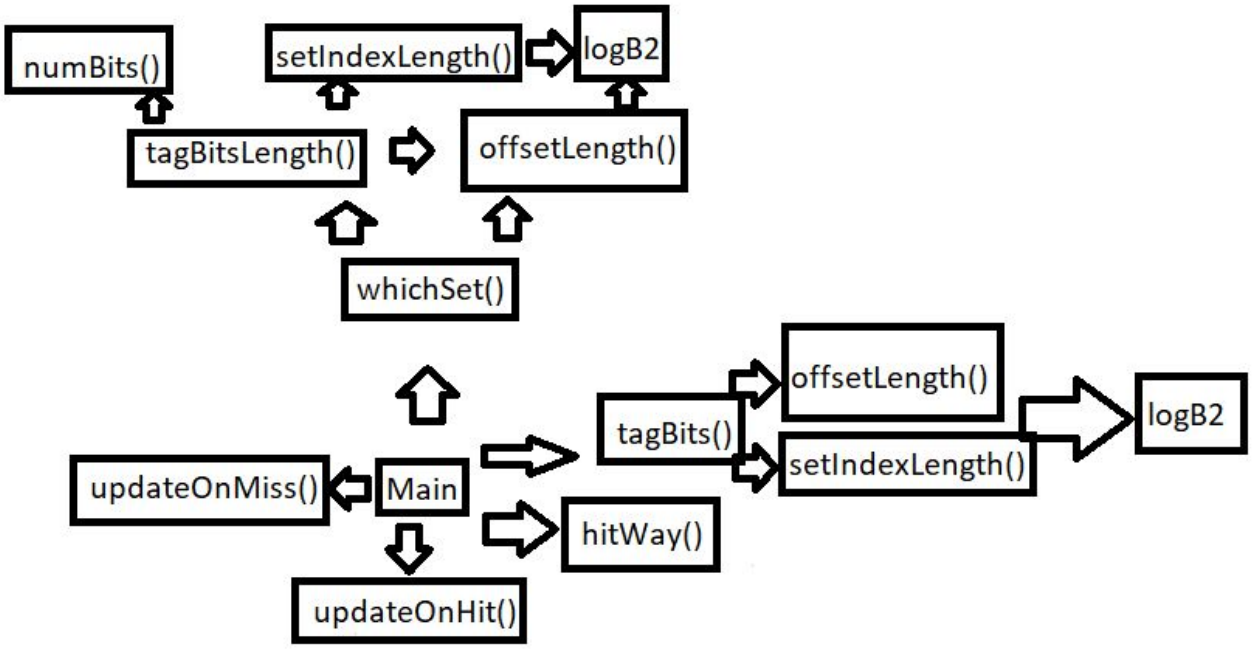
ECE 353 Lab 1 Summary Sheet Student Names:
(1) Alex Donadio
(2) Sam Fick
(3) Dhimiter Shosho

1. List all functions in your program and draw its call graph.

- **int whichSet()**
- **int setIndexLength()**
- **int offsetLength()**
- **int tagBits()**
- **int hitWay(...)**
- **void updateOnHit(...)**
- **void updateOnMiss(...)**
- **int logB2(...)**
- **int tagBitsLength(...)**
- **int numBits(...)**



2. List all assertions used in each function

Generally, we programmed each function to work with the upper bound of 32-bit addresses.

- **int whichSet()** outputs the cache set in which the address falls.
- **int setIndexLength()** outputs the number of bits in the set index field of address.
- **int offsetLength()** outputs the number of bits in the line offset field of address.
- **int tagBits()** outputs the tag bits associated with the address.
- **int hitWay(...)** if there is a hit, this outputs the cache way in which the accessed line can be found; it returns −1 if there is a cache miss.
- **void updateOnHit(...)** updates the tagArray and lruArray upon a hit. This function is only called on a cache hit.
- **void updateOnMiss(...)** updates the tagArray and lruArray upon a miss. This function is only called on a cache miss.
- **int logB2(...)** is a helper method used for getting index bits, offset bits, etc.
- **int tagBitsLength(...)** is a helper method used for getting number of tag bits.
- **int numBits(...)** is a helper method used for counting total number of bits in given address.

3. Did your code pass all your tests? Mention what these tests were

We noticed that we had to create our own log base 2 function in order to find the offset and index bits for each address. Our logB2() function was tested thoroughly by feeding it numbers that were powers of two and checked to see if it successfully gave us correct values via printf.

We noticed that we had to create a function that would tell us the total number of bits in each given address. We tested our numBits() function by feeding in addresses of different sizes and checked to see if resulted in the correct number of bits.

We tested our hitWay() function by feeding it a sample of about ten to fifteen addresses and checked to see if the result was a -1 or the actual block using print line debugging.

We tested our setIndexLength() and offsetLength() changing our values for C, K, and L and printing the results. We changed our values for C, K, and L to check to see if this worked properly for different values.

With out numBits(), setIndexLength(), and offsetLength() working, we tested our tabBitsLength() by feeding in addresses and seeing if we got the current number of bits for the tag.

We testing our tagBits() function similarly, feeding in a sample of twenty different addresses and checking to see if this function resulted in the correct tag bits for each address.

We tested our whichSet() function by feeding in a sample of twenty different addresses and checking to see if it output the correct index bits.

We tested our updateOnHit() and updateOnMiss() functions by using a sample of about fifteen addresses as we know the rest of the functions were working fine. We tested addresses that resulted in all hits, all misses, and a mixture of both.

After successfully completing each function, we tested our cache simulator using the sample trace given to us on the class website. This had roughly one million traces that went through the cache simulator. The runtime for this was about 38 to 39 seconds with parameters 8 16 512. Which resulted in a miss rate of about .025 or 2.5%. When we increased the block length L, we noticed the miss rate would decrease as it should as we are increasing the set associativity. When we reduced L to 1 we would notice that the miss rate would increase to about 12% which is as expected as we reduced the set associativity. After testing our program will over one million traces, we did not run into any major memory issues and the output was as expected. We are confident that our cache simulator works effectively.