Introduction to Machine Descriptions

Uday Khedker
(www.cse.iitb.ac.in/~uday)

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

13 June 2014
Outline

- Influences on GCC Machine Descriptions
- Organization of GCC Machine Descriptions
- Machine description constructs
- The essence of retargetability in GCC
Part 1

Influences on Machine Descriptions
Examples of Influences on the Machine Descriptions

Source Language
- INT_TYPE_SIZE
- Activation Record

GCC Architecture
- Generation of `nop`
- tree covers for instruction selection
- `define_predicate`

Build System
Host System

Target System
- Instruction Set Architecture
- Assembly and executable formats

Machine Description

<target>.h

hwint.h

{<target>.md
{<target>.h
{<target>.h
other headers

Uday Khedker
GRC, IIT Bombay
Part 2

Organization of GCC MD
GCC Machine Descriptions

- Processor instructions useful to GCC
- Processor characteristics useful to GCC
- Target ASM syntax
- Target specific optimizations as IR-RTL $\rightarrow$ IR-RTL transformations
  
  (GCC code performs the transformation computations, MD supplies their *target patterns*)

  - Peephole optimizations
  - Transformations for enabling scheduling
Syntactic Entities in GCC MD

- **Necessary Specifications**
  - Processor instructions useful to GCC
    - One GIMPLE $\rightarrow$ One IR-RTL
    - One GIMPLE $\rightarrow$ More than one IR-RTL
  - Processor characteristics useful to GCC
  - Target ASM syntax
  - IR-RTL $\rightarrow$ IR-RTL transformations
  - Target Specific Optimizations

- **Programming Conveniences**
  (eg. `define_insn_and_split`, `define_constants`, `define_cond_exec`, `define_automaton`)

```c
#define_insn
#define_expand
#define_cpu_unit
part_of(define_insn)
define_split
#define_peephole2```
The GCC MD comprises of

- `<target>.h`: A set of C macros that describe
  - HLL properties: e.g. `INT_TYPE_SIZE` to h/w bits
  - Activation record structure
  - Target Register (sub)sets, and characteristics
    (lists of read-only regs, dedicated regs, etc.)
  - System Software details: formats of assembler, executable etc.

- `<target>.md`: Target instructions described using MD constructs.

- `<target>.c`: Optional, but usually required.
  C functions that implement target specific code
  (e.g. target specific activation layout).
The GCC MD comprises of

- `<target>.h`: A set of C macros that describe
  - HLL properties: e.g. INT_TYPE_SIZE to h/w bits
  - Activation record structure
  - Target Register (sub)sets, and characteristics
    (lists of read-only regs, dedicated regs, etc.)
  - System Software details: formats of assembler, executable etc.

- `<target>.md`: Target instructions described using MD constructs.
  (Our main interest!)

- `<target>.c`: Optional, but usually required.
  C functions that implement target specific code
  (e.g. target specific activation layout).
Part 3

Essential Constructs in Machine Descriptions
The GCC Phase Sequence

- Parse
- Gimplify
- Tree SSA Optimize
- Generate RTL
- Optimize RTL
- Generate ASM

GIMPLE → RTL
RTL → ASM

Target Independent → Target Dependent
The GCC Phase Sequence

Target Independent → Target Dependent

Parse → Gimplify → Tree SSA Optimize → Generate RTL → Optimize RTL → Generate ASM

GIMPLE → RTL
RTL → ASM

MD Info Required
The GCC Phase Sequence

Observe that

- RTL is a target specific IR
- GIMPLE $\rightarrow$ non strict RTL $\rightarrow$ strict RTL.
- Standard Pattern Name (SPN):
  “Semantic Glue” between GIMPLE and RTL
  - operator match + coarse operand match, and
  - refine the operand match
- Finally: Strict RTL $\Leftrightarrow$ Unique target ASM string

Consider generating RTL expressions of GIMPLE nodes

- Two constructs available: `define_insn` and `define_expand`
Running Example

Consider a *data move* operation

- **reads** data from *source* location, and
- **writes** it to the *destination* location.
- **GIMPLE** node: GIMPLE_ASSIGN
- **SPN**: “movsi”

Some possible combinations are:

- Reg ← Reg : Register move
- Reg ← Mem : Load
- Reg ← Const : Load immediate
- Mem ← Reg : Store
- Mem ← Mem : Illegal instruction
- Mem ← Const : Illegal instruction
Specifying Target Instruction Semantics

(define_insn
  "movsi"
  (set
   (match_operand 0 "register_operand" "=r")
   (match_operand 1 "const_int_operand" "k")
  )
  "" /* C boolean expression, if required */
  "li %0, %1"
)
Specifying Target Instruction Semantics

Introduce instruction pattern

