ARM-11 FINAL REPORT

Darius Nilforooshan, Dominic Qu, Sam Tackaberry, Sachin Wadhwani

June 24, 2022

Structure of Assembler

As we were starting the assembler, we realised that there would be utility functions that we created throughout `emulator` which would be required for assembler, such as convertFormat. Therefore, shortly after we started, we thought it would be a smart decision to create a common folder, storing the utility functions which were used in both the emulator and assembler. Furthermore, we also grouped our files together keeping a folder for all the assembler files such as the `symbolTable` c file and header and for all the emulator files to make the code neater.

Implementation of Assembler

Looking at the spec, we decided that a two pass implementation would be easier to work on as a group. Our implementation worked as follows - the program would first add the predefined instruction symbols to the symbol table (a data structure which we would have to create). The first pass would populate the symbol table with labels and their corresponding addresses. The second pass would call the relevant assembly helper function (e.g. assembleSdt) on each line, which would substitute in the given tokenized symbols using the symbol table to obtain machine code. As well as this, file I/O had to be implemented, to read and write to a binary file.

We started with the general format of the program, creating the structures that we thought would be required as a foundation, such as a structure to store the instructions in the symbol table, a structure for the symbol table itself and how it would hold symbols, and the structure of how we would hold and access the lines of the file we were given. After this, we had to create the abstract data type for the symbol table. Reading the spec further, we considered what information would be required to be stored for each symbol in order to make debugging easier, such as the mnemonic of an instruction and the full instruction line as a string. We then created the symbol table and the essential add and get functions which we thought necessary for its function, before adding the predefined instruction symbols to it.

At this point, once all of us were caught up on the spec and our current progress, we split up the different parts of the project in order to progress faster. Sachin worked on the binary file reader, to read the assembly code given and put it into an array of file lines so that they could be easily accessed throughout the program, while Dom and Sam started working on the first pass. Darius looked at emulator, looking for any functions and utils that could be shared between `emulator` and `assembler`, creating the common folder and utils.

After this, Sam, Darius and Dom started on the second pass while Sachin started working on the binary file writer. While Dom worked on the structure of the second pass function itself and the creation of the instruction structure to be inputted into the assemble functions, Darius and Sam started on the actual `assembler` functions, working on assembleDp and assembleSdt, with Sachin working on assembleBranch and assembleMul. As we progressed, we realised we would require some utility functions, and Sachin and Sam then worked on creating those utility functions, such as functions extracting register numbers and operands from instructions, and hexadecimal conversion helpers. Throughout this process, we realised further what must be

passed into the symbol table in order for the assemble functions to return the correct binary, so we optimised our instruction structure to include the opcode. Once the structure of the second pass was finalised, we all worked on finishing the remaining `assembler` functions assembleBranch and assembleSdt.

After all the functions were written, we found that we had an issue with the binary file writer, and that it would print out outputs that were often not expected. After a meeting with our tutor, we found the error and fixed it, and started on the debugging process. After further debugging of the assembler to the point where most of the tests passed, we decided to split up, Sam and Sachin finishing the debugging of the assembler and including functions to free memory and Darius and Dom starting on the extension.

Here is the file structure that we decided on for our assembler project implementation

