**What is Project 2?**
- Write the part of gmipc that actually executes instructions
- A simulation single-cycle implementation of the MIPS subset
- Means an you only operate on one instruction at a time
  - Decode the instruction
  - Fetch the register values
  - Compute and store new register values or memory addresses
  - Load values from memory or store values to memory
- You’re given some skeleton code and you basically have to fill it in and “simulate” the effects of all the instructions on the processor state – what is the state?
- You are given a pointer to memory and a pointer to the register file, but you should not modify memory directly – instead you are given two interface functions to memory, one for storing and one for loading.
  - Keep in mind that you have to do a “load” to simulate the effects of a store properly – why?

**Structure of a C Program**
- Kernighan-Ritchie “The C Programming Language” is a good investment
- Can compile your own C programs by typing: gcc filename.c –o outputfilename

```c
#include <stdio.h>   //this is a comment, stdio lets you do printf and stuff like that

/*this is a
   multiline comment*/

int foo(int, float); //function **declarations**
char x = 3;  //variable **declarations** and optional initializations
             //x can be accessed and modified from anywhere it’s “**global**”

int main(void){
   …   //a bunch of statements

   return 0;  //“ok” exit status
}

int foo(int a, float b){ //**definition** of foo
   …   //a bunch of declarations and statements – all declarations “**local**”
   return (blah);   //at least one of these somewhere because return int
}
```
C Types

- Four basic types
  - int, char, float, double
  - actual size of the types is architecture dependent
  - come in different “flavors”: e.g. long int, unsigned char, unsigned int
  - different flavors don’t amount to different bit patterns in general, just different mathematical interpretation (e.g. comparison, shifting)
  - the basic types are implied to be “signed”

- Arrays/Pointers
  - You declare an array by saying: int x[5]
  - The name x refers to the base address of the array
  - You access elements of the array by giving the array name and an offset into the array, indexes start at zero – isn’t CS fun! For example: a = x[3];
  - You can declare a pointer explicitly, and it initially points to nothing (NULL) as follows: int *z; //declares a pointer to an integer and names it z.
  - Pointers can be used to indirectly change the contents of a memory location, but to do so you must dereference the pointer to get at the data it points to.
  - For example *x = 5 //change the value to which x points, to 5
  - You can get the address of a variable by prefixing it with an ampersand.
  - An equivalent statement to j = r[3], is j = *(r+3)

- You can also type cast values to explicitly interpret their values a certain way

- What do the following things do?

```c
int *A, *B;
int C=1, D=2;
A = &C;
B = &D;
//ints are 4 bytes long, shorts are 2 bytes long

printf("A+B=%d\n", A+B);
printf("A+B=%d\n", *A+*B);
printf("A=%d\n", (short)(*A));
printf("A=%d\n", *((short *) A));
printf("A=%d\n", *((short *)A+1));
```

C constructs/statements

Conditional
If(condition){
  …do some thing…
} else if (some other condition){
  …do something else…
} else {
  …if nothing else do this…
}
Selection

- A convenient way of “decoding”

```c
switch(variablename){
    case value1: // if variablename == value1
        ...some statements...
        break;
    case value2: // if variablename == value2
        ...some other statements...
        break;
    ...default: // if the variablename’s value is not listed
        ...do some other values...
}
```

Loops

- The usual loop constructs, for(i=0;i<bound;i++){...}, do{...}while(condition), while(condition){...}
- Probably don’t need to write any loops for your project

Bit Manipulation

- How do you set the \(i^{th}\) bit of \(x\)? \(x = x | (1<<i)\);
- How do you clear the \(i^{th}\) bit of \(x\)? \(x = x & ~(1<<i)\);
- Figure out whether \(i^{th}\) bit of \(x\) is set? \((x&(1<<i))\)\
- What does ((signed) \((x<<16))>>16\) do?