The asm statement can be used to embed code written in assembly inside a C or XC function. For example, the add instruction can be written as follows:

```
asm("add %0, %1, %2" : "=r"(result) : "r"(a), "r"(b));
```

Colons separate the assembler template, the output operands and the input operands. Commas separate operands within a group. Each operand is described by an operand constraint string followed by an expression in parentheses. The "r" in the operand constraint string indicates that the operand must be located in a register. The "=" in the operand constraint string indicates that the operand is written.

Each output operand expression must be an lvalue and must have "=" in its constraint.

The location of an operand may be referred to in the assembler template using an escape sequence of the form %num where num is the operand number. The escape sequence "%=" can be used to emit a number that is unique to each expansion of an asm statement. This can be useful for making local labels. To produce a literal "%" you must write "%%".

If code overwrites specific registers this can be described by using a third colon after the input operands, followed by the names of the clobbered registers as a comma-separated list of strings. For example:

```
asm ("get r11, id\n\tmov %0, r11"
     : "=r"(result)
     : /* no inputs */
     : "r11");
```

The compiler ensures none of input or output operands are placed in clobbered registers.

If an asm statement has output operands, the compiler assumes the statement has no side effects apart from writing to the output operands. The compiler may remove the asm statement if the values written by the asm statement are unused. To mark an asm statement as having side effects add the volatile keyword after asm. For example:

```
asm volatile("in %0, res[%1]" : "=r"(result) : "r"(lock));
```

If the asm statement accesses memory, add "memory" to the list of clobber registers. For example:

```
asm volatile("stw %0, dp[0]"
     : /* no outputs */
     : "r"(value));
```

This prevents the compiler caching memory values in registers around the `asm` statement.

The *earlyclobber* constraint modifier "`&`" can be used to specify that an output operand is modified before all input operands are consumed. This prevents the compiler from placing the operand in the same register as any of the input operands. For example:

```
asm("or %0, %1, %2\n"
    "or %0, %0, %3\n"
    : "=&r"(result)
    : "r"(a), "r"(b), "r"(c));
```

Jumps from one `asm` statement to another are not supported. `asm` statements must not be used to modify the event enabled status of any resource.