GCC Internals
Code generation

Diego Novillo
`dnovillo@google.com`

November 2007

Code generation

- Code is generated using a rewriting system
- Target specific configuration files in

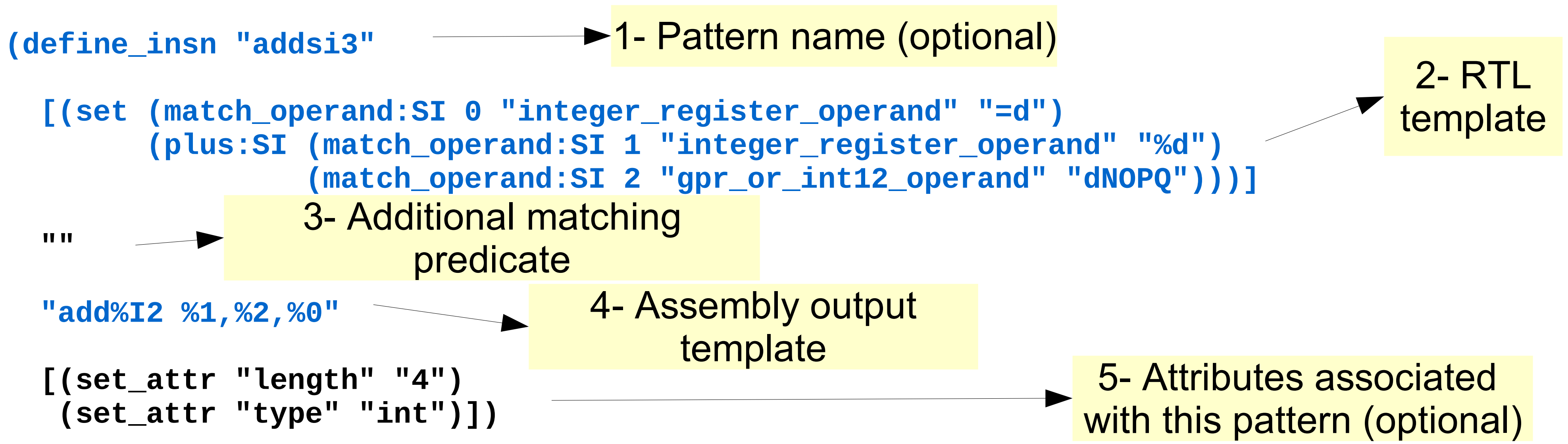
  `gcc/config/<arch>`

- Three main target-specific files

– `<arch>.md`	Code generation patterns for RTL insns
– `<arch>.h`	Definition of target capabilities (register classes, calling conventions, type sizes, etc)
– `<arch>.c`	Support functions for code generation, predicates and target variants

Code generation

- Two main types of rewriting schemes supported
  - Simple mappings from RTL to assembly (`define_insn`)
  - Complex mappings from RTL to RTL (`define_expand`)
- `define_insn` patterns have five elements



```
(define_insn "addsi3"

  [(set (match_operand:SI 0 "integer_register_operand" "=d")
        (plus:SI (match_operand:SI 1 "integer_register_operand" "%d")
                 (match_operand:SI 2 "gpr_or_int12_operand" "dNOPQ")))]

  ""

  "add%I2 %1,%2,%0"

  [(set_attr "length" "4")
   (set_attr "type" "int")])
```

1- Pattern name (optional)
2- RTL template
3- Additional matching predicate
4- Assembly output template
5- Attributes associated with this pattern (optional)

Code generation

`define_insn    addsi3`

- Named patterns
  - Used to generate RTL
  - Some standard names are used by code generator
  - Some missing standard names are replaced with library calls (e.g., `divsi3` for targets with no division operation)
  - Some pattern names are mandatory (e.g. move operations)
- Unnamed (anonymous) patterns do not generate RTL, but can be used in insn combination

Code generation

```
[(set (match_operand:SI 0 "integer_register_operand" "=d,=d")
      (plus:SI (match_operand:SI 1 "integer_register_operand" "%d,m")
               (match_operand:SI 2 "gpr_or_int12_operand""dNOPQ,m")))]
```

Matching uses
- Machine mode (SI, DI, HI, SF, etc)
- Predicate (a C function)
- Both operands and operators can be matched

Constraints provide second level of matching
Select best operand among the set of allowed operands
Letters describe kinds of operands
Multiple alternatives separated by commas

Code generation

```
"add%I2 %1,%2,%0"
```

- Code is generated by emitting strings of target assembly
- Operands in the insn pattern are replaced in the %n placeholders
- If constraints list multiple alternatives, multiple output strings must be used
- Output may be a simple string or a C function that builds the output string

Pattern expansion

- Some standard patterns cannot be used to produce final target code. Two ways to handle it
  - Do nothing. Some patterns can be expanded to libcalls
  - Use `define_expand` to generate matchable RTL

- Four elements
  - The name of a standard insn
  - Vector of RTL expressions to generate for this insn
  - A C expression acting as predicate to express availability of this instruction
  - A C expression used to generate operands or more RTL

Pattern expansion

```
(define_expand "ashlsi3"
  [(set (match_operand:SI 0 "register_operand" "")
        (ashift:SI
           (match_operand:SI 1 "register_operand" "")
           (match_operand:SI 2 "nonmemory_operand" "")))]
  ""
  "{
     if (GET_CODE (operands[2]) != CONST_INT
         || (unsigned) INTVAL (operands[2]) > 3)
       FAIL;
   }")
```

- Generate a left shift only when the shift count is [0...3]
- **FAIL** indicates that expansion did not succeed and a different expansion should be tried (e.g., a library call)
- **DONE** is used to prevent emitting the RTL pattern. C fragment responsible for emitting all insns.