# Team Fireball: Binh Nguyen, Derek Willms, Brian Quigley

# Team Fireball Specification

* Program starts.
* Display Welcome Message (Awesome Fireball ASCII Art!!!).
* Display prompt for starting address.
* Read user input for starting address.
* Convert the ASCII input into Hex.
* Check if the input is odd and re prompt to user for new input if necessary.
* Check if the input is less than or equal the minimum starting address (minimum starting address is $7000) and re prompt to user for new input if necessary.
* If starting address is valid, store the starting address in A6, then prompt user for ending address.
* Read user input for ending address.
* Convert the ASCII input into Hex.
* Check if the input is odd and re prompt to user for new input if necessary.
* Check if the input is less than or equal the starting address and if the input is greater than the maximum ending address (maximum ending address is $00FFFFFE), and re prompt to user for new input if necessary.
* If ending address is valid, store the ending address in D6.
* The current address is then display to the output.
* The program then reads a word from memory and advance the pointer in A6.
* The word data is then separated as follow:
  + Bits 0-2 is put into D2
  + Bits 3-5 is put into D3
  + Bits 6-8 is put into D4
  + Bits 9-11 is put into D5
  + Word is put into D0 and an indirect jump based on the bits 12-15 is performed
* The program continues at the specific decoding from the jump table based on those 12-15 bits.
* The data is then used to decode into the appropriate instruction by putting the instruction into the print buffer in A1.
* Additional data can be retrieved from A6 if necessary and the memory pointer in A6 is advanced.
* If data does not represent a valid required instruction, then the program will reload the print buffer in A1, then print the data with the actual hex value to the output.
* If data is a valid instruction, all operands are retrieved (if any) and the program print the instruction and operand to the output with character return.
* The program checks for current memory pointer with the ending address to see if reach the end of the data range and stop if necessary.
* The program also increases the counter for number of lines display on screen and wait (ask user to press Enter) if the number of lines equals the max\_line (This make sure only a certain amount of lines are display per screen).
* The program then repeats the loop to print the current address of the memory pointer from A6 and read the next word for decoding.
* At the end, then the program will prompt user to press ‘y’ or ‘Y’ to rerun the program again. Any other input from keyboard will terminate the program.
* The program display ending message and terminate if ‘y’ or ‘Y’ is not the input.

The program covers the following as instructed

## Effective Addressing Modes:

Data Register Direct

Address Register Direct

Address Register Indirect

Immediate Data

Address Register Indirect with Post incrementing

Address Register Indirect with Pre decrementing

Absolute Long Address

Absolute Word Address

## Instructions:

MOVE, MOVEA, MOVEM

ADD, ADDA

SUB, SUBQ

MULS, DIVS

LEA

OR, ORI

NEG

EOR

LSR, LSL

ASR, ASL

ROL, ROR

BCLR

CMP, CMPI

Bcc (BCS, BGE, BLT, BVC)

BRA, JSR, RTS