T34 Emulator CSC317 Lucas Carpenter

Description

T34Emulator is a program that creates a memory array of 4096 (2^{12}) words where each word contains 24 bits. Memory can be read from and written to using the putWord and getWord functions, a memory dump can be executed using the memdump function and you can parse words of memory into their instructions using the parse function. The program also must read in an object file that contains memory addresses and their data to store, the last item in the object file will be the address the program counter will start at, this value is stored in a char* as a variable called 'pc'. To perform a memory dump, the function memdump(byte*) can be used, or to parse a string the function parse(byte*, std::string) can be used

Compiling & Usage

To compile the program, use g++ as follows:

```
$ g++ -o T34Emulator T34Emulator.cpp
```

To run the program, use the following line:

```
$ T34Emulator [objfile.obj]
```

Where [objfile.obj] is replaced with the file path for your .obj file

Function Definitions/Information

main(int, char[])*

- **Description:** Function is the starting point of our program; the program begins and ends here.
- **Inputs:** int - the number of arguments from command line
  char*[] - a 2D array of chars returning input from command line
- **Outputs:** returns 0 on successful program finish, physical output to terminal if invalid num of arguments.
- **Testing Methods:** Nothing really to test, this is where all the testing for other functions working was done.

readObj(char, byte*)*

- **Description:** Function takes in an array of characters representing an object filename to be opened and an array of bytes representing memory. The function reads through the object file and uses a specific object file format to store values into continuous memory.

- **Inputs:** char* - an array of characters representing an obj filename
  byte* - an array of bytes representing memory

- **Outputs:** no data output, physical output only if file can't be opened

- **Testing Methods:** I tested the function would read from the char* filepath by checking to see if the file was opened and read and output to the screen the contents of the file, and by seeing if the function would exit if the file path didn't exist. I tested the byte* memarray by outputting the contents of the array to the screen in the main function to see if everything matched.

putWord(byte, char*, char*, int=16)*

- **Description:** Function takes in an array of bytes acting as memory, an array of characters representing in hex a memory address, another array of characters representing a value in a base to store in the hex memory address, and an int defaulted to 16 to represent the base of the value being put into the hex address is needed to be converted from.

- **Inputs:** byte* - an array of bytes acting as memory
  char* - an array of chars representing a hex memory address
  char* - an array of chars representing a value in an unknown base
  int - an int, defaulted to 16, representing the previous char*'s base

- **Outputs:** no data output, no physical output

- **Testing Methods:** To test this function I would run this function in main and output the contents of memarray afterwards. I changed the base a few times to see if it would always convert properly and it appears to change bases correctly

getWord(byte, char*, char*, int=16)*

- **Description:** Function takes in an array of bytes acting as memory, an array of characters representing in hex a memory address and two ints representing input base and output base both defaulted to 16 and 10 respectively. The code converts the input base memory address to a decimal integer and navigates to the correct offset in the byte array. Then, the current byte and the next two bytes are OR'd together and shifted appropriately to create a 24bit word, this word is converted to the proper output base as a char* and returned as output.

- **Inputs:** byte* - an array of bytes acting as memory
  char* - an array of chars representing a hex memory address
  int - an input base to convert char* to decimal representation correctly
  int - an output base for to convert the output value to

- **Outputs:** char* - a character array in a specific base containing the 24bit value of a memory address

- **Testing Methods:** To test this function I would load in an obj file and pull the value from different memory addresses, if the values all matched what the input file had, then I assumed the function was working.

memdump(byte)*

- **Description:** Function takes in an array of bytes functioning as memory, the code loops through each memory address and outputs any addresses with a value not equal to 0.

- **Inputs:** byte* - an array of bytes acting as memory

- **Outputs:** no data output, physical output to terminal in form…
  [MEMADDR] - HEXVAL(6 hex digits)

- **Testing Methods:** To test this function I would populate memarray by reading in an obj file and then just call the function afterwards and see if the output matched what the obj file had

parse(byte, std::string)*

- **Description:** Function takes in an array of bytes functioning as static memory and a string of hex addresses separated by ','s or ' 's and splits the string into a list of strings, then each address is parsed and the output of each byte in memory is displayed as 3 bitstrings

- **Inputs:** byte* - an array of bytes acting as memory
  std::string - a string of memory addresses to parse

- **Outputs:** no data output, physical output to terminal in form…
  [MEMADDR]: ADDR(12bits) OP(6bits) AM(6bits)

- **Testing Methods:** To test this function I would populate the byte array by reading in an obj file then call the function using "", "0c5", "0c5 0c6", and "0c5,0c6 0c7", this let me see if my parsing of the string would work so you could chain multiple calls in one function call instead of in multiple separate calls and to see if the output matched the binary representation of the values in each memory address