**Team:** ManPower

**Members:** Yezin Rashid, Andy Thompson, Race Brocx

**Project Documentation**

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.

Our design for the disassembler program uses an input/output management routine to control user interaction. The user I/O routines scans memory locations, based on user input and stores the address words as an ascii value. The ascii values are converted to decimal values and stored using our conversion algorithm.

The program checks if the ascii value is less than $40. If it is, the conversion algorithm subtracts 37 from the ascii value if it is found to be a letter to get the hex value. Otherwise, it is a numerical value, 30 is subtracted instead. The pseudo code for the conversion algorithm is shown in figure 1.

**Figure 1:** Conversion Algorithm

Return hex convertascii(address){

If(address < 40){

Return (address – 30)

}

Else

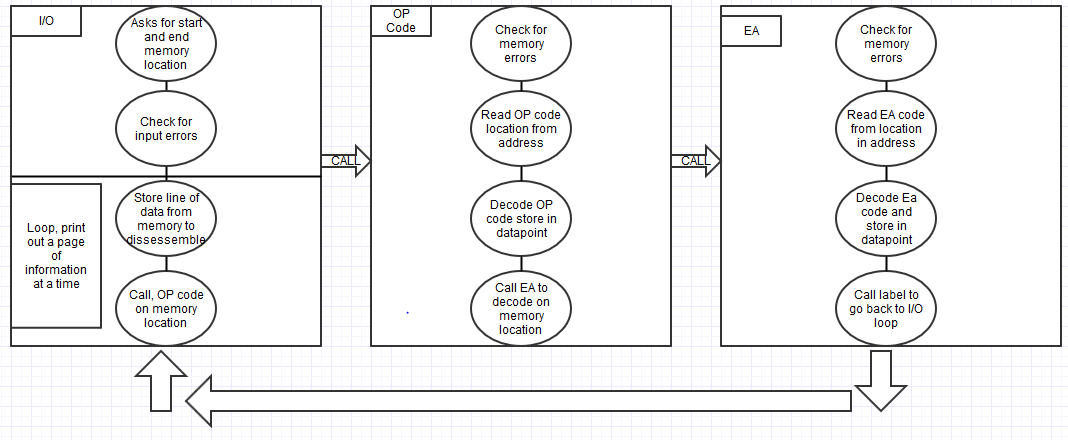
Return(address - 37)

}

Once we have the converted memory values, we begin the process of decoding. The decoding process initially, compares the 12th, 13th, 14th, and 15th bits to all possible permutations of those bits to route to the subroutine related to that particular bit pattern. The four bit permutation determines what Opcode the input data is using.

For example, if we find that the 12-15 inclusive bits are 1001, we know that the Opcode is either SUB or SUBA. From there we determine the length being used with 3rd, 4th, and 5th bits, (D2) and the EA Mode being used with the 6th, 7th, and 8th bits (D3). The proper length and EAs are then printed. Figure 2 shows the flow of the program.

**Figure 2:** Flowchart



The two most difficult opcodes to account for were the branching statements and the MOVEM opcodes. Designing for branching statements was time consuming because of the sheer number of different branching statements that we needed to handle.

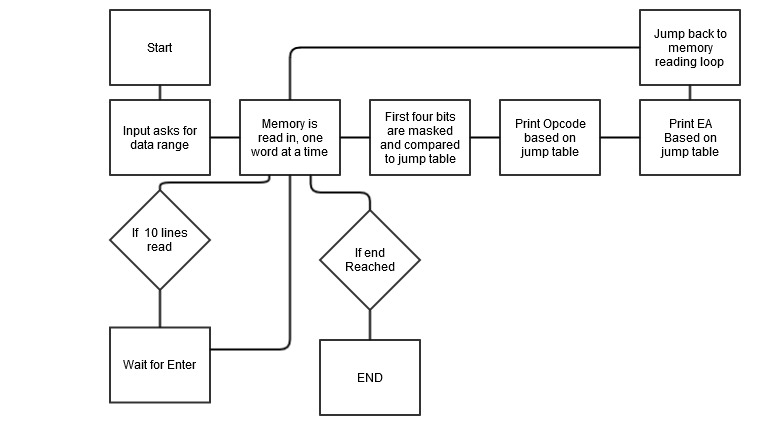
MOVEM was difficult to account for because of its unique nature. Specifically, handling printing addresses ranges required special subroutines that are unique to MOVEM.

