Group Members: Adam Eichelkraut & Jared Brown

Our group began collaboration on the project by meeting to discuss our initial plan. Once we had an idea of how our simulation machines were going to be developed, it was easier to figure out who could start working on what. We used GitHub in order to help collaborate and share code while developing the project. There were also a few more meeting over the past week in order to update our plan and wrap up the project.

Contributions:

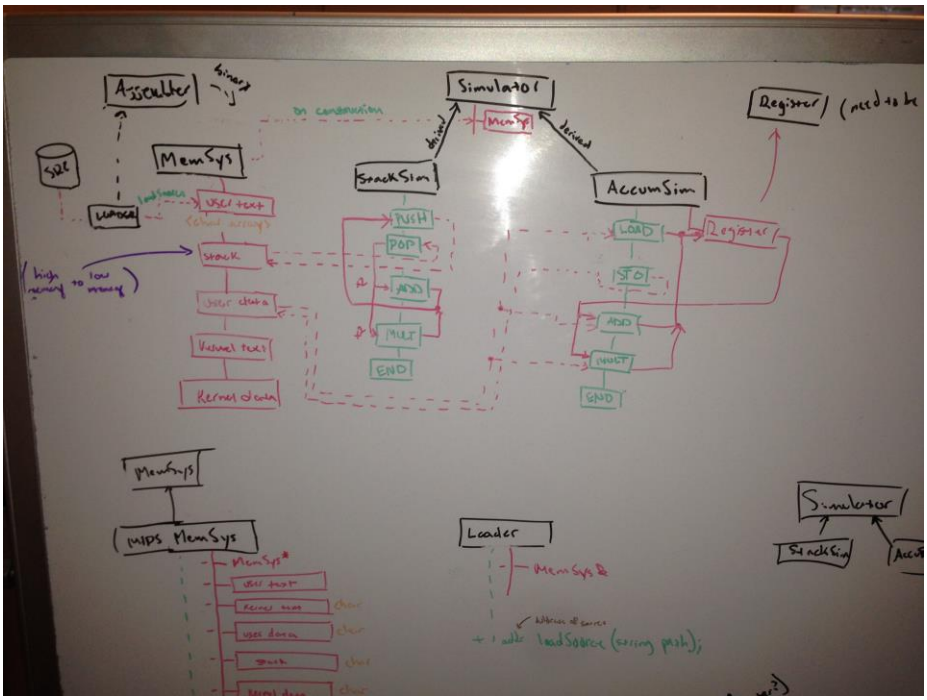
Adam:

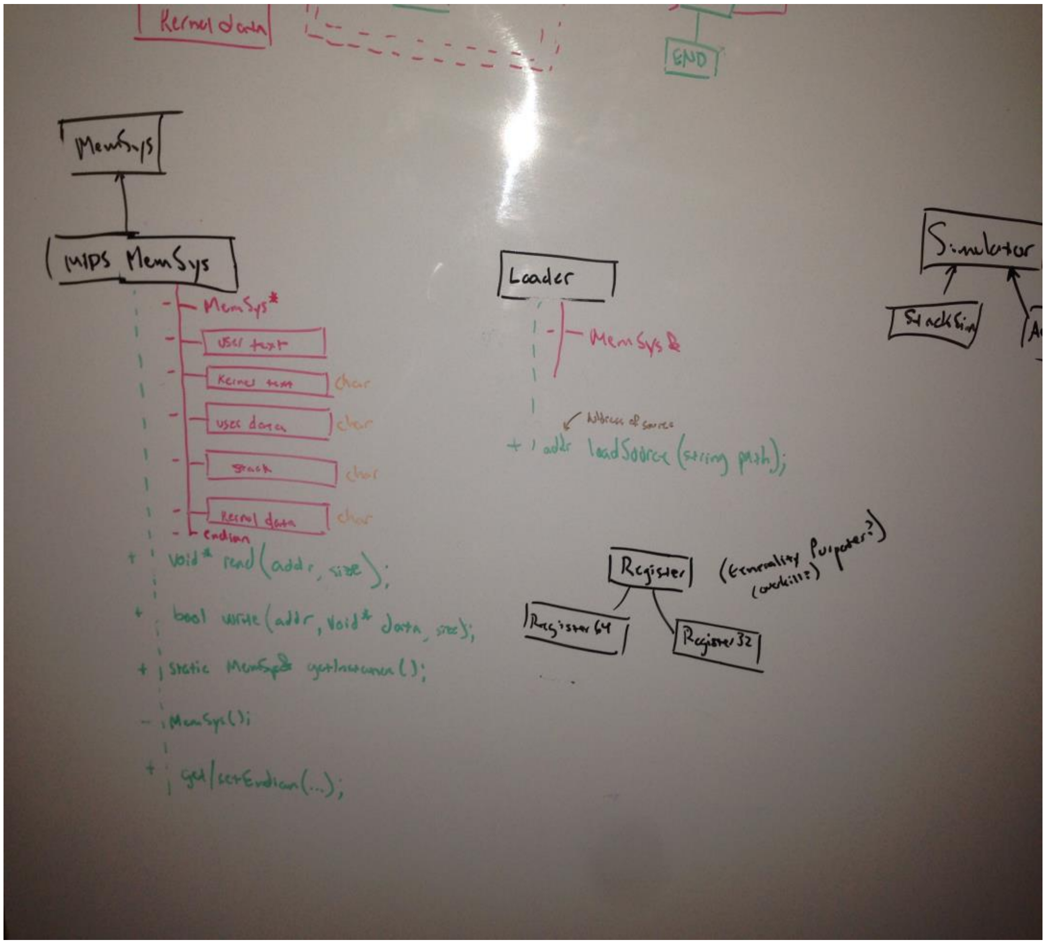
- Designed Memory System and Loader
- Set up source code tree
- Created makefiles
- Corrected quadratic_eval.s source code and converted for Accumulator
- Set up Simulator interface and drivers
- Bug fixes
- Wrote README file

