Program Specification

General Overview

Our program is designed to disassemble binary data into human readable assembly code. The user is prompted to enter start and ending memory locations. The valid range for the input is currently $3500 ~ $9FFF because we didn't conduct enough tests on the program beyond the range $10000. The program requires user to type the range of the value in hexadecimal format, (not case sensitive, but valid characters are ONLY 0-9 and a, b, c, d, e). After getting input from the user, the program will validate the input, and start disassembling the code.

The program uses a lot of variables, and they are saved in two separate files named 'Const_Variable_Setting.x68', and 'Variable.x68'. By using a lot of variables, we were able to minimizing using data registers, and source registers.

The program stack begins at $100000 so we can safely utilize it without worrying about overriding our program. In other words, although we push lots of data into stack, the data in stack will never reach the main program stored in memory.

The Program flow (General)

The program uses an iteration loop, to analyze the data loaded into memory word by word. The main loop tests the leading four bits of the current memory location and depending on the first four bits it branches to another another subroutine. We have 14 subroutines and on any other case, the program will show an invalid message.

When the program gets into the subroutine from the main loop, the program will validate remaining 12 bits. After validation, the program will begin to analyze the with variables that we made.

DEST_REGISTER	3 bits saving place for destination register
DEST_MODE	- 3 bits saving the destination mode
SRC_REGISTER	- 3 bits saving source register
SRC_MODE	- 3 bits saving source mode
SIZE	- 2 bits saving size
DATA_EIGHT_BIT	- 8 bits saving displacement

After initializing these variables, we invoke ADDRESS_READ_DECISION_LOAD subroutine, and it will analyze the variables: DEST_REGISTER, DEST_MODE,

SRC_REGISTER, and SRC_MODE to decide to read more memory from current reading or not. Through the subroutine ADDRESS_READ_DECISION_LOAD, the program may or may not initialize the following variables:

- SRC_NUMBER_DATA (Long size variable to save source Effective address number data)
- DST_NUMBER_DATA (Long size variable to save destination Effective address number data)

Before our program executes the next step, we have validated our memory, so now we print out the instruction with the size tag and print out the operands. To print out operands, we made three different subroutines.

INITIAL_TWO_EA_LOAD_OUT	Analyze SRC_MODE, SRC_REGISTER and SRC_NUMBER_DATA to print out only one operand.
INITIAL_FOUR_EA_LOAD_OUT	Analyze SRC_MODE, SRC_REGISTER, DEST_MODE, DEST_REGISTER, SRC_NUMBER_DATA, DEST_NUMBER_DATA to print out two operands.
MCL_MM_OPERAND	The subroutine only for movem instruction. It will print out the <list> only

After printing the operands, the program go back to the main loop.

Additional Note

We allocate our size variable to print out the size tag (‘.b’,’.w’,’.l’), so we figured out it is better to use only one pattern.

Default size instruction

#$01	BYTE
#$11	WORD
#$10	LONG

Size Converter type 1
#0 → BYTE
#1 → WORD
#2 → LONG

Size converter type 2
%000 V %100 → BYTE
%001 V %101 → WORD

%010 V %110 → LONG