CS2261 Media Device Architecture

* Started off with a demo of a rectangle bouncing around on the screen
  + Size of rectangle can be enlarged or reduced using the keyboard
  + No flicker noticeable at first, because the writing to memory was happening during the VBlank
* Problem with the rectangle was that, as the rectangle got bigger, it began to flicker
  + This occurred due to too much being written to memory during the VBlank
  + There’s not really a fix for the short time in Mode 3
    - What you can write to memory during VBlank must be short and efficient
  + It’ll be different in different modes
* Buttons:
  + There are 10 input buttons
    - As the programmer, you have full control over the buttons and their funcitons, so you can make the START button be jumping, or have the LEFT SHOULDER button start the game
    - Bad practice
  + There is a register that keeps track of what buttons are pushed
  + Button register located @0x4000130
    - It tells you what button is pushed
    - Only one will be pushed at a time
  + There are 16 bits in the register, but only 10 are used at a time

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15-10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Left | Right | Down | Up | Left Shoulder | Right Shoulder | Start | Select | A | B |
|  | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

* Interestingly, instead of having a 1 in the place of a button being pushed, there is a 0
* Example, left button pushed:
  + 1111101111111111
* You’ll never see multiple 0s
* It’s helpful to create macros for the buttons
* Example:
  + #define BUTTON\_LEFT 512
* You can check the number in the button registry by creating a macro for the button register and checking it against all the values for pushed buttons to see which is pushed
  + Ideal way to do this is to:
    - Use the ~ (complement) to flip the value from the button register to one more easily readable, one that corresponds to a smaller value
      * ~1111101111111111 = 0000010000000000
      * =512
    - Use the & AND bitwise operator to check which value from the register matches one of the buttons values
    - If they match, you know which value was pushed
* We use the volatile keyword in the Button Register’s macro because if a button is pushed, the value at the register will change but the compiler won’t notice it in the main program, so by making it volatile you force the compiler to recompile the macro each time