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

The program begins by asking the user to specify a memory location that will be used by the program to dissembler that memory location. See figure 4 for an example of the starting information.

The final representation the project has a more refined design that can be seen in the flow chart in Figure 3.

**Figure 3:** Final Flow Chart



**Figure 4:** I/O initialization



After being supplied with a starting address and ending address the disassembler attempts to decode program in memory. If a given address is invalid the program prints an error message. See figure 5 for the error message.

**Figure 5:** Address error



If the address range given is valid, the I/O reads 1000 words starting from the starting address. If it reads a null value, or if it reaches the end of 1000 words.

Each word is dissembled by analyzing bits to determine the Opcode and length of the command. The program reads bit masks of the word length being read to determine what Opcode the address is. The Opcode is then printed.

From there, EA, is decoded in the same manner as the Opcode. The program uses bit masks to check what EA is being used. The EA is then printed.

If the program finishes reading 1000 words, the program waits for the user to press enter to continue reading.

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 tested our program by creating all possible different types of valid input that is required, to check for the correct opcode and effective address print outs. We created a program with these random commands that encompassed possible input and compiled them into a runnable program.

The test code file that we created is shown in Figure 6. The test file contains comprehensive testing values for the disassembler.

**Figure 6:** Test Code

SADDR EQU $00007000 \*starting address

START ORG SADDR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MOVE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MOVE.B D0,D0

MOVE.W D0,(A1)

MOVE.L D1,(A2)+

MOVE.B D5,-(A3)

MOVE.W D1,$4444

MOVE.L D4,$44445555

MOVE.W A0,D0

MOVE.W A0,(A1)

MOVE.L A1,(A2)+

MOVE.W A5,-(A3)

MOVE.W A1,$4444

MOVE.L A4,$44445555

MOVE.B (A0),D0

MOVE.W (A0),(A1)

MOVE.L (A1),(A2)+

MOVE.B (A5),-(A3)

MOVE.W (A1),$4444

MOVE.L (A4),$44445555

MOVE.B (A0)+,D0

MOVE.W (A0)+,(A1)

MOVE.L (A1)+,(A2)+

MOVE.B (A5)+,-(A3)

MOVE.W (A1)+,$4444

MOVE.L (A4)+,$44445555

MOVE.B -(A0),D0

MOVE.W -(A0),(A1)

MOVE.L -(A1),(A2)+

MOVE.B -(A5),-(A3)

MOVE.W -(A1),$4444

MOVE.L -(A4),$44445555

MOVE.W $1234,D0

MOVE.W $1234,(A1)

MOVE.W $1234,(A2)+

MOVE.W $1234,-(A3)

MOVE.W $1234,$4444

MOVE.W $1234,$44445555

MOVE.L $12345678,D0

MOVE.L $12345678,(A1)

MOVE.L $12345678,(A2)+

MOVE.L $12345678,-(A3)

MOVE.L $12345678,$4444

MOVE.L $12345678,$44445555

MOVE.B #$12,D0

MOVE.W #$1234,(A1)

MOVE.L #$12345678,(A2)+

MOVE.B #$12,-(A3)

MOVE.W #$1234,$4444

MOVE.L #$12345678,$44445555

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MOVEQ\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MOVEQ #$12,D0

MOVEQ #$12,D1

MOVEQ #$12,D2

MOVEQ #$12,D3

MOVEQ #$12,D4

MOVEQ #$12,D5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MOVEM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MOVEM (A0),D0-D5

MOVEM (A1)+,D0-D3

MOVEM $2222,A2-A7

MOVEM $33333333,A3-D4

MOVEM A2-A0,(A0)

MOVEM D7-A7,-(A1)

MOVEM A3-A2,$2222

MOVEM A2-A3,$33333333

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ADD.B D1,D0

ADD.W A3,D1

ADD.L (A2),D2

ADD.B (A5)+,D3

ADD.W -(A7),D4

ADD.W $1234,D5

ADD.L $12345678,D6

ADD.W #$1234,D7

ADD.L #$12345678,D0

ADD.W D0,(A1)

