Assignment 1: cellular, 1D Cellular Automaton in MIPS

version: 1.0 last updated: 2020-06-26 17:00:00

Aims

- to give you experience writing MIPS assembly code
- to give you experience with data and control structures in MIPS

Getting Started

Create a new directory for this assignment called `cellular`, change to this directory, and fetch the provided code by running these commands:

```
$ mkdir cellular
$ cd cellular
$ 1521 fetch cellular
```

This will add the following files into the directory:

- `cellular.c`: a cellular automaton renderer
- `cellular.s`: a stub assembly file to complete

cellular.c: A Cellular Automaton Renderer

`cellular.c` is an implementation of a one-dimensional, three-neighbour cellular automaton. It examines its neighbours and its value in the previous generation to derive the value for the next generation.

```
. . . . # . . . .
. . . # # # . . .
. . # # . . # . .
. # # . # # # # .
# # . . # . . . #
```

Here, we using `'#'` to indicate a cell that's alive; and `'.'` to indicate a cell that is not.

Given we examine three neighbours, there are eight states that the prior cells could be in. They are:

```
. . .   . . #   . # .   . # #   # . .   # . #   # # .   # # #
  0       1       2       3       4       5       6       7
```

For each one, we decide what action to take. For example, we might choose to have the following 'rule':

```
. . .   . . #   . # .   . # #   # . .   # . #   # # .   # # #
  .       #       #       #       #       .       .       .
```

We apply this rule to every cell, to determine whether the next state is alive or dead; and this forms the next generation. If we print these generations, one after the other, we can get some interesting patterns.

The description of the rule above — by example, showing each case and how it should be handled — is inefficient. We can abbreviate this rule by reading it in binary, considering live cells as 1's and dead cells as 0s; and if we consider the prior states to be a binary value too — the above rule could be `0b00011110`, or `30`.

To use that rule, we would mix together the previous states we're interested in — left, middle, and right — which tells us which bit of the rule value gives our next state.

The size of a generation, the rule number, and the number of generations are supplied on standard input. For example:

```
$ ./cellular
Enter cells size: 60
Enter rule: 30
Enter how many generations: 10

0   ............................#............................
1   ...........................###...........................
2   ..........................##..#..........................
3   .........................##.####.........................
4   ........................##..#...#........................
5   ........................##.####.###......................
6   .......................##..#....#..#.....................
7   .......................##.####..######...................
8   ......................##..#...###.....#..................
9   ......................##.####.##..#...###................
10  .....................##..#....#.####.##..#...............
```

If generations is a negative number, the generations should be printed in reverse.

```
$ ./cellular
Enter cells size: 60
Enter rule: 30
Enter how many generations: -10

10  .....................##..#....#.####.##..#...............
9   ......................##.####.##..#...###................
8   .......................##..#...###.....#..................
7   ........................##.####..######...................
6   .........................##..#....#..#.....................
5   ..........................##.####.###......................
4   ...........................##..#...#........................
3   ............................##.####.........................
2   .............................##..#..........................
1   ..............................###...........................
0   ...............................#............................
```

cellular.s: The Assignment

Your task in this assignment is to implement `cellular.s` in MIPS assembler.

You have been given some assembly and helpful information in `cellular.s`. Read through the provided code carefully, then add MIPS assembly so it executes exactly the same as `cellular.c`.

```
$ 1521 spim -f cellular.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter cells size: 60
Enter rule: 30
Enter how many generations: 10

0   ............................#............................
1   ...........................###...........................
2   ..........................##..#..........................
3   .........................##.####.........................
4   ........................##..#...#........................
5   ........................##.####.###......................
6   .......................##..#....#..#.....................
7   .......................##.####..######...................
8   ......................##..#...###.....#..................
9   ......................##.####.##..#...###................
10  .....................##..#....#.####.##..#...............
```

To test your implementation, you can compile the provided C implementation, run it to collect the expected output, run your assembly implementation to collect observed output, and then compare them — for example:

```
$ dcc cellular.c -o cellular
$ parameters="20 15 5"
$ echo $parameters|./cellular|tee c.out
0   ..........#.........
1   ###########.########
2   #..........#.......
3   #.###########.######
4   #.#..........#.....
5   #.#.###########.####
$ echo $parameters|1521 spim -f cellular.s|sed 1d|tee mips.out
```

0 ....#..... 1 ###########.######## 2 #..........#..... 3 #.###########.###### 4 #.#..........#..... 5 #.#.###########.#### `diff -s c.out mips.out` Files c.out and mips.out are identical

Try this for different values of the parameters.

