Ahmet Eren Aslan, Umut Şendağ CMPE 230 Assignment 2 Documentation

The project translates a postfix expression to its corresponding RISC-V machine code. When the program is run it waits for input. In this state the user should give a postfix expression such as " 2 3 + 4 5 + *" to the input. The implementation details are explained below.

The GNU Assembly code for postfix translator starts with taking the input from the console. Then it sets rcx, r13 and r14 to zero which will be explained later. Then the program checks what the input character is.

- If the character is a new line character it means the postfix expression is translated and it ends the program.
- If the character is a space the program checks what is in the register r14 which is used to determine if the current space character is after a digit or an operator. This is done by checking the register r14, which is 0 if the last seen character is a digit and 1 if the last seen character is an operator. If it is after a digit, it means the current number is evaluated so it gets pushed to stack. After that the input pointer r8 gets incremented, and the register rcx which is used to store the current decimal number is reset to 0 and the program returns to check the next character. If the space character is after an operator the program it only resets rcx, increments r8 and returns to check character.
-If the character is one of the operations "+", "-", "*", "^", "&" or "|"; an item is popped from the stack into r10 and r11 which will be used as a counter to convert the popped number to a 12-bit binary number. If the number is positive, firstly r11 is checked to make sure the number is converted. If the number is converted it gets printed and then the rest of the row is printed according to which row we are in with the help of r13. If the number is not converted, it gets shifted right one bit and the carry bit is added to the output buffer. Then it checks the next digit. If the number is negative 4096 which is 2^{12} is added to it which gives us the 2's complement of the number. Then the same operation is done to this number as if it is a positive.

This operation is done two times. The first time it is done the number is moved to the register r15, the second time it is in the register r10. After the numbers are converted and the corresponding output is given, the actual operation is done and the result is pushed into the stack. This is followed by the corresponding opcode and the rest of the row. After each print operation the pointer to the output buffer gets incremented accoırdingly.

- If the character is none of the things that are mentioned, it is a digit. In this case the the register r14 is changed so that it tells us that the last seen character is a digit. Then the ascii value of 0 is subtracted from the input character so that it gives us the decimal value of that character. Then we multiply the current number register which

is rcx and add the input number to it. For example if the number 123 will be written rcx will start at 0 and after reading 1 from the input it will do:

rcx = 10*rcx + 1

which will update rcx into 1. After the is read it will do the same operation and update it into 12 and then 123. As said earlier if a space is encountered after a digit this rcx will be pushed to the stack.

Example inputs and outputs for the code:

Input 1:

2 3 + 4 5 + *

Output 1:

000000000011 00000 000 00010 0010011
000000000010 00000 000 00001 0010011
0000000 00010 00001 000 00001 0110011
000000000101 00000 000 00010 0010011
000000000100 00000 000 00001 0010011
0000000 00010 00001 000 00001 0110011
000000001001 00000 000 00010 0010011
000000000101 00000 000 00001 0010011
0000001 00010 00001 000 00001 0110011

Input 2:

47 36 - 13 57 1 | + ^

Output 2:

000000100100 00000 000 00010 0010011
000000101111 00000 000 00001 0010011
0100000 00010 00001 000 00001 0110011
000000000001 00000 000 00010 0010011
000000111001 00000 000 00001 0010011
0000110 00010 00001 000 00001 0110011
000000111001 00000 000 00010 0010011
000000001101 00000 000 00001 0010011
0000000 00010 00001 000 00001 0110011
000001000110 00000 000 00010 0010011
000000001011 00000 000 00001 0010011
0000100 00010 00001 000 00001 0110011