Computer Architecture 2024-2025
University of Groningen
Homework Assignment 7

by Vitalii Sikorski

January 29, 2024

Program Description

The goal of this homework is to apply the acquired assembly concepts and problem-solving abilities to break down and execute a more complex assembly assignment. The end result of this work is a compiler for an unconventional programming language known as Brainfry, which bears similarities to another language with a less appropriate name.

The implementation of the project began with breaking down the bigger task into smaller ones.

1. **Getting the user input**
   The input includes Brainfry Instructions, comments and random characters, thus the first step was to ignore the characters which aren't instructions and to store the commands in contiguous memory (GET_INPUT Subroutine)

2. **Processing the input**
   After storing the input, the program then executes the instructions one by one, interpreting them to LC3 instructions and handling the loops/comments logic (HANDLE_LOOP_ENTER and HANDLE_LOOP_EXIT subroutines)

1 Getting the user input

```
CHECK_RIGHT
    LD R5 MOVE_RIGHT_NEG ; checking if the input is move right sign
    ADD R5 R0 R5
    BRnp CHECK_LEFT
    ADD R5 R5 #1 ; Instruction #1 for Move Tape Right
    STR R5 R1 #0 ; Store #1 in the INS register
    ADD R1 R1 #1 ; Increment the INS register
    BRnzp READ_INPUT
```

The implementation of this section of the program is straight forward, we receive input from the user as ASCII decimal values and we're checking if the input corresponds to the value of an instruction by subtracting the negative value of that instruction from the input. The program then stores that instruction in memory (I used #1 to #9 to represent instructions for easier implementation of this process backwards) and increments the pointer which indicates at the address the instruction should be saved in.

2 Processing the input

```
PROCESS_INS
       LDR R3 R2 #0 ; Load R3 with the value of the INS
       ADD R6 R3 #-1
       BRz PROCESS_MOVE_RIGHT
```

The program part which is responsible for processing the instructions, begins by determining which instruction it is supposed to implement, because I've used a decimal representation 0..9 to represent them, this is done easily, by continuously subtracting 1 from the value and this way determining the branching accordingly.

```
 PROCESS_INC
       LDR R0 R4 #0 ; Loading the value of R4 into R0
       ADD R0 R0 #1 ; Incrementing the R0
       STR R0 R4 #0 ; Storing the value of R0 into the R4
       ADD R2 R2 #1 ; Incrementing to the next instruction
       BRnzp PROCESS_INS
```

Instructions that move the tape or increment/decrement are done in the same subroutine, to avoid excessive jumping and memory loading.

```
LOOP_ENTER_STACK
       LD R0 STACK_COUNTER ; Loading R0 with the stack counter
       ADD R0 R0 #1 ; Incrementing R0 to the next empty space
       LEA R3 STACK ; Get the address of the beginning of the stack
       ADD R3 R3 R0 ; Get the address of the first empty stack position
       STR R2 R3 #0 ; Storing the INS address in the stack
       ST R0 STACK_COUNTER ; Storing the Stack Counter in memory
```

The main logic lays in the handling of the opening and closing brackets as these can be either comments (which may include inner comments) or loops (which may include inner loops), for this reason I've used a simple counter to determine the number of opening brackets in case it is a comment and decrement its value when a closing bracket is encountered, this way, when the counter balances/becomes 0, the program knows the comment section has ended and can exit the subroutine and increment the instruction pointer.
In the case it is a loop, I've implemented a simple stack pushes the the address of the opening brackets to it and when a closing bracket is encountered and it is the case that it needs to loop back, the next instruction of the INS is programmed to be the first one from the stack.

3 Conclusion

The program I've developed proves to implement the logic correctly and passes 9/10 cases on Themis, the reason in doesn't pass test case 7 seems to be related to the fact that it is too slow. I have tried to reduce the number of instructions that are being stored by grouping multiple increment/decrement/move instructions into one but this didn't fix much. I believe a better implementation is one which uses less memory access instructions as these are most costly, I assume this can be done by using registers as temporary storage for handling the values instead of storing the values in memory at every step. Nonetheless, I found this project very informative and one that boosted my understanding of LC3.