```
(define_insn
  "movsi"
  (set
    (match_operand 0 "register_operand" "=r")
    (match_operand 1 "const_int_operand" "k")
  )

"" /* C boolean expression, if required */
"li %0, %1"
```

RTL Expression (RTX):
Semantics of target instruction

Target assembly instruction =
Concrete syntax for RTX
Specifying Target Instruction Semantics

```
(define_insn "movsi"
  (set
   (match_operand 0 "register_operand" "=r")
   (match_operand 1 "const_int_operand" "k")
  )

"" /* C boolean expression, if required */
"li %0, %1"
```

- **RTL operator**: `define_insn` and `match_operand`
- **MD constructs**: `set`
- **Predicates**: `register_operand` and `const_int_operand`
- **Constraints**: `=r` and `k`
(define_insn "movsi"
  (set (match_operand 0 "register_operand" "%r")
       (match_operand 1 "const_int_operand" "%k"))
  "" /* C boolean expression, if required */
  "li %0, %1"
)
Instruction Specification and Translation

- GIMPLE: target independent
- RTL: target dependent
- Need: associate the semantics

⇒ GCC Solution: Standard Pattern Names

(define_insn "movsi"
  (set (match_operand 0 "register_operand" "=r")
       (match_operand 1 "const_int_operand" "k"))
  "li %0, %1"
)

Uday Khedker
Instruction Specification and Translation

- GIMPLE: target independent
- RTL: target dependent
- Need: associate the semantics

GCC Solution: Standard Pattern Names

GIMPLE \rightarrow RTL

RTL Template

ASM

(define_insn "movsi"
  (set (match_operand 0 "register_operand" "=r")
       (match_operand 1 "const_int_operand" "k"))
  "li %0, %1"
  /* C boolean expression, if required */
)
General Move Instruction

(define_insn "maybe_spn_like_movsi"
  (set (match_operand 0 "general_operand" "")
       (match_operand 1 "general_operand" ""))
  ""
  "mov %0, %1"
)

- This define_insn can generate data movement patterns of all combinations
- Even Mem → Mem is possible.
- We need a mechanism to generate more restricted data movement RTX instances!
The `define_expand` Construct

```
(define_expand "movsi"

[(set (match_operand:SI 0 "nonimmediate_operand" "")
     (match_operand:SI 1 "general_operand" ""))

"

"

{
    if (GET_CODE (operands[0]) == MEM &&
        GET_CODE (operands[1]) != REG)
        if (can_create_pseudo_p())
            operands[1] = force_reg (SImode, operands[1]);
}
```

Uday Khedker
Part 4

The Essence of Retargetability
Instruction Specification and Translation: A Recap

- GIMPLE: target independent
- RTL: target dependent
- Need: associate the semantics

⇒GCC Solution: Standard Pattern Names

GIMPLE

(define_insn "movsi"
(set (match_operand 0 "register_operand" "=r")
(match_operand 1 "const_int_operand" "k"))
"
/* C boolean expression, if required */
"li %0, %1"
)

RTL Template

ASM

GRC, IIT Bombay
Translation Sequence in GCC

(define_insn
  "movsi"
  (set
    (match_operand 0 "register_operand" "=r")
    (match_operand 1 "const_int_operand" "k")
  )
  "" /* C boolean expression, if required */
  "li %0, %1"
)
Translation Sequence in GCC

```
(define_insn
  "movsi"
  (set
    (match_operand 0 "register_operand" "=r")
    (match_operand 1 "const_int_operand" "k")
  )
  "" /* C boolean expression, if required */
  "li %0, %1"
)
```

D.1283 = 10;

