Inline Assembly in C

Rishi Dua, 2010EE50557

February 20, 2014

1 ASSEMBLY CODE IN C

1.1 PROBLEM STATEMENT

1. Given two integers, write a time efficient c code, that spends as less time in memory access and more in calculations as possible, to get their GCD and LCM. Use directives for conditional compilation of code as follows.
   Identifier Status: 1 -> GCD
   Identifier Status: 0 -> LCM
   Find the critical parts of code that consume more percentage of time and replace those parts with inline assembly coding to optimize speed and compare the time profiling for both codes.

2. Find the most significant non zero bit position for largest of the two given integers using pure c code and inline assembled c code and show which one is faster.

3. Convert the given integer(assumed to be angle in degrees) to radians and find the floating cosine value with pure c and inline assembled c and determine fastest one.

1.2 ABSTRACT

All parts have been implemented.

In mathematics, the greatest common divisor (gcd), also known as the greatest common factor (gcf), or highest common factor (hcf), of two or more integers (at least one of which is not zero), is the largest positive integer that divides the numbers without a remainder. For

example, the GCD of 8 and 12 is 4.

An assembly language is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple architectures, but require interpreting or compiling.

Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

1.3 Specification And Assumptions

Tool Specifications:
Platform: Ubuntu 12.04
Bash Version: GNU bash, version 4.2.25(1)-release (x86_64-pc-linux-gnu)

Problem specifications: Assume the following utilities are present on the system
C compliler

Assumptions
The time required to read input from file is negligible

1.4 Logic Implementation

GCD and LCM

We use the Euclidean algorithm, which uses a division algorithm such as long division in combination with the observation that the gcd of two numbers also divides their difference. To compute gcd(48,18), divide 48 by 18 to get a quotient of 2 and a remainder of 12. Then divide 18 by 12 to get a quotient of 1 and a remainder of 6. Then divide 12 by 6 to get a remainder of 0, which means that 6 is the gcd. Note that we ignored the quotient in each step except to notice when the remainder reached 0, signalling that we had arrived at the answer.

MSB of Larger Integer We use bit shifting. The CPU has an automatic bit-detector already, used for integer to float conversion. So we use that.

Pseudo-code:

number = gets
bitpos = 0

while number != 0
bitpos++ // increment the bit position
number = number » 1 // shift the whole thing to the right once
end
puts bitpos
if the number is zero, bitpos is zero.

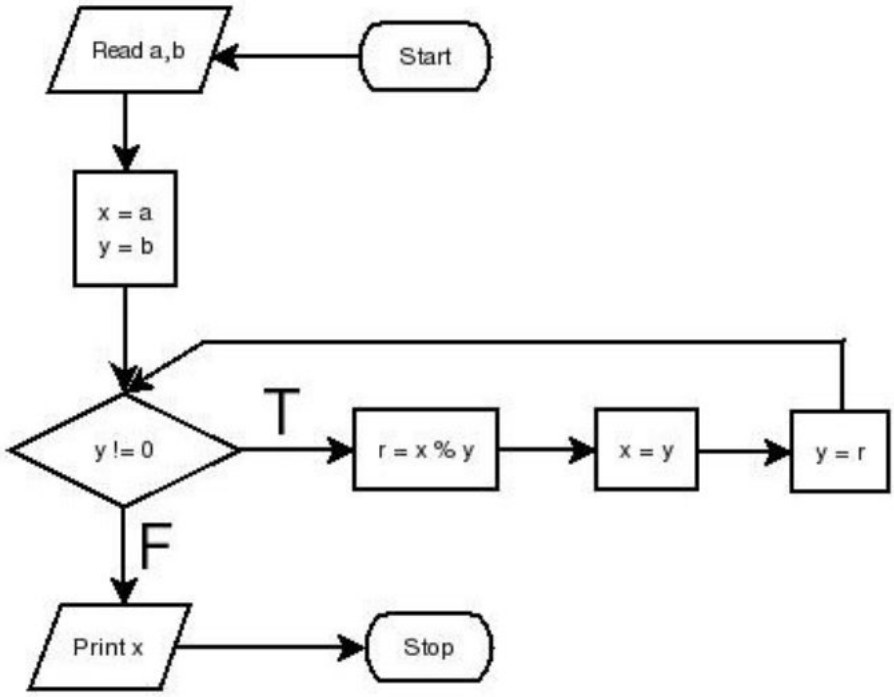
Cosine of angle

1. Convert angle to radian
2. Use systems'c cosine function

1.5 Execution Directive

For LCM
gcc code.c -lm -pg ./a.out
For GCD
gcc code.c -lm -DGCD -pg ./a.out
lm is used for math functions pg is used for profiling
On running the program, gmon.out is created which can be read using
gmon a.out gmon.out >profileresults.txt

1.6 FLOWCHART



1.7 Output Of The Program

```
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass9/code$ gcc code.c -lm -DGCD -pg
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass9/code$ ./a.out

GCD LCM calculator
Enter two integers
12
16

Greatest common divisor of 12 and 16 = 4
Unoptimized time is 49.552000 ms
Greatest common divisor of 12 and 16 = 4
Optimized time is 27.707000 ms

MSB of larger integer
Enter two integers
34
25
The larger integer is 34
MSB is 6
Unoptimized time is 65.651000 ms
MSB is 6
Optimized time is 54.501000 ms

cos of angle in degrees
Enter angle
30
cos of of 30.000000 = 0.866158
Unoptimized time is 64.877000 ms
cos of of 30.000000 = 0.866025
Optimized time is 60.267000 ms
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass9/code$
```

1.8 Result

Running time
Successfully calculated GCD using C code and inline assembly code.
Problems encountered:

1. Code was too small to optimize
2. The least count of gprof is too high. Therefore usng system time was required to calculate time.

Function	C code (ms)	Assembly code (ms)
GCD/LCM	49	27
MSB	65	54
cos	64	60

Table 1.1: Running time comparison

1.9 CONCLUSION

Learnt using inline assembly. Generally the inline term is used to instruct the compiler to insert the code of a function into the code of its caller at the point where the actual call is made. Such functions are called "inline functions". The benefit of inlining is that it reduces function-call overhead.

Now, it's easier to guess about inline assembly. It is just a set of assembly instructions written as inline functions. Inline assembly is used for speed, and you ought to believe me that it is frequently used in system programming.

We can mix the assembly statements within C/C++ programs using keyword asm. Inline assembly is important because of its ability to operate and make its output visible on C/C++ variables.

Library management did not give a measurable improvement with optimization.

Performance comparison of assembly with c

- Most of the code given in this assignment to optimize are too small that compiler generated code worked more or less same (or even better in some cases.
- Compilers can do optimizations that most people can't even imagine. They can take in account inter-procedural optimization and whole-program optimization. Assembly programmer has to make well-defined functions with a well-defined call interface. This prevents many of the optimization methods that compilers use, such as register allocation, constant propagation, common subexpression elimination across functions etc.