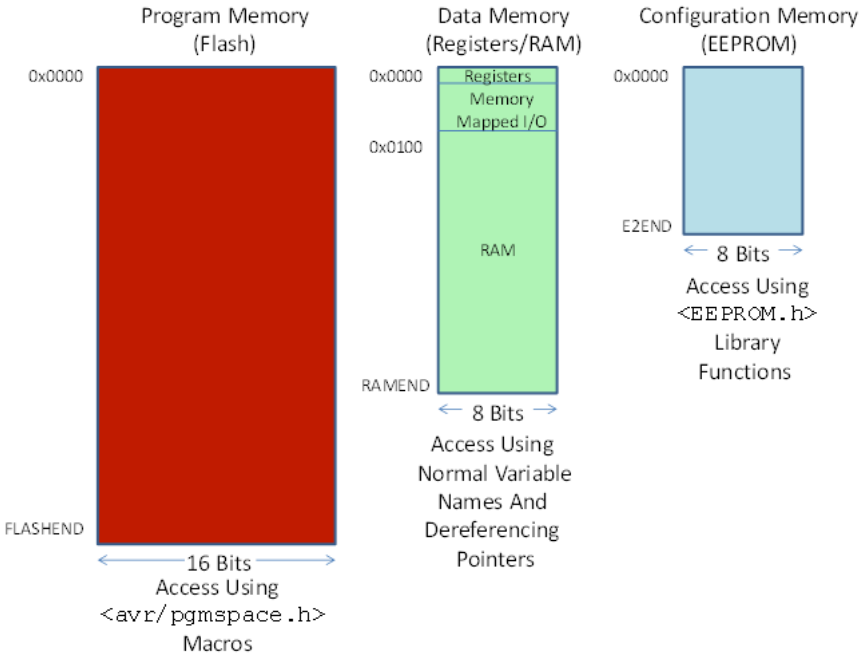
CS 341 – Lab 2
Memory Exploration

The goal in this lab will be to explore the Harvard architecture and its memory spaces though we will focus primarily on the Data memory. The starter code can be found on blackboard under Lab 2. The code is mostly written for you already, you should uncomment it bit by bit and observe what happens.

Part 1

The processor used in the Arduino UNO (the ATMEGA328) is built along the lines of the Harvard architecture. In this architecture, the processor stores program code and program data/stack in separate memory spaces.



There are three memory spaces in the Arduino UNO processor architecture. See the Figure above. Note that addresses for each of the three memory spaces start with 0x0000 and go to a symbolic constant as the end address - FLASHEND, RAMEND, or E2END. These constants are defined in the IDE compiler and you will determine their hex values below.

The Flash (program code) and EEPROM (configuration data) are non-volatile meaning that their contents are preserved across power cycles. The RAM is volatile so its contents are lost when the power is turned off. The contents of RAM are initialized during the power on sequence in the program code. Note that the 32 general purpose registers and the I/O device registers are "memory mapped" into the data memory address space from 0x0000 to 0x00ff. The program memory is organized as 16 bit words while the data memory and EEPROM are organized as bytes.

1. Plug in an Arduino
2. Copy the starter code into a new sketch in Arduino IDE
3. Find the section in the code labeled "Part 1" and uncomment it
4. Click run, and open the serial monitor
5. **Record the 3 constants in the results section of your lab report**

Part 2

Now that we know the size of our memory spaces, lets create some arrays and see where they end up.

1. I have already created 6 arrays, 3 as global variables and 3 in setup
2. Take a careful look at the arrays, and how they are initialized
3. Uncomment the section in the code labeled "Part 2"
4. Clear the serial monitor, and click run
5. Look at the locations of the arrays and try to piece together how the compiler allocates memory. Where does the heap, stack, and initialized data end up?

Part 3

After seeing where our arrays ended up, we're going to print out sections of the RAM to confirm our guesses at the end of part 2.

1. I have already created the functions displayRAM and displayAllRAM to help you out. Call them like this:

displayRAM((char *) 0x100, (char *) 0x200, false);
displayAllRAM(2000, false); //displays memory in 0x100 blocks with delays

The output will look something like this.

```
200: . . . . A r r a y   4 . . . . .
210: . . . . . . . . . . . . . . . .
220: . . . . . . . . . . . . . . . .
230: . . . . . . . . . . . . . . . .
240: . . . . . . . . . . . . . . . .
250: . . . . . . . . . . . . . . . .
260: . . . . . . . . . . . . . . . .
270: . . . . . . . . . . . . . . . .
280: . . . . . . . . . . . . . . . .
290: . . . . . . . . . . . . . . . .
2A0: . . . . . . . . . . . . . . . .
2B0: . . . . . . . . . . . . . . . .
2C0: . . . . . . . . . . . . . . . .
2D0: . . . . . . . . . . . . . . . .
2E0: . . . . . . . . . . . . . . . .
2F0: . . . . . . . . . . . . . . . .
```

The first row corresponds to memory addresses 0x200 through 0x20F. Thus, the memory address 0x2A0 is currently storing the integer 4.

2. **In your lab report answer the following question:**

What memory addresses were used by the stack, heap, and initialized data? Give rough estimates such as "the stack starts around 0x300 and extends towards 0x00."

Part 4 (Optional):

Uncomment part 4. (re-comment part 3 for your own sake). I have put a long variable into memory, a =0x12345678 in hex or 305419896 base 10. Look at it's location, both shown in hex digits and ascii values.

In your report, tell me:

- if Arduino is a little or big endian machine?

- Can you spot long e? Why are e and a in the Ram stack and not the heap?

- Is the location of array 3 the same as before you uncommented part 4? If not, is it higher or lower? What did adding another item on the stack lower in the program do to its location? (Note, the stack is assembled during compilation, only in assembly and not c can we actually push items onto the top of the stack)