Computer Architecture - Project II

Professor Billoo

Shyam Paidipati & Kevin Kerliu

Design Rationale

The following sections include an explanation and discussion of the fulfilled requirements of the project, the program architecture, and the challenges faced in completing the project.

Fulfillment of Requirements

Once the user compiles the source code using the Makefile, as explained in the README, the user can run the executable. Upon running the executable, the user will be prompted to input a name for the input file, which should at most contain one hundred 32-bit integers. The user is then prompted to input a name for the output file, which will have the sorted version of the input file data. Once the user inputs the names of both the input and output files, the program takes care of the rest, sorting the numbers and writing them to the output file.

Program Architecture

The source code begins with the data section, which is dedicated to 4-byte aligning all the prompts, inputs, outputs, addresses, and predefined data. The prompt for the user to enter the input file name, the prompt for to enter the output file name, the array of integers, the scanning and printing patterns, the file stream addresses, read and write modes, and error and return messages are all in this section.

The text section follows the data section and accounts for the bulk of the file. The code is divided into sections according to function as follows:

main:

Main takes care of logistics. Main is responsible for prompting the user for the input file name, output file name, and opening both the input file and output file.

file_loop:

File_loop is a branch that loops to itself and calls fscanf to read each line of the input file and copy that line to an array. The loop ends either because the input file contains more than one hundred integers or there are no more integers to read from the file.

error:

Error is a branch that is branched to in the case that file_loop ended because the input file contained more than one hundred integers. It closes both opened files, prints an error message to the command line, and exits the program.

sort_loop:

Sort_loop is a branch that sets up the rest of the sort algorithm by moving the necessary address and values into accessible registers. It sorts the integer array in order from least to greatest.

sort_loop2:

Sort_loop2 is a branch that checks if two adjacent integers in the array must be swapped, and if so, swaps them. If not, it branches to no_swap.

no_swap:

No_swap is a branch that handles the case for which no swap in necessary in the sort.

print_loop:

Print_loop is a branch that loops to itself and calls fprintf to read each element of the sorted array and copy it to the output file. The loop ends once it has copied every element of the array to the output file.

end:

End is a branch that is reached once the array is successfully sorted. It closes both opened files and exits the program.

The end of the program is followed by the labels used to access all the 4-byte aligned data. Moreover, external C standard library functions used in the program are listed.

Challenges

The challenges that we encountered included using the C standard library functions and keeping track of the addresses and pointers for their arguments. The functions we used from the C standard library were fopen, fclose, scanf, printf, fscanf, and fprintf. Using these built in functions allowed us to focus on writing our sort rather than worry about how to open and close files as well as read user inputs and write to files. Once we became comfortable with using these functions for interacting with the user and the input and output files, all that was left to do was sort the array of integers. Implementing bubble sort in ARMv8 was a fair challenge as it required keeping track of many registers as well as worrying about indexing.