Assumptions and Clarifications

Like all good programmers, you should make as few assumptions as possible.

Your submitted code must be hand-written MIPS assembly, which you yourself have written. You may not submit code in other languages. You may not submit compiled output.

You may not copy a solution from an online source (e.g., Github).

There will be a style penalty for assignments that do not use the stack to save and restore `$ra` (in main)

There will be a style penalty for assignments that do not follow these important MIPS calling conventions:

- function arguments should be passed in `$a0` through `$a3`
- `$s0`..`$s9` should be preserved across function calls: if a function changes these registers, it must restore the original value before returning

The above two style penalties apply only to assignments which score above CR (65+) on performance testing.

If you need clarification on what you can and cannot use or do for this assignment, ask in the class forum.

You are required to submit intermediate versions of your assignment. See below for details.

Change Log

Version 1.0
(2020-06-26 17:00:00)

- Initial release onto unsuspecting students.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest cellular
```

Assessment

Submission

When you are finished working on the assignment, you must submit your work by running `give`:

```
$ give cs1521 ass1_cellular cellular.s
```

You must run `give` before **Thursday July 16 21:59 2020** to obtain the marks for this assignment. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

You can run `give` multiple times. Only your last submission will be marked.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

Manual marking will be done by your tutor, who will mark for style and readability, as described in the **Assessment** section below. After your tutor has assessed your work, you can view your results here; The resulting mark will also be available via give's web interface.

Due Date

This assignment is tentatively due **Thursday July 16 21:59 2020**.

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 2%. For example, if an assignment worth 74% was submitted 10 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

Assessment Scheme

This assignment will contribute 15 marks to your final COMP1521 mark.

80% of the marks for assignment 1 will come from the performance of your code on a large series of tests.

20% of the marks for assignment 1 will come from hand marking. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, you will be assessed on how easy it is for a human to read and understand your program.

An indicative assessment scheme follows. The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

HD (85+)	beautiful documented code, which uses the stack with MIPS calling conventions, and implements spec perfectly
DN (75+)	very readable code, prints cells correctly if n_generations positive or negative, all rules correctly handled
CR (65+)	readable code, prints cells correctly if n_generations positive, rules mostly working
PS (55+)	prints initial world (first line) correctly and sometimes more correct lines
PS (50+)	good progress on assignment but not passing autotests
0%	knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP1521	submitting any other person's work; this includes joint work.
academic misconduct	submitting another person's work without their consent; paying another person to do work for you.

Intermediate Versions of Work

You are required to submit intermediate versions of your assignment.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below. It is fine if intermediate versions do not compile or otherwise fail submission tests. Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a Git repository and made available to you via a web interface at `https://gitlab.cse.unsw.edu.au/`*`z5555555`*`/20T2-comp1521-ass1_cellular` (replacing *z5555555* with your own zID). This will allow you to retrieve earlier versions of your code if needed.

Attribution of Work

This is an individual assignment.

The work you submit must be entirely your own work, apart from any exceptions explicitly included in the assignment specification above. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted.

You are only permitted to request help with the assignment in the course forum, help sessions, or from the teaching staff (the lecturer(s) and tutors) of COMP1521.

Do not provide or show your assignment work to any other person (including by posting it on the forum), apart from the teaching staff of COMP1521. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted, you may be penalized, even if that work was submitted without your knowledge or consent; this may apply even if your work is submitted by a third party unknown to you. You will not be penalized if your work is taken without your consent or knowledge.

Submissions that violate these conditions will be penalised. Penalties may include negative marks, automatic failure of the course, and possibly other academic discipline. We are also required to report acts of plagiarism or other student misconduct: if students involved hold scholarships, this may result in a loss of the scholarship. This may also result in the loss of a student visa.

Assignment submissions will be examined, both automatically and manually, for such submissions.

---

the School of Computer Science and Engineering

at the University of New South Wales, Sydney.

For all enquiries, please email the class account at cs1521@cse.unsw.edu.au

CRICOS Provider 00098G