```
(set
  (reg:SI 58 [D.1283])
  (const_int 10: [0xa])
)
```

li $t0, 10
The Essence of Retargetability

When are the machine descriptions read?
When are the machine descriptions read?

- During the build process
When are the machine descriptions read?

- During the build process
- When a program is compiled by gcc the information gleaned from machine descriptions is consulted
Retargetability Mechanism of GCC

Input Language

Compiler Generation Framework

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Selected

Copied

Copied

Copied

Generated

Generated

Generated

Development Time

Build Time

Use Time

Generated Compiler

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Uday Khedker
Retargetability Mechanism of GCC

Retargetability Mechanism of GCC

Input Language

Compiler Generation Framework

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Selected

Copied

Copied

Generated

Generated

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Development Time

Build Time

Use Time

GIMPLE → IR-RTL

IR-RTL → ASM

Generated Compiler
Retargetability Mechanism of GCC

- Input Language
- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

Compiler Generation Framework:
- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Generated Compiler:
- GIMPLE → PN
- PN → IR-RTL
- IR-RTL → ASM

Development Time:
- GIMPLE → PN
- PN → IR-RTL
- IR-RTL → ASM

Build Time:
- GIMPLE → IR-RTL
- IR-RTL → ASM

Use Time:
- GIMPLE → PN
- PN → IR-RTL
- IR-RTL → ASM

Uday Khedker
GRC, IIT Bombay
Retargetability Mechanism of GCC

Input Language

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Compiler Generation Framework

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Generated Compiler

GIMPLE → PN

PN → IR-RTL

IR-RTL → ASM

GIMPLE → IR-RTL

IR-RTL → ASM

Development Time

Build Time

Use Time
Retargetability Mechanism of GCC

Input Language

- Language Specific Code
- Language and Machine Independent Generic Code

Compiler Generation Framework

- Machine Dependent Generator Code
- Machine Descriptions

Target Name

- GIMPLE → PN
- PN → IR-RTL
- IR-RTL → ASM

Generated Compiler

- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Uday Khedker

GRC, IIT Bombay
Part 5

More Features of MD
Pattern Names in .md File

All Patterns

Named Patterns

Unnamed Patterns
Pattern Names in .md File

- All Patterns
  - Named Patterns
    - With *
    - Without *
  - Unnamed Patterns
Pattern Names in \texttt{.md} File

- All Patterns
  - Named Patterns
    - With $\star$
      - Standard
    - Without $\star$
      - Non-Standard
  - Unnamed Patterns
Pattern Names in .md File

- All Patterns
  - Named Patterns
    - With *
      - No gen_ function
    - Without *
      - Standard
      - Non-Standard
  - Unnamed Patterns
    - No gen_ function
**Pattern Names in .md File**

- **All Patterns**
  - **Named Patterns**
    - With *
      - No gen function
    - Without *
      - Standard
        - gen_name function
        - Called implicitly
        - Can be called explicitly
      - Non-Standard
        - gen_name function
        - Not called implicitly
        - Can be called explicitly
  - Unnamed Patterns
    - No gen function
Role of `define_expanded`

Uses of `define_expanded`

- **generate RTL**
  - implicitly
  - code in a preparatory statement
- explicitly
  - some function in a `.c` file

- **do not generate RTL**
  - setting operands
  - setting global variables

Uday Khedker

GRC, IIT Bombay
Use of Predicates

```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" ";=r")
    (plus:SI (match_dup 0))
    (match_operand:SI 1 "general_operand" ";r"))])

""
"..."
```

Predicates are using for matching operands

- For constructing an insn during expansion
  <name> must be a standard pattern name

- For recognizing an instruction (in subsequent RTL passes including pattern matching)
Use of Predicates

Predicates are used for matching operands

- For constructing an insn during expansion
  `<name>` must be a standard pattern name

- For recognizing an instruction (in subsequent RTL passes including pattern matching)