ADD.L D1,(A2)+

ADD.B D5,-(A3)

ADD.W D1,$4444

ADD.L D4,$44445555

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADDA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ADDA.W D1,A0

ADDA.W A3,A1

ADDA.L (A2),A2

ADDA.W (A5)+,A3

ADDA.W -(A7),A4

ADDA.W $1234,A5

ADDA.L $12345678,A6

ADDA.W #$1234,A7

ADDA.L #$12345678,A0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADDQ\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ADDQ #1,D0

ADDQ #1,A1

ADDQ #1,(A2)

ADDQ #8,(A3)+

ADDQ #8,-(A4)

ADDQ #5,$1234

ADDQ #3,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUB\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SUB.B D1,D0

SUB.W A3,D1

SUB.L (A2),D2

SUB.B (A5)+,D3

SUB.W -(A7),D4

SUB.W $1234,D5

SUB.L $12345678,D6

SUB.W #$1234,D7

SUB.L #$12345678,D0

SUB.W D0,(A1)

SUB.L D1,(A2)+

SUB.B D5,-(A3)

SUB.W D1,$4444

SUB.L D4,$44445555

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SUBA.W D1,A0

SUBA.W A3,A1

SUBA.L (A2),A2

SUBA.W (A5)+,A3

SUBA.W -(A7),A4

SUBA.W $1234,A5

SUBA.L $12345678,A6

SUBA.W #$1234,A7

SUBA.L #$12345678,A0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SUBI.B #$12,D0

SUBI.W #$1234,(A2)

SUBI.L #$12345678,(A3)+

SUBI.B #$12,-(A4)

SUBI.W #$1234,$1234

SUBI.L #$12345678,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MULS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MULS.W D1,D0

MULS.W (A2),D2

MULS.W (A5)+,D3

MULS.W -(A7),D4

MULS.W $1234,D5

MULS.W $12345678,D6

MULS.W #$1234,D7

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DIVS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DIVS.W D1,D0

DIVS.W (A2),D2

DIVS.W (A5)+,D3

DIVS.W -(A7),D4

DIVS.W $1234,D5

DIVS.W $12345678,D6

DIVS.W #$1234,D7

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LEA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LEA (A2),A2

LEA $1234,A5

LEA $12345678,A6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CLR.B D0

CLR.L (A2)

CLR.B (A3)+

CLR.W -(A4)

CLR.L $1234

CLR.B $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AND\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AND.B D1,D0

AND.L (A2),D2

AND.B (A5)+,D3

AND.W -(A7),D4

AND.W $1234,D5

AND.L $12345678,D6

AND.W #$1234,D7

AND.L #$12345678,D0

AND.W D0,(A1)

AND.L D1,(A2)+

AND.B D5,-(A3)

AND.W D1,$4444

AND.L D4,$44445555

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ANDI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ANDI.B #$12,D0

ANDI.W #$1234,(A2)

ANDI.L #$12345678,(A3)+

ANDI.B #$12,-(A4)

ANDI.W #$1234,$1234

ANDI.L #$12345678,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EOR.B D1,D0

EOR.W D3,(A2)

EOR.L D5,(A3)+

EOR.B D1,-(A4)

EOR.W D3,$5555

EOR.L D2,$55556666

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EORI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EORI.B #$12,D0

EORI.W #$1234,(A2)

EORI.L #$12345678,(A3)+

EORI.B #$12,-(A4)

EORI.W #$1234,$1234

EORI.L #$12345678,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LSL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LSL D1,D0

LSL #1,D2

LSL #8,D3

LSL (A4)

LSL (A5)+

LSL -(A6)

LSL $1234

LSL $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LSR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LSR D1,D0

LSR #1,D2

LSR #8,D3

LSR (A4)

LSR (A5)+

LSR -(A6)

LSR $1234

LSR $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ASL D1,D0

ASL #1,D2

ASL #8,D3

ASL (A4)

ASL (A5)+

ASL -(A6)

ASL $1234

ASL $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ASR D1,D0

ASR #1,D2

ASR #8,D3

ASR (A4)

