

山东大学 计算机科学与技术 学院

汇编语言 课程实验报告

学号：202120130276	姓名：王云强	班级：21.2 班
实验题目：实验 8：GCC 内联汇编优化		
实验学时：2	实验日期：2023.12.8	
<p>实验目的：</p> <ol style="list-style-type: none">1. 掌握 AT&T 语法下的 AMD64 汇编编写。2. 掌握 SIMD 指令的使用，并对程序进行向量化优化。3. 掌握 C 与内联汇编联合编程的方法。		
实验环境：Windows10、DOSBox-0.74、Masm64		
<p>源程序清单：</p> <ol style="list-style-type: none">1. vector.c（示例 vector.c 源程序）		
<p>编译及运行结果：</p> <pre>H:\>gcc vector.c -o vector H:\>vector naive - 2751463424 - 15720861024 cycles expert - 2751463424 - 6478171840 cycles sse - 2751463424 - 4397160032 cycles avx-auto - 2751463424 - 2998107509 cycles avx-manual - 2751463424 - 3009818251 cycles H:\>vector naive - 2751463424 - 15694866742 cycles expert - 2751463424 - 6510630527 cycles sse - 2751463424 - 4420407061 cycles avx-auto - 2751463424 - 2948108022 cycles avx-manual - 2751463424 - 2940962016 cycles</pre>		

问题及收获：

问题：

使用 mingw-w64 作为编译器。该编译器内联汇编的语法与非内联汇编有何不同，又与 MASM 有何不同？

答：首先所谓内联汇编就是将汇编语言插入到高级语言程序中，以汇编语言编写高级语言程序中的一部分子程序，然后遵循高级语言函数的调用规则，这样做的目的是为了优化性能。

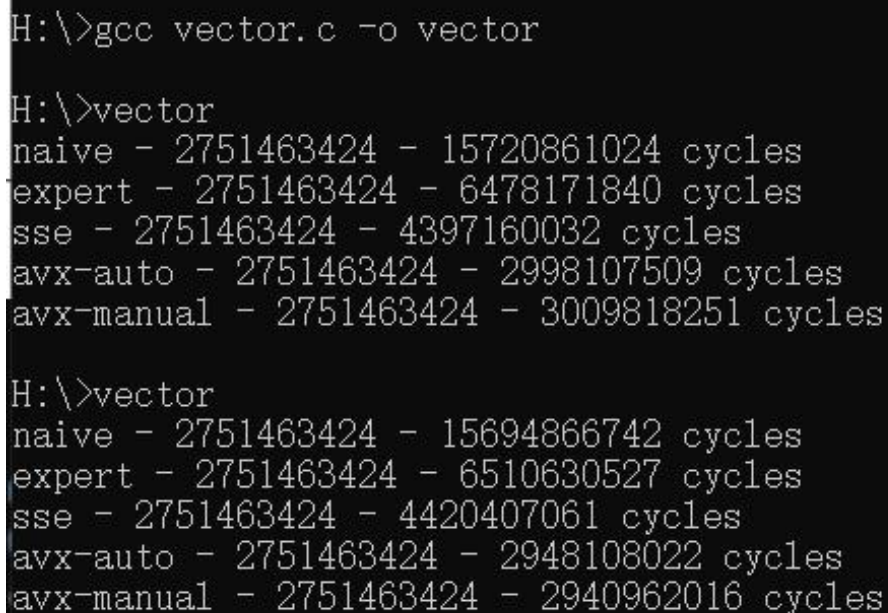
在 mingw-w64 编译器中，内联汇编主要是用来跟 C 语言进行混合，遵循 C 语言的函数调用约定。使用 `__asm__ __volatile__` 声明一段内联汇编。分为四个部分，汇编列表（含有汇编语句的段落）、输入列表（C 语言中作为参数传进来的变量与汇编语言进行对应）、输出列表（将信息从汇编中传回 C 语言）、破坏列表（在汇编中被调用，但是不在输入、输出列表中，需要被 C 语言编译器保护起来的寄存器，如果内存被改动也需要保护内存）。而非内联汇编则就是全汇编语言，不与高级语言进行交互。