```
(define_insn "<name>"
 [(set (match_operand:SI 0 "general_operand" "=r")
     (plus:SI (match_dup 0)
       (match_operand:SI 1 "general_operand" "r"))))
 ""
 "..."
```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" ";=r")
        (plus:SI (match_dup 0)
                 (match_operand:SI 1 "general_operand" ";r").))]
  ""
  "...")
Understanding Constraints

```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" "]="r")
       (plus:SI (match_dup 0)
                 (match_operand:SI 1 "general_operand" "r")))
   ""
   "...")
```
Understanding Constraints

(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" "=r")
       (plus:SI (match_dup 0))
       (match_operand:SI 1 "general_operand" "r")))]

""
"...")

- Reloading operands in the most suitable register class
Understanding Constraints

- Reloading operands in the most suitable register class
- Fine tuning within the set of operands allowed by the predicate

```scheme
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" ";=r")
      (plus:SI (match_dup 0)
        (match_operand:SI 1 "general_operand" ";r")))]
  ""
  "..."
)
### Understanding Constraints

- Reloading operands in the most suitable register class
- Fine tuning within the set of operands allowed by the predicate
- If omitted, operands will depend only on the predicates

```lisp
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" ";r")
       (plus:SI (match_dup 0)
         (match_operand:SI 1 "general_operand" "r"))))]

  ""
  "..."
)
Part 6

Machine Descriptions in specRTL
The specification mechanism for Machine descriptions is quite adhoc

- Only syntax borrowed from LISP, neither semantics not spirit!
- Non-composable rules
- Mode and code iterator mechanisms are insufficient
Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple `define_insn` and `define_expand_pattern`
  - Some Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ, everything else may be identical
  - One RTL expression may appear as a sub-expression of some other RTL expression

- Repetition of C code along with RTL expressions in these patterns.
Redundancy in MIPS Machine Descriptions: Example 1

\[
[(\text{set} \ (\text{match\_operand}: m \ 0 \ "register\_operand" \ "c0"))
\left(\text{plus}: m \ (\text{match\_operand}: m \ 1 \ "register\_operand" \ "c1"))
\left(\text{match\_operand}: m \ 2 \ "p" \ "c2")))\]
\]
Redundancy in MIPS Machine Descriptions: Example 1

\[(\text{set} \ (\text{match\_operand}: m \ 0 \ "\text{register\_operand}\ " \ "c0")
\ (\text{plus}: m \ (\text{match\_operand}: m \ 1 \ "\text{register\_operand}\ " \ "c1")
\ (\text{match\_operand}: m \ 2 \ "p\ " \ "c2")))]
Redundancy in MIPS Machine Descriptions: Example 1

\[
[(\text{set} \ (\text{match\_operand:} \ m \ 0 \ "\text{register\_operand}\" \ "c0\")) \\
(\text{plus:} \ m \ (\text{match\_operand:} \ m \ 1 \ "\text{register\_operand}\" \ "c1\") \\
(\text{match\_operand:} \ m \ 2 \ "p" \ "c2\")])]
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Pattern name} & m & p & c0 & c1 & c2 \\
\hline
\text{define\_insn} & \text{ANYF} & \text{register\_operand} & =f & f & f \\
\text{add\_mode>3} & & & & & \\
\hline
\text{define\_expand} & \text{GPR} & \text{arith\_operand} & & & \\
\text{add\_mode>3} & & & & & \\
\hline
\text{define\_insn} & \text{GPR} & \text{arith\_operand} & =d,d & d,d & d,Q \\
\text{*add\_mode>3} & & & & & \\
\hline
\end{array}
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
\begin{align*}
\text{(set } & (\text{match_operand:}_m 0 \text{ "register_operand" } "c0") \\
& (\text{mult:}_m (\text{match_operand:}_m 1 \text{ "register_operand" } "c1") \\
& (\text{match_operand:}_m 2 \text{ "register_operand" } "c2")))\]
\end{align*}
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
[(\text{set } (\text{match_operand: } m \ 0 \ "\text{register_operand}" \ "c0"))
(\text{mult: } m \ (\text{match_operand: } m \ 1 \ "\text{register_operand}" \ "c1"))
(\text{match_operand: } m \ 2 \ "\text{register_operand}" \ "c2"))]
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
\begin{align*}
