Contents

[Available variables 1](#_Toc507420860)

[Current program flow 1](#_Toc507420861)

[GET\_START\_ADDRESS: 1](#_Toc507420862)

[CONVERT\_ASCII\_TO HEX 2](#_Toc507420863)

[ERROR\_INVALID\_INPUT 2](#_Toc507420864)

[FINISH\_CONVERT\_STARTING\_ADDRESS 2](#_Toc507420865)

[FINISH\_CONVERT\_ENDING\_ADDRESS 2](#_Toc507420866)

[PREPARE\_START\_AND\_END\_ADDRESS 3](#_Toc507420867)

[PARSING\_OP\_CODE 3](#_Toc507420868)

[FIRST\_BYTE\_SECTION 3](#_Toc507420869)

[FIRST\_BYTE\_IS\_4 3](#_Toc507420870)

[FIRST\_BYTE\_4\_SECOND\_BYTE\_E 3](#_Toc507420871)

[PRINT\_OUT\_NOP 4](#_Toc507420872)

[PRINT\_OUT\_RTS 4](#_Toc507420873)

[FINISH\_ONE\_INSTRUCTION 4](#_Toc507420874)

# Available variables

I have starting address, ending address, a temporary var, a destination var to hold the number, a destination mode to hold the mode, a source mode and a source var

# Current program flow

# GET\_START\_ADDRESS:

A welcome prompt is displayed as a welcome message, with line and carry feed

It will ask for user input and convert the ASCII to HEX

Used: A1,D0. Load data to print and put trap task

# CONVERT\_ASCII\_TO HEX

It will save the data into D0. Compare if #$30 Is less than. Since valid data is only in the range #$30 - #$39 and $41-46. If less than, then it is invalid. If greater than 39. Then maybe from A-F

If not. Must be between 30 – 39. Subtract 30 from it => number . So this all the way until the length that is stored in D1 is 0. After each time shift to the left 1 bytes to make room for other

Used: A1 increment, D0 that store that byte we are at, D3 that is used to store the entire address after shifting

# ERROR\_INVALID\_INPUT

Display the prompt to user and end the program

Suggestion: Maybe can keep looping until valid input are input

Suggestion: Maybe also do a check of valid address: aligned or not

Used: A1 and D0. Print and put trap task

# FINISH\_CONVERT\_STARTING\_ADDRESS

Get called when ever the length of the input is 0. At that time, we know that we have looped through all the variables. Check if we should move to convert ending address. Move the data to starting address

After the convert starting address , increment a counter that let user know that D2 now hold value 1 , indicating that we should convert ending address

Variables used: D2,D3, Starting Address

# FINISH\_CONVERT\_ENDING\_ADDRESS

Get called from the FINSIH\_CONVERT\_STARTING\_ADDRESS. Called Prepare Start and End Address

Move the data (address in D3 after conversion) to ending address

Suggestion: Maybe find a way to group the convert ending and starting address together

Used: D2, TEMP\_VAR,D3

# PREPARE\_START\_AND\_END\_ADDRESS

Load the table that represent the first byte, FIRST\_BYTE\_JUMP\_TABLE

Load the starting address into A2 to start iterating

Load the ending address into A3 to start iterating

Used: A0,A2,A3

# PARSING\_OP\_CODE

Go through each address starting from A2 (start address). Move the 4 bytes op code into D3. Save the opcode into a temporary variables TEMP\_VAR. Begin isolating the first byte of the word by bits shifting

Jump to the appropriate first byte

Used: A2,D3,A0 and TEMP\_VAR

Suggestion: Wait for the implementation of other features that can detect if the instructions require 0,1 or 2 operands. At the moment, Did not take into the possibility of having additional operand after op code

# FIRST\_BYTE\_SECTION

## FIRST\_BYTE\_IS\_4

There are only NOP,LEA,RTS,JSR,NEG

Know that the first byte is 4. Checking the second byte. If it is E. Then must be NOP,JSR,RTS

Else must be LEA,NEG

Used: D3, TEMP\_VAR

### FIRST\_BYTE\_4\_SECOND\_BYTE\_E

For NOP, the third and fourth byte are 71.

For RTS, it is 75

Variables used: D3

# PRINT\_OUT\_NOP

Load hardcoded NOP into A1 and print out

Branch to waiting for user to press enter

Branch to finish one instruction

Used: A1,D0

# PRINT\_OUT\_RTS

Same like NOP, added the hardcoded value to A1 and print out and branch

# FINISH\_ONE\_INSTRUCTION

Move the data to next place. Because of word , move 2

Go to parsing next op code

Used: A2

Suggestion: Added case check incase of long operand. Might be 1 or 2 more that specifies the data

# CHECK\_TYPE\_DATA

Check and save the 3 bits that correspond to source register, source mode, destination register, destination mode into the variables defined above

Used: D3, Destination\_VAR, DESTINATION\_MODE, SOURCE\_MODE, SOURCE\_VAR

# PRINT\_LEA

Check type data first then print out the appropriate LEA command with the addition of source and destination operand