Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Program Description:

Our program acts as a disassembler in the easy68k program. It takes in a starting and ending address from user input through the console and will disassemble the instructions within the range given by the user and revert the code to its original form.

List of Instructions:

- NOP
- MOVE, MOVEQ, MOVEM, MOVEA
- ADD, ADDA, ADDQ
- SUB
- LEA
- AND, OR, NOT
- LSL, LSR, ASL, ASR
- ROL, ROR
- Bcc(BGT,BLE,BEQ)
- JSR,RTS
- BRA

Effective Addressing Modes:

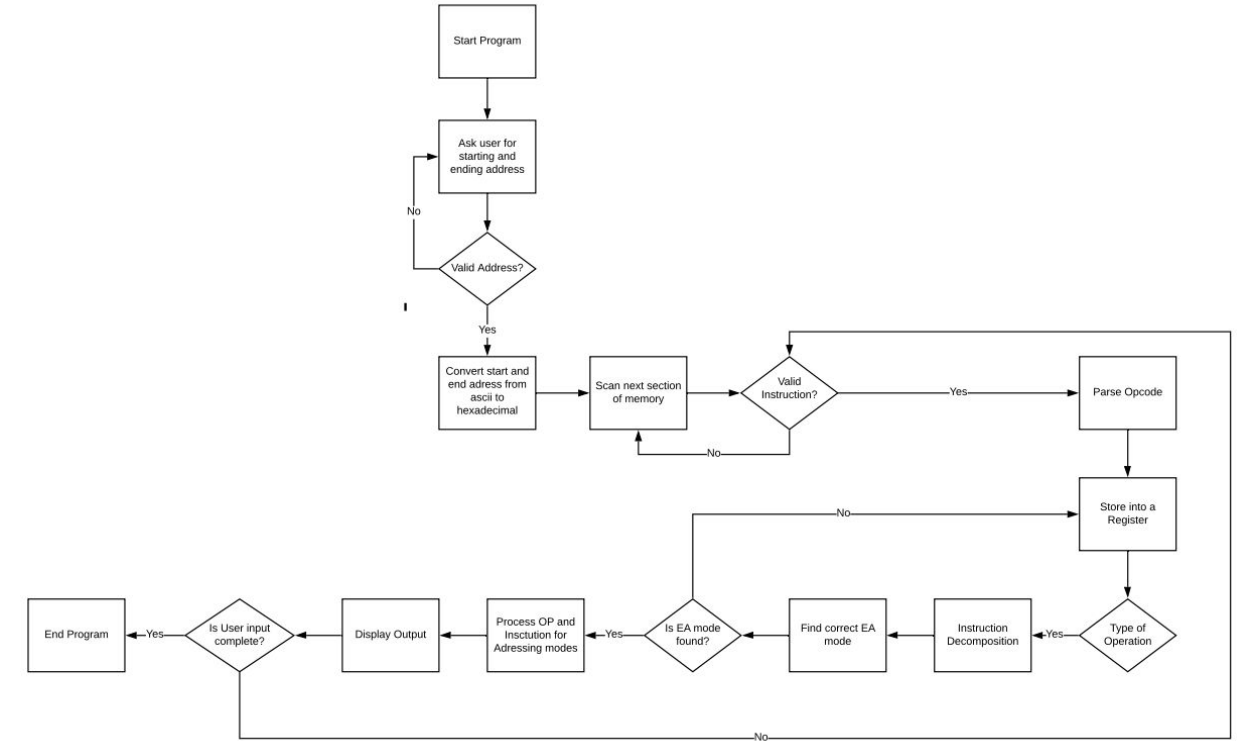
- Data Register Direct
- Address Register Direct
- Address Register Indirect
- Immediate Addressing
- Address Register Indirect with Post-Incrementing
- Address Register Indirect with Pre-decrementing
- Absolute Long Address
- Absolute Word Address

Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Design philosophy:

Our design philosophy started with the idea to have a collaborative setting to build an idea of approaches we wanted to take in order to develop a full flushed out disassembler. In our early iterations we started by understanding the inputs and what it would take to decompose the instructions into the machine code provided per instruction. Once we had some high level of understanding, we started by drafting rough concepts of what was to be expected out of separate parts of the project(IO,EA, OPcode). We wanted to aim to have functionality and worry about the form and integration of the different components once there was a success in how an operation was completed.

