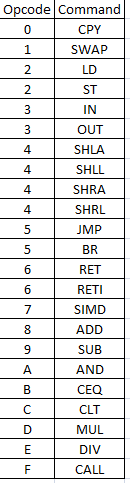
**Holden Sandlar**

**Advanced Computer Architecture – 0301-810**

For my CPU I Plan to implement the CPU simulator using C++. Over to the right is a table describing how I am mapping each given command to an opcode. The memory elements (registers, data memory, program memory, stack pointer, program counter, I/O memory space) will be implemented as character arrays. The character type in C and C++ are uniform sized bytes.



Each stage of the MIPS style pipeline will have both input and output registers and at the end of each machine cycle the input registers will be sent to the output registers after the interrupts have been serviced.

Further ideas about implementation have been found on this website: <http://fms.komkon.org/EMUL8/HOWTO.html>

Other languages besides C++ were considered prior to the selection of C++ however, due to my strong background using C and C++ versus the learning curve I would be facing in languages such as VHDL, Ruby, Perl, Python, etc. it was easy to make the decision to use C++. Also, there are a very strong set of standard template classes in C++ which makes dealing with things such as mapping dictionaries, dealing with strings, byte arrays, file i/o, etc. very simple.

At this point, I have implemented a basic shell type interface which provides information about the registers, program counter, stack pointer, and data memory counter (although I have questions about that). I have also created a register file and class to interact with the register file.

An “opcode file” to create the table on the right as well as a “cycles per instruction file” to be able to easily vary how many cycles each instruction uses in the pipeline have been implemented. The current development environment being used is the recommended Bloodshed Dev-C++.

A basic assembler class has been written to convert a specified “code.asm” file into a “machine code” file. After conversion, the machine code is read in to the program and data memory byte arrays for simulation.

Basic overview of commands planning to be implemented: help – list of commands, info – prints state of system, ni – execute next machine cycle, c – execute till end of program, c <int> - execute desired number of machine cycles