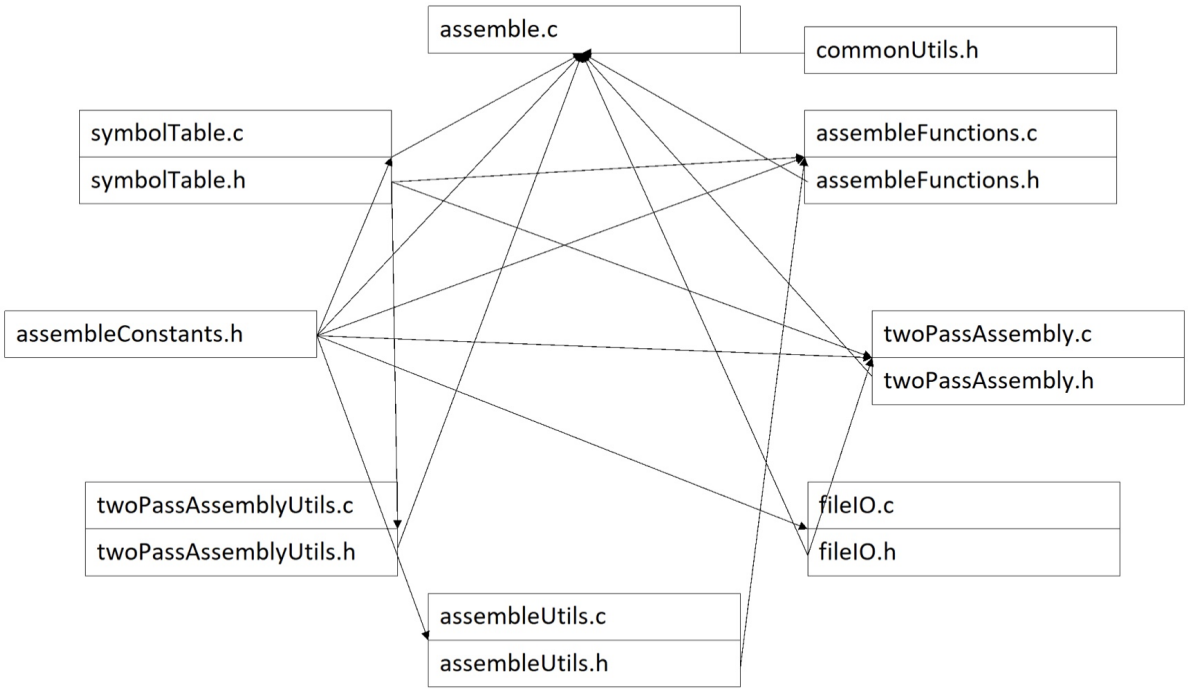

Figure 1: File structure of assemble

Extension Description

We took inspiration from this year's theme and decided to build a basic model of a food bank. Along with the fundamental data types and functions, we implemented a command line interface whereby a user can "enter" (log in to) our food bank as a staff member, donor or recipient. A staff member is able to view the inventory, inspect the donations that are on hold and also is the only type of user that can close the food bank. A donor is asked for their name and the details of the donation they wish to make, before their item is added to the inventory. If the inventory is full, the item is placed on hold. A recipient is first asked if they have an account with us; if they don't, we ask for their full name and provide them with a unique ID that they can use to log in to the store. We then attempt to allocate a box of food to them (which should last them roughly a week based on the gram requirements we have used) provided they haven't visited

the food bank within the last 5 days. If they have, we ask them to return in the appropriate number of days. Food allocation revolves around expiry dates and gram requirements. We allocate certain amounts of carbohydrates, protein and fruit and always allocate items with the earliest expiry date first. If there is not enough of a particular food type (e.g., we are only stocking three fruits when we want to give seven), then we give the recipient the remaining stock and apologise for the incomplete box of food. If a particular type of food does not exist at all, then we tell them that type is unavailable. In the case that the food bank is completely empty, we ask them to return another day.

Extension Implementation

The design of the food bank revolves around an inventory structure, which stores stocks of different food items as arrays and also the size of these arrays, along with the donations on hold. Each food item in an array has a unique ID (which each individual instance of the item also uses), the amount of this item available and an array of items consisting of the individual instances of this item. For example, in the carbohydrate stock there may be a food item 'bread' with ID 530 and an integer showing that there are 4 loaves of bread available. The food item will then have an array of individual items of bread, each with their own expiry date and nutritional information and ID 530. These individual items are what a user can donate to the food bank. If a user donates an item that the food bank has never stocked before, then a new food item is made and placed in the appropriate food stock array based on its category (carbohydrate, protein or fruit) and the individual item is placed in this food items array of individual products. With this design, we are able to maintain a 'menu' of products and as many individual instances of each product as we like. In addition, we are able to add new products to this 'menu.'

The command line interface is implemented in the main function. It begins by welcoming the user and asking them if they are a member of staff, donor or recipient. Then, using a while loop, it follows the commands of a user successively until they are finished with the food bank, at which point it begins its line of questioning from the start. We also included the option to input 'exit' repeatedly which will essentially retrace your steps backwards towards the 'home screen,' eventually terminating the program.

We knew our extension was ambitious and thus were prepared for the numerous challenges that it produced. The Makefile presented issues from the beginning, since we were not totally sure on the file structure we wanted to use and thus it took some time and collaboration to get our files linking successfully. Writing appropriate data structures was much harder than designing structures for the emulator and assembler since we had no guidance this time. It took much discussion to come to an agreement over how to store the different stocks of food items and how to implement a structure to represent an individual item. Even after coming to an agreement, the implementation was changed several times as we made further progress in the project, as we would gradually come to realisations about our requirements. For instance, we had to change the implementation of an individual item after we realised we needed two IDs as opposed to one. One ID would be common amongst all items of the same product (e.g., all bread items share the same food id) and one ID would be unique in order to allow for identification of an individual product when removing it from the food bank. Designing our command line interface presented some challenges too, as one of our main targets was to provide a user-friendly experience at the command line. This required us to take great care when writing the main so we decided to write separate UI functions for staff, donors and recipients to keep our main (and the print statements within it) neat and tidy.