\text{[}(\text{set (match_operand:} m \ 0 \ "\text{register_operand}" \ "c0")} \\
\text{\quad (mult:} m \ (\text{match_operand:} m \ 1 \ "\text{register_operand}" \ "c1")} \\
\text{\quad \quad (match_operand:} m \ 2 \ "\text{register_operand}" \ "c2")])
\end{align*}
\]

**RTL Template**

**Structure**

**Details**

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>m</th>
<th>c0</th>
<th>c1</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>define_insn *mul&lt;mode&gt;3</td>
<td>SCALARF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_insn *mul&lt;mode&gt;3_r4300</td>
<td>SCALARF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_insn mulv2sf3</td>
<td>V2SF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_expand mul&lt;mode&gt;3</td>
<td>GPR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>define_insn mul&lt;mode&gt;3_mul3_loongson</td>
<td>GPR</td>
<td>=d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>define_insn mul&lt;mode&gt;3_mul3</td>
<td>GPR</td>
<td>d,1</td>
<td>d,d</td>
<td>d,d</td>
</tr>
</tbody>
</table>
Redundancy in MIPS Machine Descriptions: Example 3

\[
[(\text{set} (\text{match\_operand:} m 0 "register\_operand" "c0") (\text{plus:} m \\
(\text{mult:} m (\text{match\_operand:} m 1 "register\_operand" "c1") \\
(\text{match\_operand:} m 2 "register\_operand" "c2"))))) \\
(\text{match\_operand:} m 3 "register\_operand" "c3")))]
\]

*RTL Template*
Redundancy in MIPS Machine Descriptions: Example 3

\[
\text{(set (match_operand: } \text{m} 0 \text{ "register_operand" } \text{"c0"}) \text{ (plus: } \text{m} \\
\text{ (mult: } \text{m} \text{ (match_operand: } \text{m} 1 \text{ "register_operand" } \text{"c1"}) \\
\text{ (match_operand: } \text{m} 2 \text{ "register_operand" } \text{"c2")})}) \\
\text{ (match_operand: } \text{m} 3 \text{ "register_operand" } \text{"c3"}))}
\]
Redundancy in MIPS Machine Descriptions: Example 3

\[
\text{[(set (match_operand: } m \text{ 0 "register_operand" } "c0") (plus: } m \\
\text{ (mult: } m \text{ (match_operand: } m \text{ 1 "register_operand" } "c1") \\
\text{ (match_operand: } m \text{ 2 "register_operand" } "c2")))]} \\
\text{(match_operand: } m \text{ 3 "register_operand" } "c3"))]
\]

### RTL Template

\[
\begin{array}{c}
\text{Pattern name} \\
*\text{mul.acc.si} \\
*\text{mul.acc.si_r3900} \\
*\text{macc} \\
*\text{madd4\{mode\}} \\
*\text{madd3\{mode\}} \\
\end{array}
\begin{array}{cccccc}
\text{Pattern name} \\
*\text{mul.acc.si} \\
*\text{mul.acc.si_r3900} \\
*\text{macc} \\
*\text{madd4\{mode\}} \\
*\text{madd3\{mode\}} \\
\end{array}
\begin{array}{c}
m \\
SI \\
SI \\
SI \\
ANYF \\
ANYF \\
\end{array}
\begin{array}{c}
c0 \\
=1*?*?,d? \\
=1*?*?,d*?,d? \\
=1,d \\
=f \\
=f \\
\end{array}
\begin{array}{c}
c1 \\
d,d \\
d,d,d \\
f \\
f \\
\end{array}
\begin{array}{c}
c2 \\
d,d \\
d,d,d \\
f \\
f \\
\end{array}
\begin{array}{c}
c3 \\
0,d \\
0,1,d \\
0,1 \\
\text{f} \\
0 \\
\end{array}
\]
Insufficient Iterator Mechanism

- Iterators cannot be used across define_insn, define-expand, define_peephole2 and other patterns.
- Defining iterator attribute for each varying parameter becomes tedious.
- For same set of modes and rtx codes, change in other fields of pattern makes use of iterators impossible.
- Mode and code attributes cannot be defined for operator or operand number, name of the pattern etc.
- Patterns with different RTL template share attribute value vector for which iterators can not be used.
specRTL: Key Observations

- Davidson Fraser insight

  *Register transfers are target specific but their form is target independent*

- GCC’s approach

  - Use Target independent RTL for machine specification
  - Generate expander and recognizer by reading machine descriptions

Main problems with GCC’s Approach

*Although the shapes of RTL statements are target independent, they have to be provided in RTL templates*

- Our key idea:

  *Separate shapes of RTL statements from the target specific details*
Specification Goals of specRTL

Support all of the following

- Separation of shapes from target specific details
- Creation of new shapes by composing shapes
