COMP1521 Week 4

2D arrays and MIPS functions!

Assignment has been released!

Notes on style:

- I would recommend writing your code using 'hard' tabs (don't insert spaces) with a width of 8.
  - Should be done automatically if you have the ‘Mipsy Editor Features’ extension installed
- Assembly style guide on course homepage is generally a good idea.
- Don't forget to fill out all of that documentation above your functions! Also remember to fill out your header comment :)
- Use descriptive label names, and try and follow the conventions already used in the file.
  - i.e. "main__row_init_loop" is a much better label than something like "loop1".

What is "frame", "uses", and "clobbers" (in function documentation)? What about "locals"?

- Frame = registers placed on the stack and restored before the function returns.
- Uses = registers used by the function.
- Clobbers = registers whose values are changed by the function. Hence, clobbers is generally equal to (uses - frame).
- Also, 'locals' refers to local variables in the C code.
- For further clarification, you might find https://jashankj.space/notes/cse-comp1521-better-assembly/ useful.

MIPS conventions

- Always put function arguments in registers $a0, $a1, etc…
- Return value should always be in $v0.
- You must *always* assume that anything you put in a register that isn’t a $s is ‘destroyed’ or ‘wiped’ as soon as you call a function (technically this also applies to $ra, $sp, and $fp too…).
- Call begin/end in your prologue/epilogue to prevent anything strange happening…
- You must save (push) $ra in the prologue in any function that calls other functions.
- You must save (push) $s registers in the prologue if they are used in the function.
- In the epilogue, pop whatever you push in reverse order that you pushed.

Memory storage and alignment (week 3 revision)

- What does .word 42 do? How about .space 8?
  - .word 42 allocates a ‘word’ (4 bytes) of space and stores the value ‘42’ in it
  - .space 8 allocated 8 bytes of uninitialised memory
- What does .align n do?
  - Ensures that the next field after the .align will be aligned to a memory address that is divisible by 2^n.
- Why do we need it?
  - MIPS can only read values if they are ‘aligned’ (i.e. the memory address is divisible by the size of the data type being read)

Onto the tute questions...