Assignment 1
Scrolling Text

Objectives

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

Admin

Marks	9 (towards total course mark)
Group?	This assignment is completed **individually**
Due	by 11:59:59pm on Monday 21st October
Submit	`give dp1092 assign1 scroll.s`
Late Penalty	0.08 marks per hour late (approx 1.9 marks per day) off the ceiling (e.g. if you are 36 hours late, your maximum possible mark is 6.1)
Assessment	For a guide to style, use the code in the lectures and tute solutions, and the supplied code.

7 marks	for auto-testing on a range of input strings
1 marks	for commenting the code; you don't need a comment on every line, but roughly one comment on each block of MIPS instructions that corresponds to a C statement
1 marks	for readable code; sensible names, lining up the opcodes and the args consistently
-1 mark	for missing or trivial blog about planning, implementing, testing and debugging your code

If your assembly code has syntax errors (according to `spim`) or run-time errors on all test cases, your auto-testing mark is capped at 3/7, depending on an assessment by your tutor.

Background

A common sight in shops is a grid of LEDs where text scrolls across the grid, something like ...

The aim of this assignment is to complete a MIPS program that can scroll alphabetic text strings like the above video.

Setting Up

Create a private directory for doing the assignment, and put the assignment files in it by running the following command:

```
$ unzip /home/dp1092/public_html/19T3/assignments/assign1/assign1.zip
```

If you're working on this at home, download the ZIP file and create the files on your home machine. It's fine to work on your own machine but remember to *always* test your code on the CSE machines before submitting.

The above command will create the following files:

`Makefile`

A file to control compilation of `scroll.c`. It is not critical for the MIPS assembler part: it creates the executable C program to give you an exemplar, and can produce the `exe.s` file.

`scroll.c`

A complete solution, written in C. Your goal is to write a MIPS assembler program to copy the behaviour of this program.

`chars.h`

The array of big characters used in producing the scrolling text. This is `#include`'d in `scroll.c`.

`scroll.s`

A partly complete solution to the assignment, written in MIPS assembler.

`chars.s`

A MIPS version of the array of big characters used in producing the scrolling text. This file requires no modification.

Initially, it would be worth compiling the C program and running it on some examples to get a feel for its behaviour. The compiled C program, called `scroll`, expects a single command-line argument: the text string to be scrolled.

You can compile and run the C program (`scroll`) as follows:

```
$ make
gcc -g -Wall -Werror -std=c99    scroll.c   -o scroll
$ ./scroll
Usage: ./scroll String
$ ./scroll "It's fun"
Only letters and spaces are allowed in the string!
$ ./scroll abc def
... scrolls "abc"; it only uses the first commond-line argument
$ ./scroll "Hello there"
... scrolls "Hello there", like the above video
```

The Program

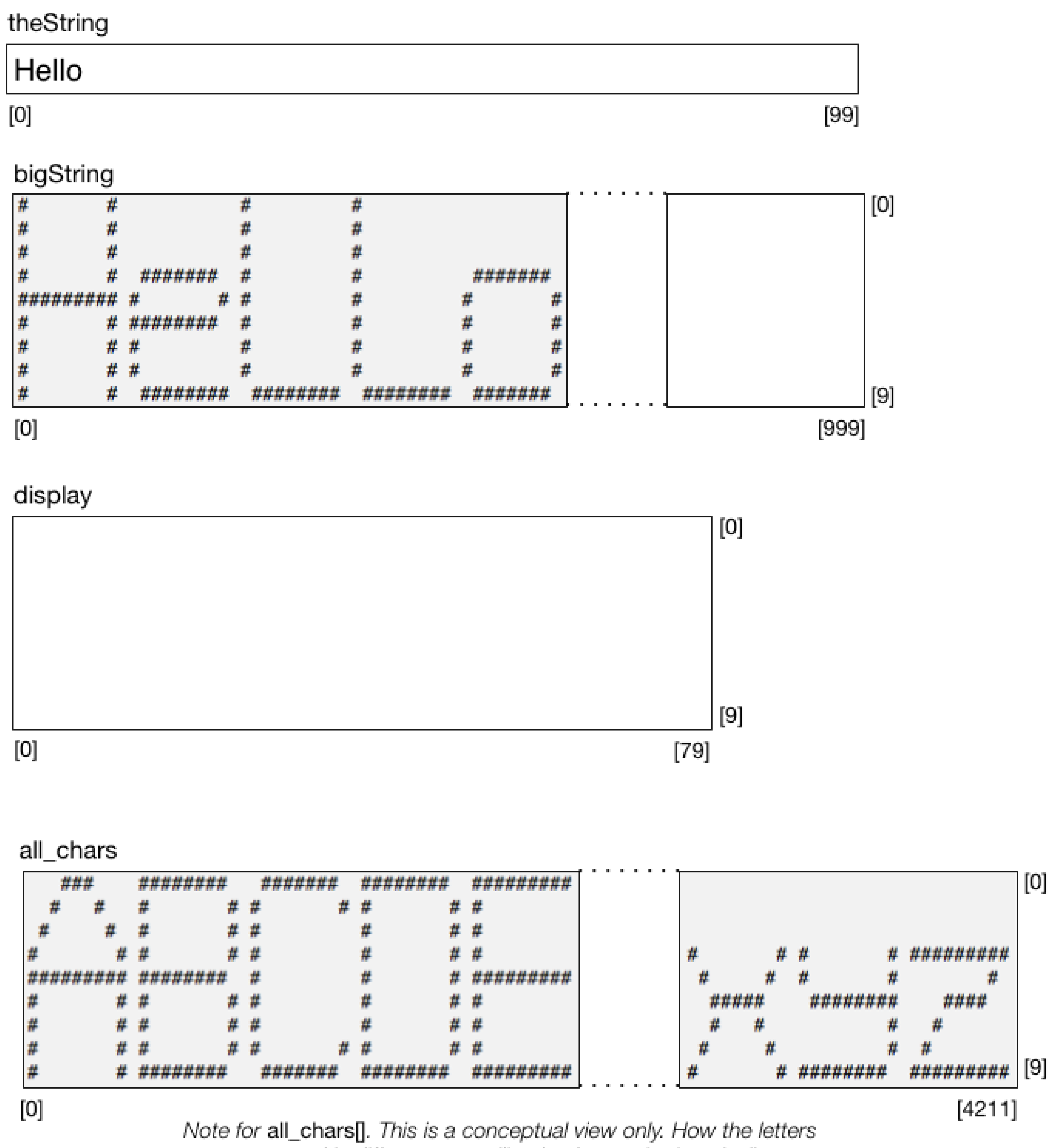
What the scrolling program should do, whether implemented in MIPS or C:

- check the command-line argument (< 100 chars, only letters and spaces)
- create a buffer containing big versions of the characters in `argv[1]`
- add part of the content of the big-char buffer into the display buffer, starting at `starting_pos`
- write the contents of the display buffer to standard output
- repeat, moving one column to the left each time, until the message scrolls of the left of the display

Both the C and the MIPS programs are structured the same, with a `main` function to handle the command-line arguments and then run the scrolling. The programs also have the same set of lower-level functions. In `scroll.c`, there are comments describing the purpose of each function and the code is hopefully clear enough that you can understand how each function works.

The diagram below shows the major data structures used by the programs:

- `theString[100]` holds a copy of the string from `argv[1]`
- `bigString[9][1000]` holds a copy of `theString` in big characters and with one column of space between adjacent big characters
- `display[9][80]` is where characters are placed before being written out to the screen
- `all_chars[52*9*9]` array containing representation of 'A'-'Z' and 'a'-'z' as big chars (not shown in the diagram; defined in the `chars.s` file)



Note for all_chars[]. *This is a conceptual view only. How the letters are stored is different; more like they're stacked vertically, and with no space between them.*

Exercise

The aim of this exercise is to complete the supplied MIPS program skeleton (in the file `scroll.s`) to behave exactly like the C program (in `scroll.c`). You should not change the `chars.s` file; treat its contents as a read-only data structure.

In `scroll.s` each function has comments to:

- indicate which registers the function uses
- indicate which registers the function overwrites (clobbers)
- give a mapping between local variables in the C code and registers in MIPS

Note that these are suggestions only; you can use whatever registers you like, provided that you save and restore any `$s?` registers that you overwrite in the function code. And, of course, provided that the code behaves the same as the C code.

To save you some time, we have included function prologues and epilogues in some functions. These save and restore registers `$fp`, `$ra`, and any `$s?` registers that the function happens to use, and also maintain the stack. You can use these as templates for how to implement the prologue and epilogue in the functions that do not provide them.

Some of the functions from `scroll.s` are already implemented, but others require you to write MIPS assembler for them. Here's a rundown of the functions in `scroll.s` and their status:

`main`	Partly complete, including the epilogue and prologue, and the command-line argument checking.
`setUpDisplay`	Function prologue and epilogue ok. ToDo: function body.
`showDisplay`	Function prologue and epilogue ok. ToDo: function body.
`delay`	Already complete, but you can tweak the numbers if you want, to speed up or slow down the animation. As supplied, the delay function in scroll.s will be way too slow. The numbers work fine for compiled C, but for interpreted MIPS assembler (which is 1000 times slower). Tweak the loop bounds in delay until you get decent scrolling speed. While you're debugging (e.g. in qtspim) the delay is not helpful. The simplest fix would be to comment out the call to delay in the main program. When we test it, we will comment out the calls to the delay function.

`isUpper`	ToDo: function prologue and epilogue, and function body.
`isLower`	Already complete (and makes isUpper very easy).

Running the program

Note that `scroll.s` is not stand-alone MIPS program; it requires access to the `all_chars[]` array in the file `chars.s`. In order to run the program (either via `spim` or `qtspim`), you'll need to combine the two files. Here's an example of you might run the program using `spim`:

```
$ cat chars.s scroll.s > exe.s
$ spim -file exe.s
... program executes ...
```

The file `exe.s` *is* a complete program that can be loaded into `qtspim` as well. You will need to do the `cat` step each time you change the `scroll.s` file and want to test it. The `Makefile` knows how to create this file too.

Challenges

(Worth kudos, but no marks)

- Make the scroll repeat. If you wait until the whole message scrolls off the left hand end, this is easy. Make it do a "continuous scroll", where the message starts repeating from the right before when the end of the right-hand end of the message has moved 10 columns from the right-hand end of the display.
- Change the colours of the '#' characters as they scroll. Make the colour change as aesthetically pleasing as possible.