Here is the file structure that we decided on for our assembler project implementation

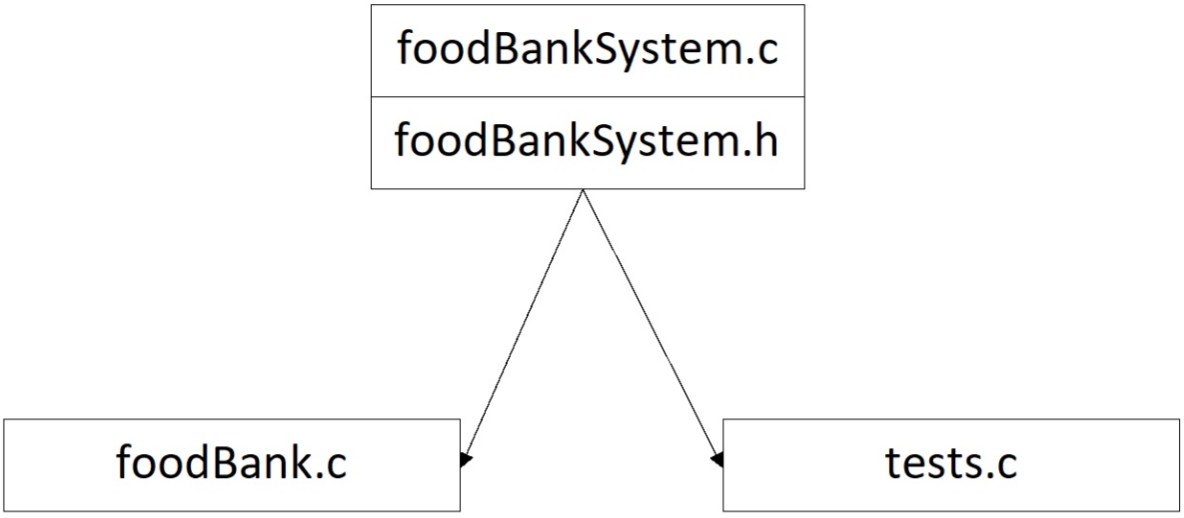

Figure 2: File structure of foodBank extension

Testing of Extension Implementation

We tested our food bank model with user-perspective testing and automated unit tests. To test outputs to the console, we repeatedly gave different inputs to the command line interface, debugging our code appropriately. We tested several edge cases, such as trying to give a person food when the food bank is empty. Moreover, to test our structures and use of memory, we wrote extensive unit tests which tested every aspect of our code and we made sure to pass all of them before expanding our program. For example, during early testing stages, the outputs in the console were correct but the program was not actually storing (or removing) the correct item from the inventory. Unit tests allowed us to test the contents of the inventory effectively and guided us to correctly implement functions that altered stocks of food.

Group Reflection

Overall, as a group, we were well organised and on top of our deadlines. We allocated work well and knew each other's strengths and weaknesses, working around them to best make the use of each other's abilities.

Approaching the C deadline, we were on track to submit on time, having completed the emulator only with a few bugs to fix, and having already written up the checkpoint report. We therefore decided to take a small hiatus from the project for a few days in preparation for the C test, so that no one felt pressured to work on the project and risk their individual grade. As such, we did not have issues with group members not being able to complete their group allocated responsibilities due to spending too much time revising for the C test.

We worked primarily together by meeting up either in labs or on MS Teams to work using CLion's 'Code with Me' feature. This way, we were mostly in communication while coding, meaning that we could explain to each other aspects which were causing trouble, and could avoid writing redundant code. We often abstracted our work when dividing it up, as not every member had to be involved with every part of the program (only the relevant information for the parts they were working on). // The person who wrote a specific block of

code provided explanations to group members for relevant parts of code or the spec where it was required for the group member to continue.