Flow chart



Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Specification:

I/O:

Displays messages to the user through the easy68k console, prompting them to enter a starting and ending address in hexadecimal.

The program will read in the user input as a string input and if the user input is valid, the program will convert the input from the ascii values to hexadecimal to be converted into machine code.

Once the opcode and instruction set has been decoded, the IO process will take the results and append them to an address register to be outputted to the console. After the user should be prompted to go again if they do choose to, else program exits.

OP code:

Once the input has been read in the input will be decoded to check for the supported OPcode in our listed specification. This will iteratively check the state of the instruction and validate its input into the system.

If successful it will continue to prepare for output according to the instruction format, or rejected as a DATA or unsupported instruction from our disassembler.

EA code:

The effective addressing applies to the respected source and destination modes for the instruction that is being decoded. Given our specific supported addressing modes, the EA will mask and shift bits to be able to isolate the snippet of variables that act differently per addressing modes within the constraints of an instruction supported.

Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Test Plan:

Testing Instruction:

- Effective addressing modes
- Data size
- Data types
- Ascii to hex conversion

In addition, we also looked at the parts of the instruction decomposition that would help us identify the specification of a particular instruction that would give us the appropriate output for readable display options.

When the program size was smaller we would first just test a couple of addresses to see what the output would look like and obtain an idea of what a display function would output and if it was valid and correct with the input taken into the buffer. When we were testing the program we originally set out to give the program correct inputs as well to ensure that if the users understood the preconditions, then the output would come out correct. Error checking was later implemented to better adjust to the format of the user in the cases of an invalid operation, address, input.

Testing file is attached to the final submission .zip file.

Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Exception Report:

In the earlier weeks of the quarter it was hard to organize as a group due to the current events and campus being closed. Our group strived to hold each other accountable as we set up meetings every other week while increasing the frequency of our meetings as the project deadline came closer. At the beginning our work was very team focused as we worked together to solve planning and design issues for our program. As the quarter progressed and the deadline came closer, our group felt it was necessary to adopt a divide and conquer strategy in order to finish the disassembler on time. We set up deadlines for integration each week and for each person as we knew it would help keep ourselves accountable for the code we were individually responsible for. We split up the project work into three different parts, I/O, OP-code, and EA, as suggested by the professor. Using this methodology of working together to get an idea of how our finished project would turn out and then working individually to speed up the code process allowed our team to finish the project before the deadline.

For the instruction read, we recognized that for the beginning of the sections of the read in we were able to get the instructions disassembled but for some reason when we looped over the addresses again, our program began to skip over address spaces. In addition, there were some cases where the move address would get the effective addressing mode wrong. We were able to resolve the problem through regression testing and finding registers that were operating on a separate op instruction, so we reset the value to run a separate subroutine.

Once we had finally finished the project, we felt extremely relieved as there was a huge amount of stress on our shoulders. Given the circumstances, we would have wished to do more testing but we felt very satisfied to be able to do step by step integration to make sure that our code was able to accomplish the tasks given the requirements by the professor.

Date: 6/6/2020
To: CSS 422, Spring 2020
From: Team Nibble's a crowd: Samuel Debesai, Lu Lu, Bryan Song
Subject: Report

Team Assignments and Report:

In the beginning of the project, our group worked together in order to create a plan to follow for the rest of the quarter to stay on track to finish the project before the deadline. Some of the code was written together as a team but as the quarter went on and the time to finish the project decreased, our team decided it would be in our best interests to divide up the project into different parts as the professor had suggested to meet the project deadline. We decided to follow the professor's suggestion and split the project work into three different parts, I/O, OP-Code, and EA. Bryan worked on the I/O, Samuel worked on the OP-Code, and Lu Lu worked on the EA. For each subtask(I/O,OP-Code,EA), we broke down what the disassembler required and built the project piece by piece until we had three separate working code bases that each accomplished a different task. When all the team members had finished their part of the disassembler, our team met up together again to piece together the project and define standards such as what data registers and addresses were being used for what.

Percentage of Work:

Bryan: 33%

Samuel: 33%

LuLu: 33%