Project 4 continued building off of what we learned this semester by creating a cache simulator. It was similar to the single cycle simulator we made earlier this semester, with the additional code to implement a cache system that can be anywhere from direct mapped to fully associative. Our cache system is flexible because it handles block size, number of sets and associativity. With these dimensions it creates the correct cache to match the given parameters.

It works very similarly to our other simulators by taking in a file of machine code, however it also takes in block size, number of sets, and associativity. First it reads in the machine code, next it creates a cache according to the given dimensions, and then sets the block values of the created cache to their default values. This means that the valid bit is 0, meaning it is ready to accept input. The dirty bit is 0, tag is 0, lru is set to associativity - 1. It begins operations like our other simulators, the only difference being that the simulator never reaches out to memory instead it will always reach out to cache for memory. Simulator only uses memory when fetching an instruction, using load word or store words. These all require memory as the instruction comes from memory, load words reach out to memory and grab a value and store words will change a value in memory. Therefore in our cache simulator whenever we attempt to do one of these things they will have to go through cache to reach memory. Looking up the cache value can be done by going to the correct line of the cache and then comparing tags of the set. First we have to determine if the memory is already in the cache, in that case it is a hit and we just grab the value from the cache. Otherwise we have to find a valid spot to place the memory block in the cache. There are two options: either there is an open spot or we must boot one block from the cache. If there is an open spot then the block is placed in the open spot or the least recently used block is kicked and the new block replaces it. We know which is the least recently used block as we keep track of which block is the least recently used block. Now it reaches out to memory and replaces the block and then uses the cache block to access memory.

When the cache deals with load words and instruction fetches, it is just allowing the simulator to read memory but, in the case of store word it changes a value in memory by first changing the cache value of that memory. This then changes that block and makes it a dirty bit. A dirty bit requires the block to change a value in memory to match the updated value in the cache. It does this when a dirty bit is being replaced or after a halt has been called and all dirty bits must update memory values.

Some things we struggled with was creating a flexible cache and test cases. The flexible cache can have direct mapping and be fully associative at the same time. Once we realized it was just a 2d array with the x axis being sets it was much easier to create. When it came to creating test cases, it was a lot harder to come up with them because testing the cache is much more complicated than testing if instructions are interpreted properly. We had to think of where we had to place the load word and store words in order to test each function of our cache.