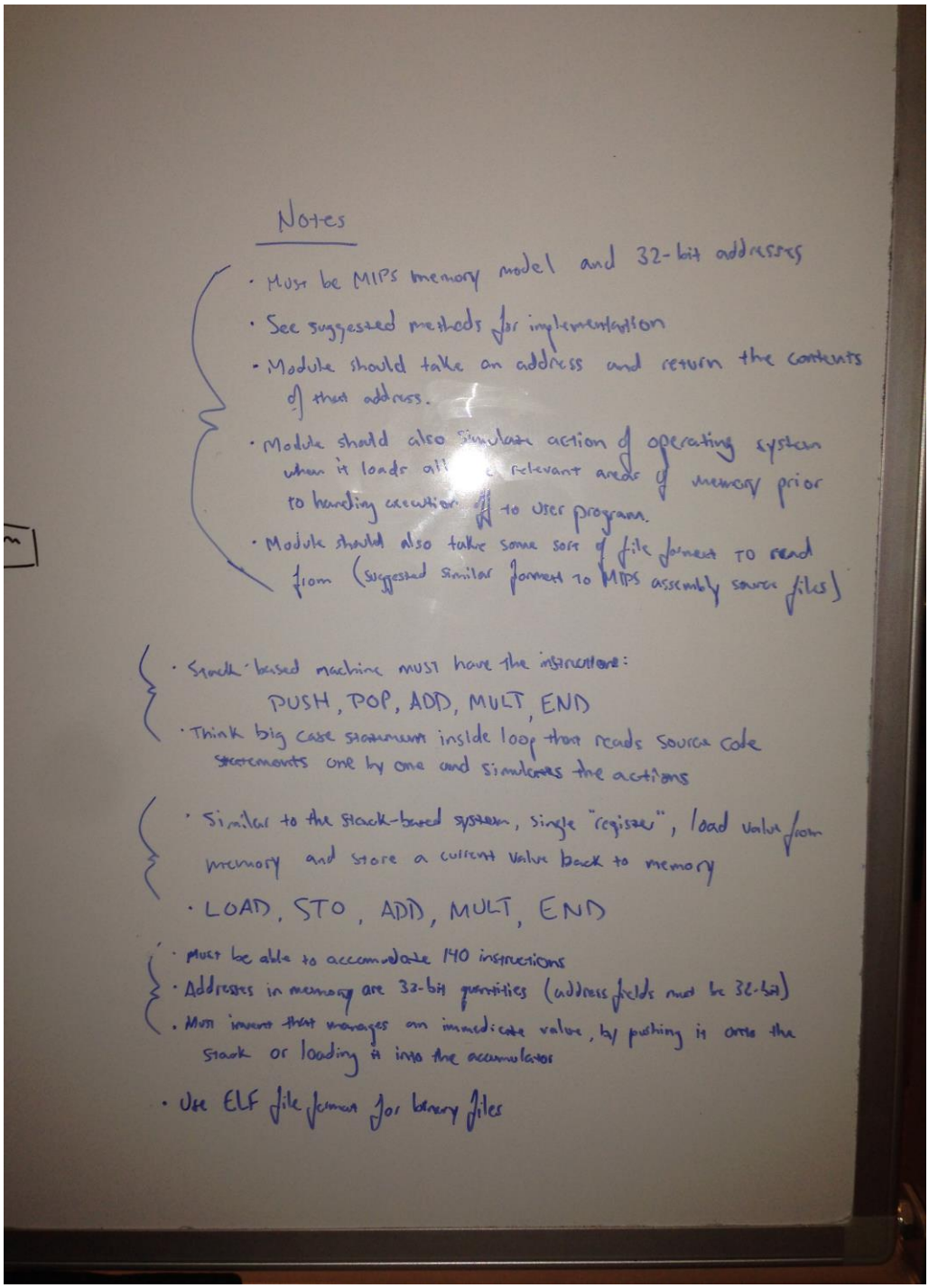
Jared:

- Stack and Accumulator Simulator Implementation
- Converted source code for Stack
- Wrote encoding.pdf (for questions in Section 4)
- Wrote collaboration.pdf

Notes created during meetings:




Kernel data
END
MemSys
MIPS MemSys
- MemSys*
- user text
- Kernel text char
- user data char
- stack char
- Kernel data char
- Endian
+ void* read(addr, size);
+ bool write(addr, void* data, size);
+ static MemSys& getInstance();
- MemSys();
+ get/setEndian(...);
Loader
- MemSys&
+ addr loadSource(string path);
Simulator
Register
Register64
Register32


Notes
• Must be MIPS memory model and 32-bit addresses
• See suggested methods for implementation
• Module should take an address and return the contents of that address.
• Module should also simulate action of operating system when it loads all relevant areas of memory prior to handing execution off to user program.
• Module should also take some sort of file format to read from (suggested similar format to MIPS assembly source files)
• Stack-based machine must have the instructions:
PUSH, POP, ADD, MULT, END
• Think big case statement inside loop that reads source code statements one by one and simulates the actions
• Similar to the stack-based system, single "register", load value from memory and store a current value back to memory
• LOAD, STO, ADD, MULT, END
• Must be able to accommodate 140 instructions
• Addresses in memory are 32-bit quantities (address fields must be 32-bit)
• Must invent that manages an immediate value, by pushing it onto the stack or loading it into the accumulator
• Use ELF file format for binary files