C final report

Group 38

Tian Zhili, Qian Xingzhi, Chen Yeye, Zhu Boan

1 Assembler Structure:

Our assembler has a similar structure as the emulator, which is shown in the figure1 in the last page.

Taking loop01.s as an typical example, it should be loaded in to a “loaded file” line by line. Then according to the definition of two-pass assembly, a “symbol table” which has property of hash map is built for the 1st-pass. All the labels with corresponding addresses are put into the table and deleted from the “load file”. For the second pass, the rest of instructions are loaded and tokenized into a “tokenized word array”, which can be considered as an array of array of the words. After that, the “symbol table” and “word array” would be. Combined together, executed to output the correct binary instructions. These binary codes would be first put into a “instruction array” and then written to the object file loop01.bin.

For the execution part, instructions are assembled differently according to their instruction types. Notice that the branch instruction needs to use “symbol table” as extra input parameter and single data transfer instruction may write on the “output instruction array” directly. Different instructions are created separately, one of benefits for doing that is that we can create other instructions without rewrite the whole execution part.

2 Extension description:

Further arm11 instructions:

Multiply-Accumulate Long (MULL, MLAL), Single Data Swap (SWP, SWPB), The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations. The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two’s-complement signed numbers and write a two’s- complement signed 64 bit result.
Ex.

UMULL R1,R4,R2,R3 ; R4,R1:=R2*R3
UMLALS R1,R5,R2,R3 ; R5,R1:=R2*R3+R5(R1 also setting condition codes)

The single data transfer instructions perform a "locked" load and store instructions. SWP command should swap in base of a word (a whole register) while SWPB command only swap the last byte of registers.
Ex.
SWP R0, R1, [R2] ; R0 = [R2] [R2] = R1
SWPB R2, R3, [R4] ; R2 = [R4] [R4] = R3 (but only swap last byte)

The remaining data processing instructions perform the similar operation as the previous part. There are six instructions and the first step is to decode them. Then we need to check the immediate operand and shift the operand2 as a constant or a register. And finally set the "cpsr" flags and write the result.
Ex.
MOV R1,1 ; R1 = 1
ADC R2,R1,2 ; R2 = R1 + 2 + carry (carry is the carry bit)

3 Challenges of Extension:

There's no many challenges in the extension part, as we chose to extend the instruction set with more ARM instructions at the end. However, there are always problems even in a comparatively simple task. The first thing we met was that the original ARM11 Data Sheet had some implicit description of the instruction details. We had to seek for other related materials and spend time discussing about it. The biggest challenge is probably how we restructured the emulator and assembler. Since this extension was based on part1 and part2, the original structure had to be changed. By adding new instructions we should also guarantee that the provided test cases still pass. We also have problems in testing so we wrote some sample tests and print the results after execution to check if they are same.

4 Test of Extension:

We builded our extension inside our project (part I and part II) and expanded by adding three more instruction sets, so we can test based on our existing successful instructions given by the testsuite. To test our extension implementation, we created similar ARM source files and then test assembler part by comparing the output with different parts of correct binary forms from ARM data sheet as sources provided from Cate. If the result looks the same, then we did it successfully. To examine the emulator part, we generate the output and see if the values are stored correctly in the registers and memory. For assemble part, we believe this is feasible as every bit has fixed values according to different instructions . For emulator part, It may not be the most effective way to test since we do not know the correct results, but by understanding we can know from the output if we did it correctly or not.

