5.38.1 Simple Constraints

The simplest kind of constraint is a string full of letters, each of which describes one kind of operand that is permitted. Here are the letters that are allowed:

whitespace
: Whitespace characters are ignored and can be inserted at any position except the first. This enables each alternative for different operands to be visually aligned in the machine description even if they have different number of constraints and modifiers.

`m'
: A memory operand is allowed, with any kind of address that the machine supports in general. Note that the letter used for the general memory constraint can be re-defined by a back end using the `TARGET_MEM_CONSTRAINT` macro.

`o'
: A memory operand is allowed, but only if the address is *offsettable*. This means that adding a small integer (actually, the width in bytes of the operand, as determined by its machine mode) may be added to the address and the result is also a valid memory address.

  For example, an address which is constant is offsettable; so is an address that is the sum of a register and a constant (as long as a slightly larger constant is also within the range of address-offsets supported by the machine); but an autoincrement or autodecrement address is not offsettable. More complicated indirect/indexed addresses may or may not be offsettable depending on the other addressing modes that the machine supports.

  Note that in an output operand which can be matched by another operand, the constraint letter `o' is valid only when accompanied by both `<' (if the target machine has predecrement addressing) and `>' (if the target machine has preincrement addressing).

`V'
: A memory operand that is not offsettable. In other words, anything that would fit the `m' constraint but not the `o' constraint.

`<'
: A memory operand with autodecrement addressing (either predecrement or postdecrement) is allowed.

`>'
: A memory operand with autoincrement addressing (either preincrement or postincrement) is allowed.

`r'
: A register operand is allowed provided that it is in a general register.

`i'

An immediate integer operand (one with constant value) is allowed. This includes symbolic constants whose values will be known only at assembly time or later.

`n'

An immediate integer operand with a known numeric value is allowed. Many systems cannot support assembly-time constants for operands less than a word wide. Constraints for these operands should use `n' rather than `i'.

`I', `J', `K', ... `P'

Other letters in the range `I' through `P' may be defined in a machine-dependent fashion to permit immediate integer operands with explicit integer values in specified ranges. For example, on the 68000, `I' is defined to stand for the range of values 1 to 8. This is the range permitted as a shift count in the shift instructions.

`E'

An immediate floating operand (expression code `const_double`) is allowed, but only if the target floating point format is the same as that of the host machine (on which the compiler is running).

`F'

An immediate floating operand (expression code `const_double` or `const_vector`) is allowed.

`G', `H'

`G' and `H' may be defined in a machine-dependent fashion to permit immediate floating operands in particular ranges of values.

`s'

An immediate integer operand whose value is not an explicit integer is allowed.

This might appear strange; if an insn allows a constant operand with a value not known at compile time, it certainly must allow any known value. So why use `s' instead of `i'? Sometimes it allows better code to be generated.

For example, on the 68000 in a fullword instruction it is possible to use an immediate operand; but if the immediate value is between −128 and 127, better code results from loading the value into a register and using the register. This is because the load into the register can be done with a `moveq' instruction. We arrange for this to happen by defining the letter `K' to mean “any integer outside the range −128 to 127”, and then specifying `Ks' in the operand constraints.

`g'

Any register, memory or immediate integer operand is allowed, except for registers that are not general registers.

`X'

Any operand whatsoever is allowed.

`0', `1', `2', ... `9'

An operand that matches the specified operand number is allowed. If a digit is used together with letters within the same alternative, the digit should come last.

This number is allowed to be more than a single digit. If multiple digits are encountered consecutively, they are interpreted as a single decimal integer. There is scant chance for ambiguity, since to-date it has never been desirable that `10' be interpreted as matching either operand 1 *or* operand 0. Should this be desired, one can use multiple alternatives instead.

This is called a *matching constraint* and what it really means is that the assembler has only a single operand that fills two roles which asm distinguishes. For example, an add instruction uses two input operands and an output operand, but on most CISC machines an add instruction really has only two operands, one of them an input-output operand:

```
addl #35,r12
```

Matching constraints are used in these circumstances. More precisely, the two operands that match must include one input-only operand and one output-only operand. Moreover, the digit must be a smaller number than the number of the operand that uses it in the constraint.

`p'

An operand that is a valid memory address is allowed. This is for “load address” and “push address” instructions.

`p' in the constraint must be accompanied by address_operand as the predicate in the match_operand. This predicate interprets the mode specified in the match_operand as the mode of the memory reference for which the address would be valid.

other-letters

Other letters can be defined in machine-dependent fashion to stand for particular classes of registers or other arbitrary operand types. `d', `a' and `f' are defined on the 68000/68020 to stand for data, address and floating point registers.