The easiest way to see how the input and output values work is to see some examples. This example:

```
asm ("assembly code" : "=a"(result) : "d"(data1), "c"(data2));
```

places the C variable `data1` into the `EDX` register, and the variable `data2` into the `ECX` register. The result of the inline assembly code will be placed into the `EAX` register, and then moved to the `result` variable.

Using registers

If the input and output variables are assigned to registers, the registers can be used within the inline assembly code almost as normal. I use the word "almost" because there is one oddity to deal with.

In extended `asm` format, to reference a register in the assembly code you must use two percent signs instead of just one (the reason for this will be discussed a little later). This makes the code a little odd looking, but not too much different.

The `regtest1.c` program demonstrates using registers within the extended `asm` format:

```
/* regtest1.c - An example of using registers */
#include <stdio.h>

int main()
{
   int data1 = 10;
   int data2 = 20;
   int result;

   asm ("imull %%edx, %%ecx\n\t"
        "movl %%ecx, %%eax"
        : "=a"(result)
        : "d"(data1), "c"(data2));

   printf("The result is %d\n", result);
   return 0;
}
```

This time, the C variables are declared as local variables, which you couldn't do with the basic `asm` format. Each C variable is assigned to a specific register. The output register is modified with the equal sign to indicate that it can only be written to by the assembly code (this is required for all output values in the inline code).

When the C program is compiled, the compiler automatically generates the assembly code necessary to place the C variables in the appropriate registers to implement the inline assembly code. You can see what is generated again by using the `-S` option. The inline code generated looks like this:

```
        movl    $10, -4(%ebp)
        movl    $20, -8(%ebp)
        movl    -4(%ebp), %edx
        movl    -8(%ebp), %ecx
#APP
        imull %edx, %ecx
```