**Journal**

**Meeting 1: Tuesday, 10 January 2017**

We began the meeting by discussing the features we most wanted to include in our processor. We hoped this would guide our design throughout the meeting.

We decided to build a processor that is primarily based on a combination of load-store and accumulator designs. One specific register will hold the value of the last arithmetic or logical computation. This register will be read only for users to copy and read the value of the last computation. We will also allow users a small set of registers to save values beyond a computation and save values over procedure calls.

Our processor will have 18 registers in total, 15 of which are available to the user in some capacity (read/write). There are 8 general purpose registers, and 8 special registers. We decided on a small, specific instruction set made up of one major type, an arithmetic/logical type that returns all results to the special computation register, as well as other instructions of varying sizes. We believe that varying sizes will help keep our programs small in size and more efficient.

For procedure calls, we thought it would be interesting to make arguments and return values memory addresses only. We acknowledge the inefficiencies of that design but felt that the value of having direct access to the memory address of the arguments and return values was a valuable asset to our design.

Work log:

(majorly a group effort worked on during the 3 hour period):

Design and description of registers, instruction type and format, procedure call conventions (Discussion)

Trinity - Journal Notes

**Meeting 2: Wednesday, 11 January 2017**

We are doing a lot of redesigning based on feedback and more direction with the project. We decided to standardize the size of instructions to 16 bits. We now have two types of instructions, a C-type for register to register computations and an I-type for other instructions that require a register and immediate values such as load/store, branch, jump. The I-type include all of our instructions with previously varying sizes. By establishing a standard size and design for instructions, we are able to greatly simplify our design and get a better direction on designing instructions.

We decided to scrap the memory address-only idea for arguments and return values as it would be a cumbersome design at our current state of progress.

By the end of the day, we’ve ended up with 20 registers, deciding to split our general purpose registers between saved and temporary registers. Our number of instructions has grown to include a number of pseudoinstructions as Shaun began coding the programs.

Work log:

*Before Meeting:*

Khaled and Shaun - Register Descriptions (write up)

*Over 7 hour group time (1800-0100):*

Logan - Assembly Syntax and Semantics, addressing modes, grammar and formatting

Trinity - Procedure Call Conventions (write up), Journal, Machine language instruction format type and semantics, rules for translating assembly to machine language

Khaled - Assembly fragments

Shaun (assisted by Trinity) - Euclid’s algorithm and relPrime (Assembly)

Group - moderate redesign of instruction format, registers, and instructions

*After meeting (2 hours):*

Shaun (assisted by Khaled) - Euclid’s algo/relPrime (Machine Language)

**Meeting 3: Thursday, 12 January 2017**

We decided to reformat the jal and j instructions to be their own type. This allowed for a larger jump block for us,

We plan on featuring a assembler, compile, linker, and exception handler at the very least in our processor. One of the registers dedicated to exception handling will be $ex, the cause register which will look like this:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |