Assignment 3

Further proficiency in assembly language. Understanding stack manipulations, as needed to create a co-routine (equivalently thread) management scheme.

Program Goal

Simulating Conway's Game of Life (for those who are interested, the original game is described here). The program begins by reading the initial state configuration of the organisms managed by the co-routines, the number of generations to run, and the printing frequency (in steps).

The program initializes an appropriate mechanism, and control is then passed to a scheduler co-routine which decides the appropriate scheduling for the co-routines. The states of the organisms managed by the co-routines, where co-routine is responsible for **one** organism, cell in the array. Every organism is a cell in 2 a dimensional array, so it has 8 neighbors (diagonal cells also count as neighbors). Life board is cyclic, it means that each cell in the last column has neighbors from the first column and vice versa. Each cell in the last row has neighbors from the first row and vice versa.

The cell organisms change according to the following rules: if the cell is currently alive, then it will remain alive in the next generation if and only if exactly 2 or 3 of its neighbors are currently alive. Otherwise it dies. A dead cell remains dead in the next generation, unless it has exactly 3 living neighbors, in which case we say that an organism is born here.

A specialized co-routine called the **printer** prints the organism states for all the cells as a two dimensional array.

User Command Line Input

Your program will be run using the following command-line formats:

```
> ass3 <filename> <length> <width> <t> <k>
> ass3 -d <filename> <length> <width> <t> <k>
```

where:

-d (debug) option
If used, you should print at the beginning of the program the input, in a manner that Printer co-routine does, and the values of the command line arguments.

Example:
>ass3 inputExample.txt 4 6 10 3
length=4
width=6
number of generations=10
print frequency=3
0 0 1 0 0 1
1 1 0 0 0 1

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

Initial Configuration
filename is the name of the initial state file. The file include just space ' '(dead) , '1'(alive) and newline character after every width characters. Width and length are the dimensions of the co-routine array. Maximum value of length and width is 100. This file determines organisms initial states. You should allocate an global array called **"state"**, contains organisms state. Every co-routine will be able to reach this array for reading and writing. After reading the arguments and file you need to allocate space and initialize the co-routine structures for the co-routine controlling each cell properly (see coroutine.s, **cors**). Two additional co-routines must be initialized: the scheduler and the printer, as described later.

Number of generations:
The **fourth** command-line argument, *t*, is the number of generations for the scheduler to iterate.

Printing frequency
Next, *k*, is the number of ***steps*** to be done by the scheduler between calls to the printer, i.e. after each *k* "resumes" of cell co-routines, there needs to be one to the printer.

Inputs can be assumed to be correct.

Cell

In any given "time slice", a cell co-routine tries to move from generation j to j+1. It does so in two stages:
(1) read state (its cell, and its neighbors' cell) and compute the new state, but do not store it in the array. Each living organism has an **age** counter of number of generation since the last time it was born, with the maximum count being 9 (at which point it stays 9, until the cell dies).
(2) update global array cell to new state
In our simulation, we act as if there is insufficient time to do both in one "time slice", so it must resume the scheduler in between. In other words, a cell co-routine loops (forever) over: Step (1) –> resume scheduler –> Step (2) –> resume scheduler
Each generation thus consists of the above two stages.

A cell co-routine should gets the cell's indices (x, y) as arguments in C calling convention. Its running "thread" needs to use these indices in order to know which cell the current co-routine represents.

Printer

Once in a while the scheduler transfers control to the printer co-routine. In each time slice the printer prints the **current states** of all cells as a width * length table (use newline character at the end of every line).

Scheduler

After initialization the "main" program transfers control to a scheduler co-routine.

The scheduler receives two arguments: the number of generation , and *k* (as given by the user). The scheduler transfers control to each one of the co-routine cells, and after each *k* transfers, it gives "time" to the printer co-routine.

Every co-routine must be visited **i times** before the scheduler resumes another co-routine in the **i+1 time**. The scheduler returns to the main program when the last generation is reached, after giving one last "time slice" to the printer.

Important Notes:

- The scheduler co-routine MUST be exclusively written in a separate file, the actual control transfer (context switch) should be done with the resume mechanism. Hence, label **resume** would be declared as extern to the scheduler, and register ebx would be used to transfer control.
- The relevant argument needs to be pushed onto the scheduler's stack during the initialization phase, so that they appear as if the scheduler is called with arguments using the C calling convention. Likewise for the cell co-routine cell number.
- You are **required to implement** a simple scheduler which will iterate ***t*** times through the cells (other co-routines) in a round-robin manner.
  Every *k* "resumes" of cell co-routines, the scheduler transfer control to the printer.
- Different implementations of the scheduler **will** be examined by the checker.

Naming conventions:
Some global variables are required (to reduce the amount of parameters, for your convenience):

- As assembly language does not support array sizes, you are required to have global variables named **WorldWidth** and **WorldLength** to hold the number of cells in the simulation. Make sure you initialize it (based on the first command-line argument) before you call the scheduler to start iterations.
- An "array" **cors**, is actually a pointer to ***WorldWidth * WorldLength + 2*** stack tops of:
  scheduler,printer,$cell_1$,$cell_2$,…,$cell_{WorldWidth * WorldLength}$.
  We will use this to "resume" the different co-routines (note that the order is important, since we identify co-routines by their index).

As stated above, we may test your code with different schedulers. Hence, this naming convention is crucial!

What You Have

Note that the co-routines code is greatly simplified: each co-routine is represented by its stack top, and all relevant functions work with co-routine index. You may extend and modify the provided code if necessary. You must **not** use the C standard library, and use system calls instead.

All the assignment code is an assembly code except of the function Cell(x,y) that should be written in C.

Submission Instructions

You are to submit a single zip file containing ass3.s (main), scheduler.s, and printer.s, coroutines.s, and cell.c . Your executable must be named ass3.
Make sure you follow the coding and submission instructions correctly.
Submissions which deviate from these instructions will not be graded!

Skeleton Code

This code is provided as an example, with no guarantees.

- coroutines.s
- scheduler.s
- printer.s
- ass3.s

Good Luck!