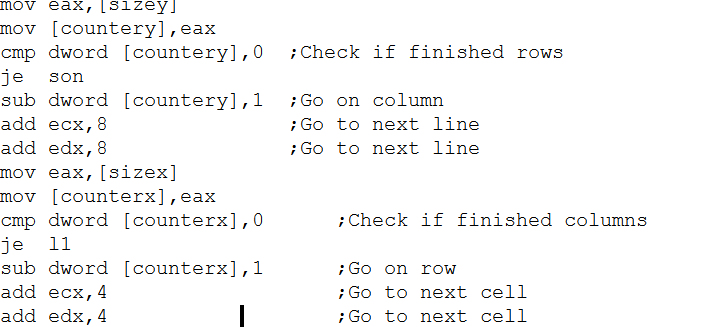
Some of the prominent variables in this program are:

* Gen: It is a dynamically allocated array which holds current generation of the grid. Its size is 2 more in both horizontal and vertical directions in order to take in consideration of the calculations of the first and last rows and columns of the grid.
* Nextgen: It is a dynamically allocated array similar to gen. It holds the new, generated line of the grid.
* Rulenum: Size of the rule
* Gennum: Number of generations the program must ran. For each generation, a new grid is created.
* Sizex: Size of a row
* Sizey: Size of a column

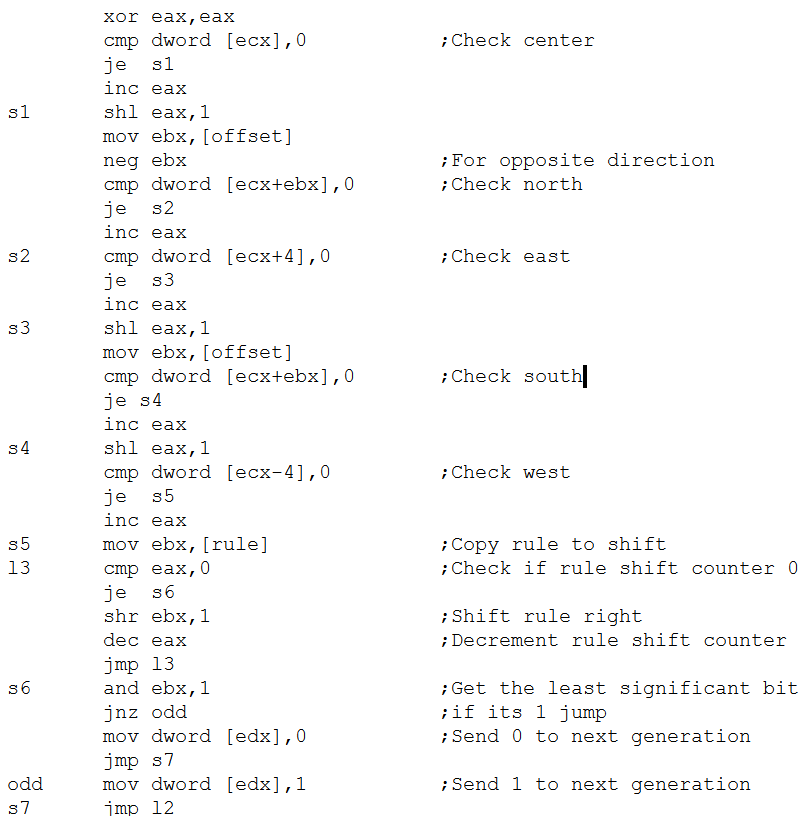
**The brief explanation of the assembly code**



First we reserved areas for rule, sizes, offset and counters. Offset is equal to the length of a row, it helps us to make comparisons with upper and lower boxes. Then, after the prologue, the rule and size is moved to its reserved space and offset is increase to count buffer spaces too. After that, we moved the generation and next generation to ecx and edx. Also the offset is added to get generation and new generation to the right box. Another adjustment is than by sub operation to config them for the first pass.



Now, we check whether the rows are finished or not by comparing sizey to 0. If they finished we jump to end. In this two 2 loops, l1 and l2, we check whether the rows and columns finished.



After cleaning eax with xor operation, all directions start to compared in order starting with the center. We did this in a very similar way to what we do in the first part of assignment. But we used offset to checkout upper and lower boxes, and negation of the offset when we want to look the north. Then like we did in the first part, we shifted the rule eax number of times and made an and operation to get its last significant bit. And add it to the next generation.

**Example input and output values**

The number of generations (how many times the program must run) is given by the user on runtime. The size and content of the rule and grid are read from a file whose name must be entered as arguments by the user. An example of input arguments is given below:

5 5

0 0 0 0 0

0 0 0 0 0

0 0 1 0 0

0 0 0 0 0

0 0 0 0 0

0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

If we run the program and enter the iteration as 3, the output of the code is shown below.

1.Iteration

0 0 0 0 0

0 0 1 0 0

0 1 0 1 0

0 0 1 0 0

0 0 0 0 0

2.Iteration

0 0 1 0 0

0 0 0 0 0

1 0 0 0 1

0 0 0 0 0

0 0 1 0 0

3.Iteration

0 1 0 1 0

1 0 1 0 1

0 1 0 1 0

1 0 1 0 1

0 1 0 1 0