ASR (A5)+

ASR -(A6)

ASR $1234

ASR $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ROL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ROL D1,D0

ROL #1,D2

ROL #8,D3

ROL (A4)

ROL (A5)+

ROL -(A6)

ROL $1234

ROL $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ROR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ROR D1,D0

ROR #1,D2

ROR #8,D3

ROR (A4)

ROR (A5)+

ROR -(A6)

ROR $1234

ROR $12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BCHG\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BCHG D1,D0

BCHG D3,(A2)

BCHG D5,(A3)+

BCHG D1,-(A4)

BCHG D3,$5555

BCHG D2,$55556666

BCHG #$12,D0

BCHG #$1234,(A2)

BCHG #$12345678,(A3)+

BCHG #$12,-(A4)

BCHG #$1234,$1234

BCHG #$12345678,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CMP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CMP.B D1,D0

CMP.W (A2),D2

CMP.L (A5)+,D3

CMP.B -(A7),D4

CMP.W $1234,D5

CMP.L $12345678,D6

CMP.W #$1234,D7

CMP.L #$12345678,D0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CMPA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CMP.W D1,A0

CMP.W (A2),A2

CMP.L (A5)+,A3

CMP.W -(A7),A4

CMP.W $1234,A5

CMP.L $12345678,A6

CMP.W #$1234,A7

CMP.L #$12345678,A0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CMPI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SUBI.B #$12,D0

SUBI.W #$1234,(A2)

SUBI.L #$12345678,(A3)+

SUBI.B #$12,-(A4)

SUBI.W #$1234,$1234

SUBI.L #$12345678,$12345678

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bcc(BGT,BLT,BLE,BVS)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BCC $1111

BCS $2222

BEQ $3333

BGE $4444

BGT $5555

BHI $6666

BLE $7777

BLS $8888

BLT $9999

BMI $AAAA

BNE $BBBB

BPL $CCCC

BVC $DDDD

BVS $EEEE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JSR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JSR (A0)

JSR $1111

JSR $22222222

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RTS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RTS

END START

We loaded the assembled test code into a program and loaded it into memory while executing our disassembler. We ran the code for the memory locations that we filled in with test code which were 7000 through 7076.

After running the disassembler, we compared the output to the code in the test code file. If it matches we know that those commands were disassembled correctly. When checking if commands worked correctly, we tested every valid effective address and opcode type combination possible. At the end of our testing, we found no missing or incorrect effective address/opcode combinations.

We also ran the program using non-valid commands to ensure that the correct output was given when invalid input is given. When testing invalid inputs, our program prints a message that warns of invalid memory input, rather than printing the invalid memory.

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.

**Exception list**

1. When memory is loaded to be dissembled that contains an AND command with immediate data, the command is stored as an ANDI function. Because of this, if the user loads an AND command into memory, it will be saved and interpreted as an ANDI command.

2. For some commands, when non immediate data is disassembled and printed, unnecessary '#' signs are printed for addresses, rather than immediate data.

3. The Opcode for MOVEM is not currently working correctly based on our tests. We understand what is theoretically needed to complete the code, but we did not have time to complete the subroutine for decoding MOVEM. MOVEM can potentially cause the program to crash in some circumstances.

4. Some of the values that are printed are decimal values when they should be in hexadecimal format.

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.

Andy Thompson – Andy was assigned to do the create subroutines to disassemble the Opcodes. Andy completed the Opcode portion early and integrated subroutines for Effective Addressing within the Opcodes decoding. Because he completed Effective Addressing within Opcodes, Race’s version of Effective Addressing was not used. 50%

Race Brocx – Race was assigned to create subroutines to disassemble the Effective Addresses. Race finished building some effective addressing decoding subroutines. However, Andy was able to integrate effective addressing as a feature of his Opcode decoding process. Because of this, the code that Race originally wrote for Effective Addressing was not used. Race did help with testing and writing bug solving code. Race also finished some of the code for Effective Addressing. Race completed the documentation because his code section was not used. 20%

Yezen Rashid – Yezen was assigned to create the I/O to interact with the user and read memory. Yezen was able to successfully create code to read in memory and interact with the user appropriately. 30%