An issue that we experienced was that we all were inexperienced with using git for group project work. This led to confusion when trying to work on multiple parts of the project at the same time and committing such changes, since we were uncomfortable using branches and merging. This led us to work more at the same time, as opposed to working in our own time and merging the code together. This, however, allowed us to understand each other's code better as well as the bigger picture more, so our individual code could be written faster and more optimally, however it slowed us down at times, as it did not allow us to work to our individual schedules. At times, one or more of us were unavailable at times of the day when others agreed to meet, and they would be unsure of what needed to be done when doing work individually.

For future projects, one thing we could have improved on was creating a logical structure and plan for our code beforehand, and mapping things out to see the bigger picture. In the long run, this would have saved us a lot of time spent on optimising our code, and on re-configuring code or functions which we found to not fit their desired purpose, or functions which we found to be redundant.

Individual Reflections

Dominic

Given I have never worked in a group to create a program before, I was unsure how group work would go. My understanding of git and how to properly use it within a group setting was limited, however improved a lot over the course of the project, and I now understand the importance of, for instance, a meaningful, detailed commit message and branches. As well as this, I have been able to learn how to use latex, and understand its numerous use cases. I now feel comfortable formatting documents in numerous ways with latex as well as using libraries, for instance to create diagrams for a PDF.

I felt that my strength within the group was that I could see the bigger picture, and so could understand the spec, plan out the foundations of the code and create structures in a way that would make it easier for the entire group. Furthermore, my understanding of code hygiene meant that I could look through the code and optimise it.

My limited understanding of git was a weakness, and I was sometimes unable to merge successfully, having to ask my group members for help or asking them to push on my behalf, slowing down group progress. With a different group of people, I would have done more to do with git and branches, and relied less on in person meetings and liveshare code features. Furthermore, I was unfamiliar with debugging tools such as the GDB debugger or the CLion inbuilt debugger, which meant that I was of less use when fixing the code to pass tests, however throughout the project this improved a lot. Overall, I believe that I, along with all of the group, pushed my weight and worked to my strengths throughout the project.

Darius

This project was the most ambitious programming task I had embarked on as a group and thus the start was understandably daunting. But as we worked through the specification and our extension, I felt increasingly integrated into my team and we managed to work fast, but effectively. During this project, I believe one of my biggest strengths that came to light was my creativity in solving complex problems, as I was able to provide innovative solutions in the form of short, concise functions for our emulator and assembler. In addition, I expected my grasp over manual memory management in C to prove useful, and it did help us when writing functions that changed the values of inputted parameters. Before starting the project, I thought my experience with Valgrind would help us remove memory leaks, but in reality the errors were sometimes spread across multiple helper functions and so large in quantity that Valgrind's messages did not prove very

helpful as errors had multiple origins. A concept I felt I initially struggled with was masking and shifting bits as it was a relatively new concept to me in C, although I was able to pick this up quickly thanks to help from my fellow group members. In future group projects, I would definitely aim to test my code more often in order to not have to deal with huge amounts of error messages in one attempt.

Sam

As my first time working in a group project, I was apprehensive about the outcome of such a task but, I feel we were all able to strongly communicate with each other and rely upon the other group members to effectively complete their individual tasks. Going into the project, I felt my debugging skills were a personal strength of mine and that they could be well utilised during this project. I believed that I could efficiently use GDB and the inbuilt debugging tools to improve our code. This was especially helpful when passing the test cases as it allowed us to find our errors quicker and successfully fix them. However, I didn't have a lot of experience with using Makefiles coming into the project so felt I had to rely on my group members to help me. By the end of the project I was more comfortable with doing it myself and didn't need to rely on them anymore. As a group we did the majority of our work on campus or using LiveShare so we were constantly with each other. With another group, I believe I would have worked more independently and made better use of Git Branches and other features, however if possible I would have used the same method as this time as it ensured everybody did their work and our vision stayed aligned with the same end goal.

Sachin

I feel we worked quite well as a group: everyone was working on one of the smaller tasks we had to complete at any given point in time. We also had a similar programming ability, so when debugging, it was easy to work together to solve problems as we all understood each other. I felt quite confident about using gdb going into this project, and I found using it very useful throughout the project in terms of saving time in finding errors. I knew going into this project that I didn't have much experience in using Git for group work, especially in terms of using branches, and we decided as a group that we were more comfortable using Live Share features to coordinate our programming, but I might have tried using branches if working with another group of people. Working remotely over Microsoft Teams is something I would like to be able to do when working in groups in the future, as it can save group members time, and make otherwise short meetings owing to busy schedules more productive.