# Assembly vs C

## Assembly Program Overview

The entry point of the program on line 5 tells the linker where to begin the program. From here, the values of the rcx are reset and the values of registers r12-r15 are set to zero. Within the main loop there is three conditions to check whether a value is divisible by 3, 5 or by 15 and in turn they determine what string or number to print. Compared to the C program, these carry the functions of a conditional statement.

The register rdx is reset each time and the contents of the rcx are moved to the register rax. The value 5 is moved to the register rbx and then the contents of rax are divided by the rbx on line 17, 25 and 33. This is compared to the value of zero and if it is equal to zero the program jumps to the corresponding section of code. Following this, the string is printed along with a new line and the registers r14 and r15 are incremented by 1 if the rcx was divisible by 3, 5 or both. If the contents of the rcx was not divisible by 3 and/or 5, then the contents of the rcx is printed.

Once the rcx has reached 21, the program will print the number of fizzbang and then terminate the process.

Every time the rcx, or rax is pushed the value of the register gets pushed to the stack. This is so the values are preserved because the registers are used so frequently for specific tasks.

## C Program Overview

The C program is a very basic program and is less complex than the assembly code. The code declares two variables fizz and bang to store the number they are printed. Next, a loop counts from 1-20, testing whether the index is divisible by 3, 5 or both using the modulus operator. Dependant on the tests, the fizz and bang counter are incremented, and the correct string is printed. Finally, the number of fizzes and bangs are printed, and the program exits on line 27.

Converting the C program to its assembly version, it takes an extra 17 instructions to test whether the index is divisible by 15 compared to the original assembly version. This instance shows how the C program uses more instructions to complete a task and it is using a greater range of registers. However, this is not always the case as C uses more complex instructions to reduce the number of instructions to complete a task and optimise performance.

## Memory Usage & Process Time

The assembly program spent 0.017s of the CPU’s time in kernel mode, 0.053s in user mode and 0.069s from the moment it was executed to the end of the program. The memory usage is as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Text | Data | Bss | Dec | Hex |
| 434 | 36 | 0 | 470 | 1d6 |

The C program spent 0.030s of the CPU’s time in kernel mode, 0.094s in user mode and 0.303s from the moment the program was executed to the end of the program. The memory usage is as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Text | Data | Bss | Dec | Hex |
| 1813 | 608 | 8 | 2429 | 97d |

Dec and hex give the total size in the given format whereas the first three – text, data and bss give them individually. Initially, we can see the C program uses almost five times the amount of memory and in terms of real-time it took longer to execute. Additionally, the C program uses 16 times more data to complete the task, each one being assigned to a frame on the stack from the method they were called from. The C program declares 8 static and global variables that are uninitialized and set to zero (bss). The disadvantage of this compared to the assembly code is that more resources are being used despite the fact the variables are uninitialized or set to zero.

C compiles the program and during this process, it optimises the program for performance, but it is not always as efficient as a well-written, assembly program. However, in most cases C will perform equally as well as the assembly program. The biggest difference between the two is the memory. The C compiler optimises the code, and this will use more instructions and data than the assembly code, yet the C program still compiles in a similar amount of time.

## Register use

Both programs use relative addressing but for different reasons. The assembly program uses relative addressing to convert the decimal into its hex equivalent, whereas the C program uses relative addressing to move an address for a string to be printed.

Both programs store the index value in the register rcx and this increments from 1-20. They both also make use of the stack and push and pop the value of the rcx to the stack.

To print the string, the assembly program will set the source location of the string (rsi), the length of the string (rdx) and the command to write (rax set to 1). This is followed by a system call to invoke the kernel. Within the C program, it uses the instruction ‘printf@plt’ which is a built-in function to speed up to process.

## Overall

The advantage of using assembly language is you have complete control over memory and the interrupts; however, the syntax and instructions are more complex to program than they are within the C program. In this scenario, the assembly program used significantly less memory and CPU time to complete the task. The main disadvantage is it is not optimised for a specific CPU compared to a C compiled program.

Compared to assembly, C is simpler to program and is easier to maintain and when it comes down to processing speed it is not far of a well-written assembly program. However, C uses more memory due to optimisation.

## Bibliography

Guest, A. (2019) *Architecture & Assembly* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [4thNovember].

Guest, A. (2019) *More Assembly* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [11thNovember].

TutorialPoint. (no date) *Assembly Programming Tutorial* [Internet]. Available from https://www.tutorialspoint.com/assembly\_programming/index.htm [Accessed 10th November 2019]