5 Group reflection:

After learning the mistakes that we had on the part1, we settled the general structure and the code style at the beginning, and things are a lot smoother than part1. Even though we are all in the same time zone, we bumped into problems all the time. So the other thing that we thought would be useful, and it did, is that adding a ton of comments everywhere, so that it would be much easier for other group members to understand the code. After completing this part, the group leader combined all the useful comments and deleted the unnecessary ones. The way we communicate is pretty effective. From 7pm to 11pm we stayed in a call meeting in an app called "WeChat". When someone had questions, he could simply unmute and ask, and we would try our best to help. If no one had any questions, just stay mute and do our own work. Because we usually talk in the evening, we collect questions in the morning. If the problem is still not resolved, we would turn to the TA lab helpers. As for splitting the work, we tried to give everyone the same amount, but our leader usually has more things to do, like combining everyone's code and handling the GitLab. Next time when we do a group project, it is most likely to be in the labs and the most effective way of communication is always face-to-face. We would still schedule a certain time slot every day for everyone to work and communicate together. However, we did run into some problems which we would like to avoid next time. We initially didn't settle down the general structure and it wasted a lot of our time combining the codes together. We ended up having very little time planning and writing the extension. Next time we would possibly set our own deadlines for each task and leave some time at the end in case something goes wrong. Overall, we think this project is very beneficial and we all learned a lot on how to work as a group.

6 Individual reflection:

6.1-Qian Xingzhi:
Everyone has contributed a lot to the project. I helped build the overall structure of both emulator and assembler and created some typedef that used in the code. I did a lot of debugging during the project. I also converted what our report to latex form. Although the procedure of debugging was really annoying sometimes, seeing the correct results finally was achievable and made me feel proud. We often encouraged and helped each other when having difficulties. I like helping teammates and explaining things that they don't understand well. This time, all my teammates are Chinese, which means that we can use mandarin to discuss and we are in the same time zone. In the future, it may be a different experience to have foreign teammates. Since C is a completely new programming language for me, I can learn a lot from this project by searching related information online and using lab hours. When looking at what extensions our group should do, I did some research on different topics and found the most suitable one for our group. I still have some weakness. For example,

I am not good at explaining things that I wrote which might cause confusion sometimes. It is important for me to not only know what I did myself but also let others know how I structured the codes overall.

6.2-Chen Yeye

Everyone made a great effort trying to make this project perfect, as it is the last big "coursework" at the end of this year. We appreciate and respect each other and learn a lot from each other. The good thing is that we are in the same time zone, and it made thing a lot easier. I don't consider myself as a leader or anything, I just finished what I am told to and try to do it as perfect as I can. If I have to state one thing that I am good at in this group, that would be finding the best solution to a certain problem from all range of resources. At the beginning, we had no direction, and we didn't know where to go. Fortunately, our leader made some good calls on the structure and code styles when we were all miserable. I think if I could give some of my suggestions or ideas, it would be smoother. Therefore, if I am going to work with a different group of people, I would speak up a little bit more and share my thoughts. Another weakness that I have is the knowledge of using git, every time I need to refer to my leader to get help. I think this project really gave us an opportunity to improve working with a group and improve our programming skills.

6.3-Zhu Boan:

I was responsible for building the data processing execution in both emulator and assembler and build the execute part in the emulator. I also completed the makefile and did much debugging thing. Every group member was actively involved in building the entire framework and structure. We meet every day and each of us participated in the whole project. I think my strengths are that I can understand well what we need to do and finish it with a fast speed. On the contrary, my weakness is that I have many problems in code style like duplicate code and too many help functions which I spent a lot of time to fixed them in a better way. Although because of the COVID-19, we can only meet and discuss online, our group member encouraged each other and had provided the most help for others. So when someone had problems, we can solve them very efficiently and quickly which made us overcome many difficulties. All in all, I learned a lot from this project and have stronger programming skills and team skills.

6.4-Tian Zhili:

It's my first experience to be a group leader in such a project and it's true that a leader means a lots. Fairly speaking, I do have several qualities which make me a qualified leader. For example, I was more willing to communicate, had better organizational skills and (maybe) had better comprehension of ARM11 instruction set. Most important was that whenever there's a problem nobody could fix, I never hesitated or felt shame to ask people in other groups or post questions on piazza, which guaranteed every single part of project work effectively. I also realized that a project was not just about to be finished, but to be done well. Therefore, I briefly went through all the files, learned the codes, deleted some repeated defined functions and restructed the project to make to easier to extend further. My group members and I also spent time on unifying the code style and comment style. However, no man is flawless, I did not manage the time well which made everyone rush to meet the deadline. I did not understand C language very well which wasted tone of time on defining type of data structures and debugging. If I were working on the next project, I would definitely to say “well begun is half done”. When planning a project, a flow chart is necessary to have. I would also suggest to write the header files first that in case if others wanted to use your functions.

