POWERPUFF TEAM PROJECT REPORT

Vu Dinh and Alejandro Alfaro

Version 1.0

**1) Program description: This is a write-up of 1 to 2 pages about your program. Describe your design philosophy, flow chart, any algorithms that you were especially proud of, any algorithms that you got from other sources ( copied, downloaded, etc. ). Using canned routines is OK but you MUST cite your use of them. Remember, canned library routines are very easy to spot. Also, the more routines that you appropriate from other sources, the more expectations I’ll have for the coding of your part of the program. The program description should also cover any limitations that you are aware of.**

### DESIGN:

The program has a centralized opcode-decoder, which:

looks at the first 4 bits of the instruction to decide which “group” the opcode belongs to,

looks for unique traits in the instructions INSIDE THAT GROUP

This part works flawlessly.

The program then attempts to extract the pieces of information from the instruction word by masking and shifting the bits. This never was a problem, although we did change to doing it this way after a while when we were struggling with doing it a different way (range comparison).

With the relevant pieces of data in the corresponding addresses, we process them with subroutine calls (for example, DECODE\_TWO\_BIT\_SIZE would expect data inside address SIZE as input, and modify data inside SIZE\_OUT as output).

SIZE: %10 ---> SIZE\_OUT: “.L”

The program then arranges the output pieces in the correct order, depending on the specific opcode and the various conditions for the output formatting (such as direction).

We severely underestimated the amount of work required to put together the code, so even though we have all the working parts, tiny mistakes here and there made it very difficult to integrate them on time.

**2) Specification: 1~2 pages.This is a simple list of what your program does.**

### Program Layout

- ORG 1000

- some variables (global constants placed after the program’s code)

- output buffer at $7000

### Functionality

- can decode MOVE, ADDQ, DIVS, (SUB/SUBA)

- can easily decode 20+ more opcodes with minor bugfix work (beside MOVEM)

As mentioned above: The program can uniquely identify among all opcodes with no problem. This is done by using bit-masking and shifting to group the opcodes into similar groups, and then further identify from there.

**3) Test Plan: 1~2 pages.This is a description of how you tested the program. It should also contain a description of your team’s coding standards. If you have, please include your testing files as well.**

We write an exhaustive list of possible ea mode variations, but we usually only change one thing at a time (for example, MOVE.B D0,D0; MOVE.W D0,D0; MOVE.L D0,D0). We are aware that some crossed modes are not covered, so we also provide random combinations of ea modes (and values). This still is not exhaustive (since there can be up to 100 different combinations of ea modes for some opcodes).

We feel that since our design separates the fields from one another, there should be no reason for one ea to affect another. So as long as all variations of each ea works fine, all possible pairings between them should also work fine. The only problem we foresee is when assembly tries to make an instruction more efficient and changes the move to something else, for example, when doing MOVE. W D0,A0, the compiler changes it to MOVEA.W D0,A0, And they end up having a similar, if not the same, values.

**4) Exception report: This is your opportunity to describe problems that you’ve encountered but couldn't fix, or chose not to fix. Anything that you feel deviates from your intended program. Also, this is where you can describe what you were able to complete in the time allotted versus what the assignment asked for. This should definitely include the results of your testing if you found defects but didn't fix them. In an ideal situation, I should be able to just read your documentation and your source listing and give you a grade, without needing to run the program. Of course, I will run the program, but I hope that you get the idea.**

We chose to build the program laterally, i.e. build all the parts, then integrate them together. As of right now (very close to the deadline) we have almost all the parts written (except for subroutines that process the information for MOVEM and Bcc); it's just that integrating them into the main program (which has several working opcodes) is taking a significantly higher amount of time than anticipated. Tiny bugs like moving WORD-sized data to variables declared to be 1-BYTE large or moving data into D0 and then (for no reason at all), using D5 instead are keeping us from completing the specifications.

We are at fault for not documenting these bugs as we fix them, since they seem to be recurring for certain groups of opcodes, and debugging them from the start is a lengthy process of root analysis.

You should have no problem testing any variants of the opcodes that we do support. Anything else would not work (because they are not included in the code yet, and instead in separate files).

**5) Team assignments and report: A description of how you organized your team’s tasks. That is, "Who did what and how". You should specify the amount of the coding, as a percentage, the member did in the project. This information is VERY important as it will be a source for the separate grading.**

We separated our tasks into who was the better coder and who understood assembly language the most. The third part of the project was split between Vu and Alejandro because we only had two members. Having only two members proved to be a bit of a hassle but we were able to make it work.

Alejandro dealt with half of I/O, the processing and preparation of the effective address data. This involved getting to understand how each instruction worked and what were their respective values. After each range for the instructions were obtained, files for each one had to be made in order to get the correct output. The ranges were obtained by writing each instruction on a separate file, and the instruction was written with all the possible scenarios that the instruction could do, this was taken from the manual. The part of the output that Alejandro did consisted of placing the output in correct order depending on the instruction. By looking at the manual, he was able to see the syntax of each instruction and write the output for each one accordingly.

Alejandro did about 70% and Vu did about 30% of the coding, however, Vu did most of the brain work about the program, came up with ideas and ways to approach the problems that arose in the disassembler. Since Alejandro was not very familiar nor understood assembly language as well, Vu was able to help him with understanding what was going on, when Alejandro understood the problems, he was able to do it alone and was able to learn and understand more about assembly language.

---

Vu:

* improvised several ways to perform basic programming constructs (loops, function calls, naming convention, scope management) and streamlined it for the project after better methods were introduced in class
* decoded the entire opcode base with bit-masking and bit-shifting
* helped Alejandro move from doing range-comparison for ea decoding to bit-masking + bit-shifting which proved to be simpler
* helped with half of the output formatting and code integration
* miscellaneous utility subroutines like conversions