INF-2200

Oblig 1

This assignment was an introduction to coding in assembly. The task was to implement a demanding process in C, find the most demanding part and then convert that part into assembly. Doing this would expose the process of what a compiler must judge and write in order for for the code to work. Although the compiler uses algorithms to complete the interpretation it is not perfect, that means that focused and optimized assembly code can at most times be better than what a compiler spits out and thus a subgoal is revealed. Optimize the assembly code so that it beats the compiler/interpreter.

Seeing as this assignment requires C code to work understanding said programming language is a must. One must also understand the underlying process for the different operands(if, for, while, etc.) in order to decompose them into writable assembly language. Since there also was a requirement for a specific type of assembly language, namely IA32 AT&T. Understanding the differences that entails must also be considered. The assignment was completed using the Sieve of Eratosthenes, an algorithm that weeds out the prime numbers in ascending fashion. It evaluates for the lowest integer that is a prime, marks all multiples of that integer as non-primes, then moves on to the next prime integer.

The sieve itself is only 4 lines of code, so it’s easily the most demanding part of the code therefore the whole 4 lines were chosen to be translated into assembly. The structure of the sieve is as follows

For loop

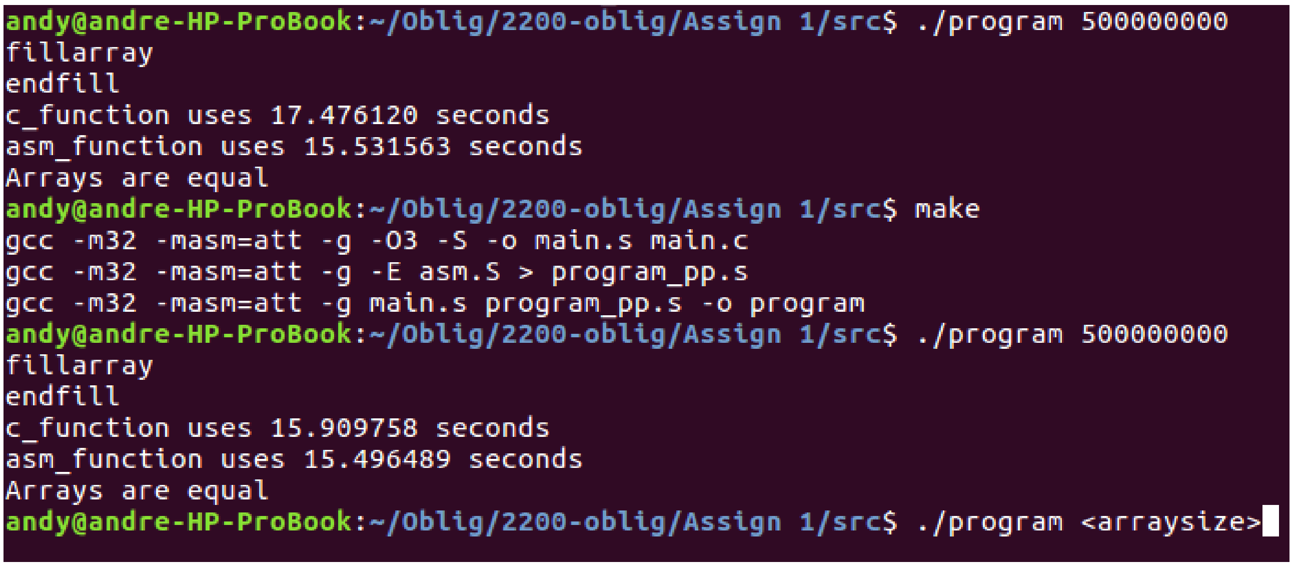
If test

For loop

Array manipulation

and as such makes it rather painless to translate the structure without losing way in the translation process.

Seeing as a compiler uses defined algorithms to complete certain functions it might be very under optimized and therefore rather slow in code execution. As that is the case writing the assembly specifically for this program it's able to beat even the strictest of optimization rules the GNU compiler had to offer. It might be a fluke but several runs have coloured the opinion that it is in fact the case



```
andy@andre-HP-ProBook:~/Oblig/2200-oblig/Assign 1/src$ ./program 500000000
fillarray
endfill
c_function uses 17.476120 seconds
asm_function uses 15.531563 seconds
Arrays are equal
andy@andre-HP-ProBook:~/Oblig/2200-oblig/Assign 1/src$ make
gcc -m32 -masm=att -g -O3 -S -o main.s main.c
gcc -m32 -masm=att -g -E asm.S > program_pp.s
gcc -m32 -masm=att -g main.s program_pp.s -o program
andy@andre-HP-ProBook:~/Oblig/2200-oblig/Assign 1/src$ ./program 500000000
fillarray
endfill
c_function uses 15.909758 seconds
asm_function uses 15.496489 seconds
Arrays are equal
andy@andre-HP-ProBook:~/Oblig/2200-oblig/Assign 1/src$ ./program <arraysize>
```

In conclusion the assignment was completed and the assembly code runs without hitch. Giving knowledge of how to correctly parse C code into assembly oneself and giving experience in doing so. Had a different algorithm than the sieve been chosen the assignment might not have been completed in time.