String Concatenation in ARMv8 Assembly

ECE-251 Project 1

Gavri Kepets

Gregory Presser

Moses Bakst

Lola Staff

Program Requirements

The purpose of this program is to accept two user inputted strings and concatenate them. Upon running the program the user is asked to input the first string, hit enter, input the second string, and hit enter. If there are no errors the concatenated string is output to stdout and the error code will have the value of the number of characters in the concatenated string.

Both strings are limited to twelve ASCII characters. If the number of characters in the first string exceeds twelve an error message will be printed, the program will be exited and the error code will be `7`. If the number of characters of the second string is over twelve, the error message will print, the program will exit, and the return code will be `8`.

Makefile

The Makefile is not necessary to run an Assembly program but it allows the process to be more efficient.

The Makefile has two functions:

When the user runs `make` the program is compiled into concat.out.

When the user runs `make clean` concat.out is cleared.

Instructions

To compile the program, run `make`.

To clear all compiled binaries, run `make clean`.

To run the program, run `./concat.out`.

Main.S

Variable Declaration

.data section (1-22)

The .data section is used to declare variables that are used later in the program.

`scanf_msg` `scanpattern` `err_msg` & `print_msg` are all used to display strings to the user. The `%s` format character is used to insert the user's string into the prewritten message.

`user_input` & `output` are used to store the user input and the program output, respectively. While the program will throw an error after either string is longer than 12 characters, `user_input` is allocated 100 bytes using the `.skip` keyword. If a user attempts to input a string longer than 100 characters, the program will exit without an error message. `output` is only allocated 24 bytes since the longest the output could be is 2 strings of 12 characters concatenated.

`return` is initialized as a `word` and will be used to store the original return link register for the function call

Addresses Section (86-92)

This section is paired with the .data section as it stores the addresses of the variables declared there. This allows us to reference the variables declared in the .data section in the .text section.

Externals Section (95-97)

This section adds the scanf and printf functions from C, which were used to access stdout and stdin respectively.

Program initialization: main (24-31)

The .text section begins with main being added as a global. This is so the compiler knows where the program begins.

The final return pointer is stored in a dedicated return variable.

R5 and R6 are both initialized with a 0 and will be used later for counting.

R4 is allocated as the user's output pointer.

Input: ask_for_string (33-43)

This section accepts the string input from the user.

The R5 register is used to keep track of the string the user is currently inputting.

The scanf_msg is loaded into R0 and then printed.

The scanf function then runs and the address of the input is stored in R0.

String Length

Length Checker: check_len (45-55)

This function runs in a loop to check the length of the string.

A letter from the string is loaded into R2 and it is compared to null. If it is null we exit the loop.

Otherwise the character is stored in the output R4 at the location of R6.

R6, the combined length is incremented by 1. R1, the current string length is incremented by 1.

If R1 is equal to 13 we go to the error branch.

The check_len branch is called again and will continue to be called until the string is over or an error is thrown.

Error Branch: err , update_error (57-67)

This branch runs on error when the string is longer than 12 characters.

The error message is loaded into R0 and the message is printed.

By default, the error code 7 is loaded into R0. The value of R5 is checked to see which string caused the error. If the value of R5 is not 1, the second string caused the error. The update_error branch updates the error code to be 8 in this case.

Second String: continue_check (69-72)

R5 indicates which string was just inputted. If R5 is 1, string 2 was not inputted yet so we branch back to ask_for_string. Otherwise we continue to the end of the program.

Program Termination

String output: second (74-78)

The print message and the concatenated string are loaded into R0 and R1 and printed.
The length of the final string (R6) is stored in R0 as the error code.

final (80-83)

The length of the final string is stored in R0 as the error code.
The return address is loaded into the LR register and the program is exited.

Challenges

While developing this project, there were a few major challenges and design choices. The first design choice was choosing to use scanf and printf. These functions can be made in assembly, but would be incredibly tedious to do by hand. The first major challenge we encountered was getting the scanf function to work. The reason this was so difficult was that the scanf function leaves a newline in the input buffer after it is called, and a specific scanpattern had to be specified in order to ignore it. Another challenge was figuring out how to concatenate the string. At first, we checked the length of each string to see if it was valid, and then concatenated them, but we found that it was way more efficient and clever to add the letters as we checked the length, so that we only need to loop through each string once.