- Associating concrete details with shapes
- Overriding concrete details
Software Engineering Goals of specRTL

- Allow non-disruptive migration for existing machine descriptions
  - Incremental changes
  - No need to change GCC source until we are sure of the new specification

GCC must remain usable after each small change made in the machine descriptions
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape $\equiv$ tree structure of RTL templates
  - Details $\equiv$ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)

- Abstract patterns and Concrete patterns
  - Abstract patterns are shapes with “holes” in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes (e.g. modes, predicates, constraints etc.)

- Abstract patterns and Concrete patterns
  - Abstract patterns are shapes with “holes” in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information

- Abstract patterns capture shapes which can be concretized by providing details
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: extends
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: extends
- Associating concrete details with shapes: instantiates
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: `extends`
- Associating concrete details with shapes: `instantiates`
- Overriding concrete details: `overrides`
Creating Abstract Patterns

abstract set_plus extends set
{
    root.2 = plus;
}

abstract set_macc extends set_plus
{
    root.2.2 = mult;
}
abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f",
             register_operand:ANYF:"f",
             register_operand:ANYF:"f");
    root.2.mode = ANYF;
}

cconcrete add<mode>3.expand instantiates set_plus
{
    set_plus(register_operand:GPR:"",
             register_operand:GPR:"",
             arith_operand:GPR:"");
    root.2.mode = GPR;
}
Generating Conventional Machine Descriptions

abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
        register_operand:ANYF:"f");
    root.2.mode = ANYF;
}

:: /* Conventional Machine Description Fragments */ :
Generating Conventional Machine Descriptions

abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
        register_operand:ANYF:"f"),
    root.2.mode = ANYF;
}

:{ /* Conventional Machine Description Fragments */ :}

Resulting MD Specification

(define_insn "add<mode>3"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (match_operand:ANYF 1 "register_operand" "f")
      (match_operand:ANYF 2 "register_operand" "f")))]
  /* Conventional Machine Description Fragments */
)
abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.expand instantiates set_plus
{
    set_plus(register_operand:GPR:"",
              register_operand:GPR:"",
              arith_operand:GPR:"");  
    root.2.mode = GPR;
}
abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.expand instantiates set_plus
{
    set_plus(register_operand:GPR:"",
             register_operand:GPR:"",
             arith_operand:GPR:""),
    root.2.mode = GPR;
}

concrete *add<mode>3.insn overrides add<mode>3.expand
{
    allconstraints = ("=d,d", "d,d", "d,Q");
}
Part 7

Conclusions
GCC MD Summary

• GCC achieves retargetability by reading the machine descriptions and generating a back end customised to the machine descriptions

• Machine descriptions are influenced by:
The HLLs, GCC architecture, and properties of target, host and build systems

• Writing machine descriptions requires:
  specifying the C macros, target instructions and any required support functions

• `define_insn` and `define_expand` are used to convert a GIMPLE representation to RTL
specRTL: Current Status and Plans for Future Work

- specRTL compiler version 2 is ready
- i386 and MIPS machine descriptions have been re-written
specRTL: Conclusions

- Separating shapes from concrete details is very helpful
- It may be possible to identify a large number of common shapes
- Machine descriptions may become much smaller
  Only the concrete details need to be specified
- Non-disruptive and incremental migration to new machine descriptions
- GCC source need not change until these machine descriptions have been found useful