作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师

内联汇编的定义

内联汇编把汇编代码嵌套到其他编程语言之中。
内联汇编有许多复杂用法，这里说明基本用法。

内联汇编的基本格式为 `asm volatile ( code : output : input : changed )` 。
asm 表示内联汇编的语句块，等价于__asm__。
volatile 表示禁止编译器优化 asm 代码，等价于__volatile__，可以省略。
code 表示汇编指令，多个汇编指令，可以写多行。
output 表示输出。多个输出，可以写多行。输出可以使用寄存器、内存。可以省略。
input 表示输入。多个输入，可以写多行。输入可以使用寄存器、内存。可以省略。
changed 表示修改了哪些寄存器、内存。可以省略。

输入输出可以指定具体的寄存器，比如 a 表示 rax、b 表示 rbx、c 表示 rcx、d 表示 rdx 等。
输入输出可以不指定具体的寄存器，使用 r 表示通用的寄存器。
输入输出可以使用 m 表示使用内存。

占位符，按照输入输出的顺序，分配编号。编号从 0 开始，依次递增。
汇编代码使用占位符的格式为%n，比如 `movsd %1, %%xmm6` 。
汇编代码使用寄存器的格式为%%reg，比如 `movq %%rbx, %%rdx` 。

示例：

```
    int bb = 66;  // 变量
    int *e = &bb; // 指针

    asm(
        "incl 0(%%rbx)" // int 值+1
        :               // 没有返回结果
        : "b"(e)        // 地址给 rbx
    );

    printf(" result b=%d \n", bb);
```

示例：

```
    int k1 = 3;
    int k2 = 6;
    int w = 0; // w=k1*k2

    asm(
        "imull %1,%2 \n\t"
        "movl  %2,%0 \n\t"
        : "=m"(w)  // 结果 1，内存
        : "m"(k1), // 参数 1，内存
          "b"(k2)  // 参数 2，寄存器
    );
```

```
    printf("result w=%d \n", w);
```

内联汇编的输入和输出

编写代码：input_output.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

// 查看输入、输出
void func_input_output(int32_t in_aa, int64_t in_bb)
{
  int32_t out_cc; // 输出
  int64_t out_dd; // 输出

  asm volatile( // 内联汇编

      "movl %%eax, %%ecx \n" // 把 eax 写到 ecx
      "movq %%rbx, %%rdx \n" // 把 rbx 写到 rdx
      "incl %%ecx \n"        // ecx 加 1
      "decq %%rdx \n"        // rdx 减 1

      : "=c"(out_cc), // 输出，把 ecx 写到 out_cc
        "=d"(out_dd)  // 输出，把 rdx 写到 out_dd

      : "a"(in_aa), // 输入，把 in_aa 写到 eax
        "b"(in_bb)  // 输入，把 in_bb 写到 rbx

      : "memory" // 提示修改了内存。禁止编译器重排序
  );

  printf("Input  :   aa = %d   bb = %lld \n", in_aa, in_bb);
  printf("Output :   cc = %d   dd = %lld \n", out_cc, out_dd);
}

// 变量。
int32_t aa;
int64_t bb;

int main()
{
  printf("Please input like :  100  200 \n");
  scanf("%d %lld", &aa, &bb);
```

```
  func_input_output(aa, bb);

  return 0;
}
```

编译代码：

```
gcc input_output.c -o input_output
```

运行代码：

```
[root@local inline]# ./input_output
Please input like :  100  200
222 333
Input  :   aa = 222   bb = 333
Output :   cc = 223   dd = 332
```

分析结果：
输入 222，加 1 后，输出 223。
输入 333，减 1 后，输出 332。

汇编代码	结果和分析
: "a"(in_aa), // 输入，把 in_aa 写到 eax "b"(in_bb) // 输入，把 in_bb 写到 rbx	输入 2 个参数。 把 32 位整数参数 in_aa 写到 eax。"a"表示 eax。 把 64 位整数参数 in_bb 写到 rbx。"b"表示 rbx。
"movl %%eax, %%ecx \n" // 把 eax 写到 ecx "movq %%rbx, %%rdx \n" // 把 rbx 写到 rdx "incl %%ecx \n" // ecx 加 1 "decq %%rdx \n" // rdx 减 1	执行运算。 把 eax 写到 ecx。然后 ecx 加 1。 把 rbx 写到 rdx。然后 rdx 减 1。 寄存器前缀使用%%。
: "=c"(out_cc), // 输出，把 ecx 写到 out_cc "=d"(out_dd) // 输出，把 rdx 写到 out_dd	输出 2 个结果。 把 32 位整数 ecx 写到变量 out_cc。"=c"表示输出 ecx。 把 64 位整数 rdx 写到变量 out_dd。"=d"表示输出 rdx。

用内联汇编实现 2 个浮点数相加

编写代码：float_sum.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

// 把 2 个 double 相加
double sum_double(double aa, double bb)
{
    double cc; // 返回

    asm volatile( // 内联汇编
```

```
        "movsd %1, %%xmm6 \n"     // 把 内存 1 写到 xmm6
        "movsd %2, %%xmm7 \n"     // 把 内存 2 写到 xmm7
        "addsd %%xmm6, %%xmm7 \n" // 把 xmm6 加到 xmm7
        "movsd %%xmm7, %0 \n"     // 把 xmm7 写到 内存 0

        : "=m"(cc) // 内存 0 。对应返回 cc

        : "m"(aa),  // 内存 1 。对应参数 aa
          "m"(bb)); // 内存 2 。对应参数 bb

    return cc;
}

int main()
{
    double aa = 3333.33;
    double bb = 22.55;

    // 调用函数
    double cc = sum_double(aa, bb);

    printf(" Param  :  aa = %f   bb = %f \n", aa, bb);
    printf(" Result :  sum = %f \n", cc);

    return 0;
}
```

编译代码：

```
gcc float_sum.c -o float_sum
```

运行代码：

```
[root@local inline]# ./float_sum
 Param  :  aa = 3333.330000   bb = 22.550000
 Result :  sum = 3355.880000
```

分析结果：

2 个浮点数 3333.330000 和 22.550000，相加后，输出 3355.880000。

汇编代码	结果和分析
: "m"(aa), // 内存 1 。对应参数 aa "m"(bb)); // 内存 2 。对应参数 bb	把浮点数 aa 和 bb 写到内存。 "m"表示内存。
"movsd %1, %%xmm6 \n" // 把 内存 1 写到 xmm6 "movsd %2, %%xmm7 \n" // 把 内存 2 写到 xmm7 "addsd %%xmm6, %%xmm7 \n" // 把 xmm6 加到 xmm7 "movsd %%xmm7, %0 \n" // 把 xmm7 写到 内存 0	执行运算。 2 个浮点数相加，结果写到内存。 寄存器前缀使用%%。 内存前缀使用%。 占位符从 0 开始。
: "=m"(cc) // 内存 0 。对应返回 cc	把内存中的结果写到变量 cc。 "=m"表示输出内存。

使用 3 个输入输出，依次为 `"=m"(cc)` 、 `"m"(aa)` 、 `"m"(bb)` 。
占位符的顺序与输入输出的顺序一致。
`"=m"(cc)` 对应编号 0。 `movsd %%xmm7, %0` 表示把 xmm7 的值写到变量 cc。
`"m"(aa)` 对应编号 1。 `movsd %1, %%xmm6` 表示把 aa 的值写到 xmm6。
`"m"(bb)` 对应编号 2。 `movsd %2, %%xmm7` 表示把 bb 的值写到 xmm7。