Inline Assembly in GCC

- General form:

```
asm( "code" : outputs : inputs : clobbers);
```

- Processor dependent
- Calling conventions on m68k:
  - d0, d1, a0, a1 - scratch registers
  - all other registers - preserved by functions
- Complete documentation available in GCC manual, section *C Extensions->Extended Asm*

Read Register Operand

```
void* a=(void*)0x1000000;
asm("movec %0,%%vbr":/* no output */ : "d"(a));
```

The generated code could be:

```
        move.l #16777216,%d0
#APP
        movec %d0,%vbr
#NO_APP
```

Write Register Operand

```
unsigned short fun()
{
        unsigned short a;
        asm("move.w %%sr,%0":"=d"(a));
        return a;
}
```

The generated code could be:

```
        link.w %fp,#0
#APP
        move.w %sr,%d0
#NO_APP
        and.l #65535,%d0
        unlk %fp
        rts
```

Read/write Register Operand

```
int fun()
{
        int a=10;
        int b=15;
        asm("addl %1,%0":"+d"(a): "d"(b):"cc");
        return a;
}
```

The generated code could be:

```
        link.w %fp,#0
        moveq #15,%d1
        moveq #10,%d0
#APP
        addl %d1,%d0
#NO_APP
        unlk %fp
        rts
```

Read/write Memory Operand - Alternate Version

```
int fun()
{
        int a=10;
        int b=15;
        int c=32;
        asm("addl %2,%0":"=d"(a): "0"(a),"d"(b): "cc");
        return a;
}
```

The generated code could be:

```
        link.w %fp,#0
        moveq #10,%d0
        moveq #15,%d1
#APP
        addl %d1,%d0
#NO_APP
        unlk %fp
        rts
```

Operand Reuse

```
int fun()
{
        int a;
        int b=15;
        int c=32;
        asm("addl %2,%0":"=d"(a): "0"(c),"d"(b): "cc");
        return a;
}
```

The generated code could be:

```
        link.w %fp,#0
        moveq #32,%d0
        moveq #15,%d1
#APP
        addl %d1,%d0
#NO_APP
        unlk %fp
        rts
```

Read/write Memory Operand

```
int fun()
{
        int a=10;
        int b=15;
        asm("addl %1,%0":"+m"(a): "d"(b):"cc");
        return a;
}
```

The generated code could be:

```
        link.w %fp,#-4
        moveq #10,%d0
        move.l %d0,-4(%fp)
        move.b #15,%d0
#APP
        addl %d0,-4(%fp)
#NO_APP
        move.l -4(%fp),%d0
        unlk %fp
        rts
```

Multiple Choices for Operand Location

```
int fun()
{
        int a=10;
        int b=15;
        asm("addl %1,%0":"+md"(a): "d"(b):"cc");
        return a;
}
```

The generated code could be:

```
        link.w %fp,#0
        moveq #15,%d1
        moveq #10,%d0
#APP
        addl %d1,%d0
#NO_APP
        unlk %fp
        rts
```

Complex Expressions

```
struct test {
  int a;
  int b;
};
struct test tab[10];
void fun() {
        asm("addl %1,%0":"+m"(tab[3].a): "d"(tab[5].b):"cc");
}
```

The generated code could be:

```
        link.w %fp,#0
        move.l tab+44,%d0
#APP
        addl %d0,tab+24
#NO_APP
        unlk %fp
        rts
```

Multiple Instructions, Clobber List

```
int fun()
{
        int a=10;
        int b=15;
        asm("addl %1,%%d3\n\tmulu %%d3,%0"
            : "+md"(a): "d"(b): "d3","cc");
        return a;
}
```

The generated code could be:

```
fun:
        link.w %fp,#0
        move.l %d3,-(%sp)
        moveq #15,%d1
        moveq #10,%d0
#APP
        addl %d1,%d3
        mulu %d3,%d0
#NO_APP
        move.l (%sp)+,%d3
        unlk %fp
        rts
```

The `volatile` Keyword

- If an `asm' has output operands, GCC assumes for optimization purposes the instruction has no side effects except to change the output operands. This does not mean instructions with a side effect cannot be used, but the compiler may eliminate them if the output operands aren't used, or move them out of loops, or replace two with one if they constitute a common subexpression.
- You can prevent an `asm' instruction from being deleted by writing the keyword `volatile' after the `asm'.

```
void fun()
{
        unsigned short a;
        asm volatile("move.w %%sr,%0":"=d"(a): :"cc");
}
```

- Even a volatile `asm' instruction can be moved relative to other code, including across jump instructions. See manual for details how to work around this problem.