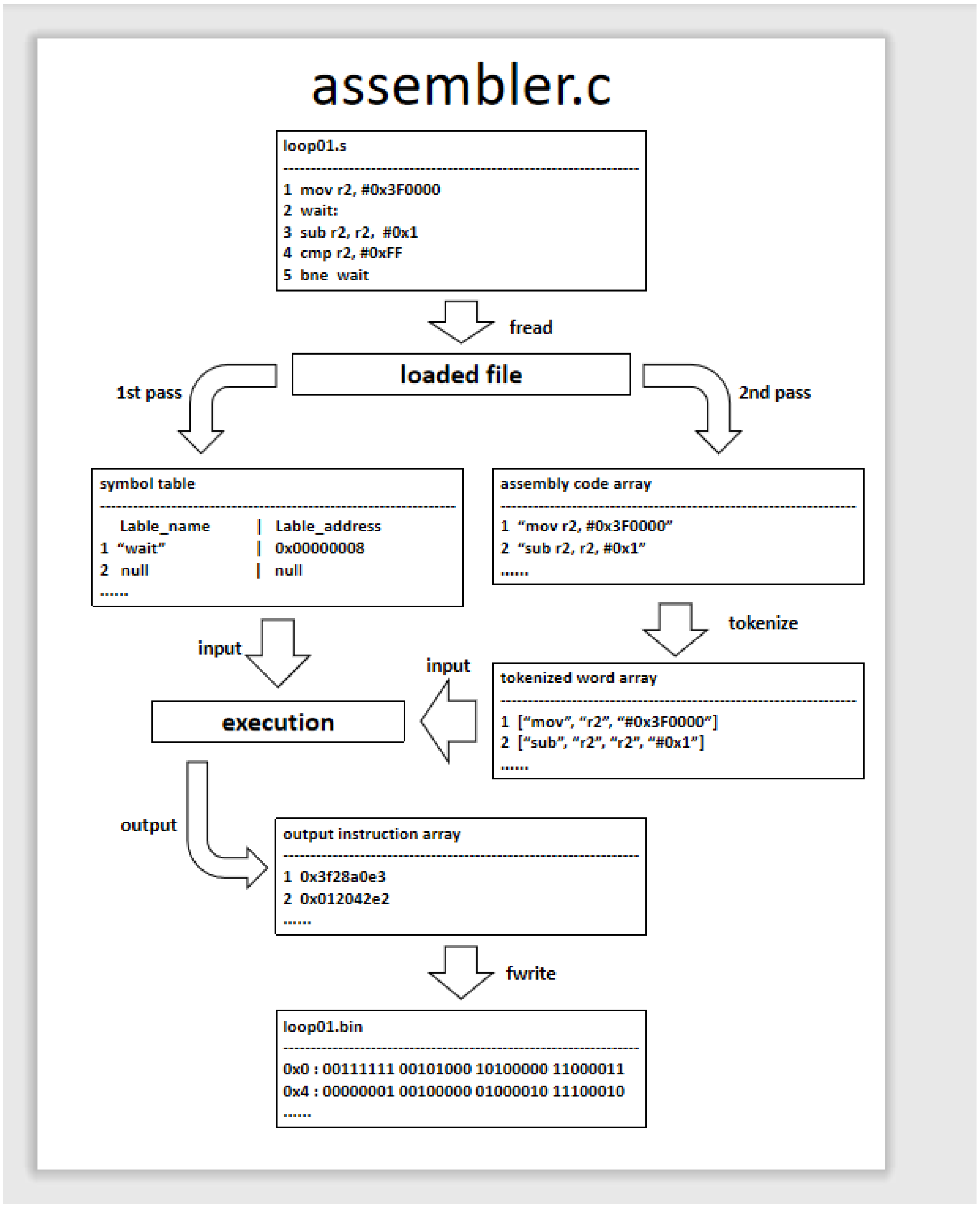

Figure 1: Assembler Structure