而 mingw-w64 编译器与 MASM 编译器不同之处一方面体现在寄存器的使用上，mingw-w64 编译器使用寄存器有前缀，同时指令是有后缀的，而 MASM 编译器寄存器无前缀，指令也没有后缀。除此之外 mingw-w64 编译器目的操作数在后，而 MASM 编译器目的操作数在前。同时，mingw-w64 中还有很多类似 `vpbroadcastd` 的指令，这在 MASM 中是没有的。在 mingw-w64 中也有之前说的汇编列表、输入列表、输出列表、破坏列表，也是 MASM 没有的，因为 mingw-w64 编译器支持内联汇编。

内联汇编的 clobber list(破坏列表)有何用途? 能否不写?

答: 内联汇编的破坏列表, 主要是告诉 C 语言编译器, 有哪些寄存器被使用了, 但是又没出现在输入列表、输出列表中, 而这些寄存器要由 C 语言编译器来保存和恢复。不可以不写, 不写的话, 会导致寄存器的内容可能被覆盖或者破坏。也可以用来保护内存单元, 只需在破坏列表中写 memory 即可。

比较并评价 SSE, AVX2 (与 AVX-512, 有条件的同学可尝试) 向量化指令的性能。能得到什么结论? 为什么?



```
H:\>gcc vector.c -o vector

H:\>vector
naive - 2751463424 - 15720861024 cycles
expert - 2751463424 - 6478171840 cycles
sse - 2751463424 - 4397160032 cycles
avx-auto - 2751463424 - 2998107509 cycles
avx-manual - 2751463424 - 3009818251 cycles

H:\>vector
naive - 2751463424 - 15694866742 cycles
expert - 2751463424 - 6510630527 cycles
sse - 2751463424 - 4420407061 cycles
avx-auto - 2751463424 - 2948108022 cycles
avx-manual - 2751463424 - 2940962016 cycles
```

答: 上图为 vector.c 程序运行结果图, naive 对应的是没有进行任何优化, expert 对应的是 O2 优化, sse 对应的是 O3 优化, avx-auto 是在 O3 优化的基础上利用 avx256 进行优化, avx-manual 是 O3 优化的基础上利用 avx256 进行优化,同时结合手写的汇编代码进行内联汇编进一步优化。中间的值执行得到的结果, 根据图片显示所有的优化得到的结果

都是正确的，而后面的 cycles 数，是执行的时候消耗的周期数，越小，代表的对应的执行速度越快。可以看到正常执行的速度要慢 O2 优化 2 ~ 3 倍（至少在这个例子上看是这样的），而 O3 优化可以进一步加快，利用 avx256 进行优化后性能能够得到进一步提升，在 avx256 基础上在使用内联汇编进行优化后此时性能在五种情况下达到最佳。这说明 SSE、AVX256 向量化指令的优化要明显优于 O2 优化和不进行优化的指令，而 AVX256 要比 SSE 实际上性能能更好一点，至少从上图结果上看是这样的，因为 AVX256 是在 O3 优化的基础上进行进一步优化。AVX256 使用到了向量化指令，功能更加强大，运算速度能够更快，执行的时间也就相应的更短。

收获：

- 1、进一步对内联汇编有了更加深刻的认识，将理论与实践相结合了起来，也对汇编语言的应用有了更深刻的体会（可以通过插入内联汇编到高级语言程序中提升程序性能）
- 2、真正意识到了 O2 优化、O3 优化还有 AVX256 优化等对于提高指令执行速度的作用，通过具体实例可以看出优化后性能提升很多，也体会到了 GCC 的强大之处，对于通过处理指令优化程序执行速度有了进一步的体会。
- 3、对于向量化处理指令有了一些简单的、表面的认识，本实验中没有尝试 AVX512，不然的话可以将其也与上图中五种情况做对比，进而进一步

分析它们之间的差异之处。