**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, L-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. We also cut down our registers to make our register file size 16.

We created a few new instructions and pseudoinstructions and added these to the design documentation. Most important of these is probably the addition of *syscall* which will be our I/O instruction.

Other than these decisions, we reformatted and reworded our design document according to the feedback provided to us and attempted to make things more clear.

Work log:

*Over 1 hour meeting:*

Logan – general editing over entire doc

Trinity – syscall, editing instruction descriptions

Shaun – fixed problems with Machine Language

Group – redued register file and redesigned jal/j instructions.

**Meeting 4: Friday, 13 January 2017**

We polished off the work from yesterday including updating our common operations table, reviewing the changes made yesterday, and finishing any work left incomplete after the meeting.

Began going over Milestone 2 requirements and planning for the next few days.

Work log:

*Over 1 hour meeting:*

Shaun – Machine code and common operations table

Trinity – finished syscall, journal

Group – Formatting and decisions about new instructions

**Meeting 5: Sunday, 15 January 2017**

We decided to use multi-cycle control as we thought this would make our performance faster and simplify our RTL design. We found that we could group all C-type and L-type instructions into individual RTL, and grouped branches, load/store word, and load lower/upper immediate instructions as well.

Planned meeting times:

Sunday: 1400-1700 (may vary)

Mon-Fri: 2030-2130, class time and lab time when cancelled

Work log:

*Over 3 hour meeting:*

Shaun – RTL Table, Component test descriptions

Logan – RTL Table, Component list,

Trinity – RTL Table, journal, formatting, editing

Khaled – Control Descriptions, input/output/control signals

Group – Discussion on major decisions

**Meeting 6: Monday, 16 January 2017**

No major decisions.

Worked on milestone 3.

Work log:

*Over 1 hour meeting:*

Trinity - Journal

Khaled – worked on Milestone 2

Shaun – Reworked Machine Language translations to conform to changes

Logan/Trinity (w/ assistance from Khaled & Shaun) – Datapath Design

**Meeting 7: Tuesday, 17 January 2017**

No major decisions.

Polished milestone 2.

Work log:

*Over 1.5 hour meeting:*

Logan – Inputs and outputs, RTL modifications

Logan and Shaun – Test descriptions

Trinity – RTL/control description/input & output modifications, additional components, journal

Logan, Shaun, & Trinity – General editing, formatting and corrections

Khaled – sick (not in attendance)

*After meeting (~1 hour):*

Trinity – Reviewed/modified RTL and Verilog to adhere to test descriptions and page numbers!!

**Meeting 8: Thursday, 19 January 2017**

Worked on Lab 7 & 8

Work log:

*Over 1 hour meeting:*

Shaun and Khaled – Lab 7

Logan and Trinity – Lab 8

**Meeting 9: Friday, 20 January 2017**

Reviewed feedback for milestone 2 and made some major changes to RTL and organization of design document.

Work log:

*Over 1 hour meeting:*

Khaled – broke up RTL tables, edited RTL, formatting (tables, color)

Logan and Trinity – inputs/outputs table, table formatting, review of changes

Trinity – Condensed/reordered machine language and assembly sections, journal

Shaun – Reviewed tests, reviewed changes, small editing

**Meeting 10: Sunday, 22 January 2017**

Worked on Lab 7 & 8

Work log:

*Over 3 hour meeting:*

Shaun and Khaled – Lab 7

Logan and Trinity – Lab 8

Trinity - Journal

**Meeting 11: Monday, 23 January 2017**Worked on Lab 7 & 8

Work log:

*Over 1 hour meeting:*

Shaun and Khaled – Lab 7 and RTL review

Logan and Trinity – Lab 8

Trinity - Journal

**Meeting 12: Tuesday, 24 January 2017**

In our RTL for the instructions *load upper immediate, load lower immediate* and *load to register*, we decided to denote the immediate value by the bits in the 16-bit immediate itself rather than where they are in the instruction. We weren’t sure how to access the correct parts of the instruction. In these instructions, we changed some logical code to simply shift the immediates left by 1.

We also decided to add a C register component that receives the value of $cr no matter what input goes into the register file.

Work log:

*Over 2 hour meeting:*

Trinity and Logan (with assistance from Shaun and Khaled) – Revised and finished datapath

Trinity - Journal

Everyone – Revised RTL and divvied up assignment

*After meeting:*

Logan – Block diagram of datapath (estimated 1 hours) **Actual: expected**

Implementation plan for each component (estimated 1 hour) **Actual: 20 minutes**

Khaled – Implementation and tests of Mem, ZE, and SE (estimated 2 hours) **Actual: Unknown/Incomplete**

Shaun – Update unit tests (estimated 30 minutes) **Actual: 5 minutes**

Integration plan and testing (estimated 1 hour) **Actual: 20 minutes**

Trinity – Control signal descriptions (estimated 30 minutes) **Actual: expected**

Specs for Control Unit in Component List (estimated 1 hour) **Actual: expected**

Implementation and tests of ALU, SL1 (estimated 4 hours) **Actual: incomplete**

**DISCUSSION OF TEST STRATEGY**

Our testing strategy consisted mostly of trying to anticipate problems before they occurred. We think of what might go wrong with a component and write tests to catch those situations. After that, we write general cases for expected outputs with non-extreme or expected data. When these tests are finished, we run through all inputs that satisfy the tests and make sure the output is expected/acceptable.

**HOW OUR ARCHITECTURE CHOICE AFFECTED OUR DATAPATH**

Our choice in architecture – Load/Store with Accumulator, caused us to have a lot of extra muxes to choose the value from our special computation result register as well as a special register in the datapath that always stores the value of the computation result register. Al of this was caused by our accumulator computation register.

Because our design is also load/store, we found it necessary to have a memory unit as well as a register file to hold data values. Between these two components and our ALU, we found it necessary to have special temporary register such as IR, MDR, A/B/C to hold necessary values between stages in the datapath.

**Meeting 13: Wednesday, 25 January 2017**