Workshop on Essential Abstractions in GCC

More Details of Machine Descriptions

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

2 July 2011

Outline

- Some details of MD constructs
  - On names of patterns in `.md` files
  - On the role of `define_expand`
  - On the role of predicates and constraints
  - Mode and code iterators
  - Defining attributes
  - Other constructs
- Improving machine descriptions and instruction selection
  - New constructs to factor out redundancy
  - Cost based tree tiling for instruction selection

2 July 2011 MD Details: Outline 1/36

Outline

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

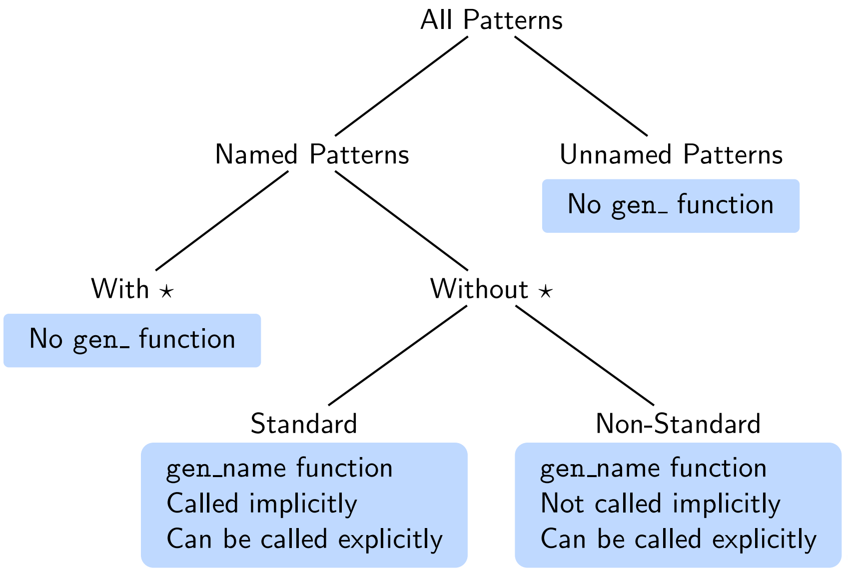
Part 1

More Features

Notes

2 July 2011 MD Details: More Features 2/36

Pattern Names in `.md` File

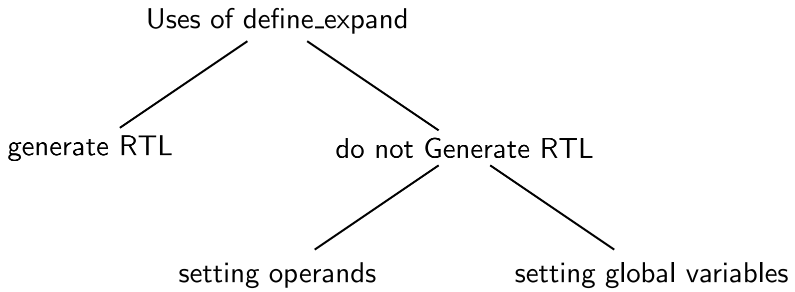


Pattern Names in `.md` File

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Role of `define_expand`



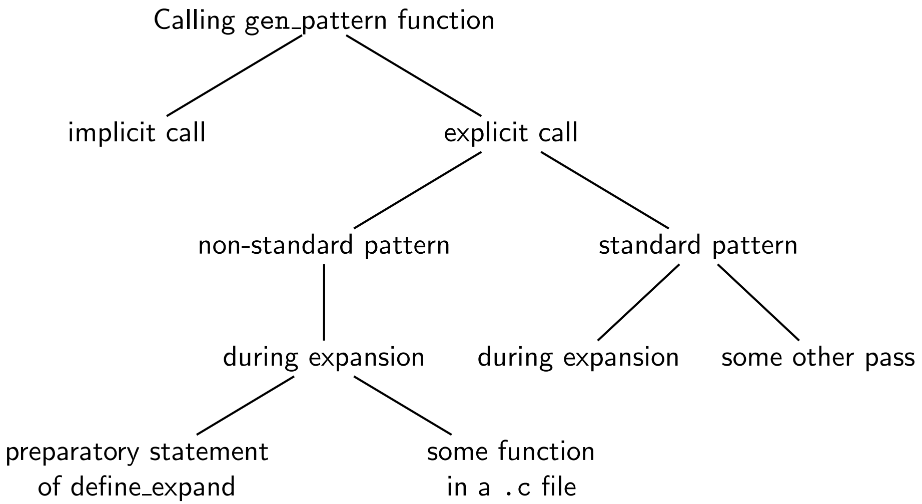
2 July 2011 MD Details: More Features 3/36

Role of `define_expand`

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Using `define_expand` for Generating RTL statements



2 July 2011 MD Details: More Features 4/36

Using `define_expand` for Generating RTL statements

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Use of Predicates

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)

2 July 2011 MD Details: More Features 5/36

Use of Predicates

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Understanding Constraints

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
- Fine tuning within the set of operands allowed by the predicate
- If omitted, operands will depend only on the predicates

2 July 2011 MD Details: More Features 6/36

Understanding Constraints

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Role of Constraints

Consider the following two instruction patterns:

- ```
  (define_insn ""
    [(set (match_operand:SI 0 "general_operand" "=r")
        (plus:SI (match_dup 0)
             (match_operand:SI 1 "general_operand" "r")))]
     ""
     "...")
  ```
  - During expansion, the destination and left operands must match the same predicate
  - During recognition, the destination and left operands must be identical
- ```
  (define_insn ""
    [(set (match_operand:SI 0 "general_operand" "=r")
        (plus:SI (match_operand:SI 1 "general_operand" "z")
             (match_operand:SI 2 "general_operand" "r")))]
     ""
     "...")
  ```


Role of Constraints

Notes

Role of Constraints

- Consider an insn for recognition
  ```
  (insn n prev next
     (set (reg:SI 3)
        (plus:SI (reg:SI 6) (reg:SI 109)))
         ...)}
  ```
- Predicates of the first pattern do not match (because they require identical operands during recognition)
- Constraints do not match for operand 1 of the second pattern
- Reload pass generates additional insn to that the first pattern can be used

```
(insn n2 prev n
    (set (reg:SI 3) (reg:SI 6))
   ...)
(insn n n2 next
    (set (reg:SI 3)
        (plus:SI (reg:SI 3)(reg:SI 109)))
   ...)
```


Role of Constraints

Notes

Part 2

Factoring Out Common Information

Notes

Handling Mode Differences

```
(define_insn "subsi3"
    [(set (match_operand:SI 0 "register_operand" "=d")
          (minus:SI (match_operand:SI 1 "register_operand" "d")
                    (match_operand:SI 2 "register_operand" "d")))]
    ""
    "subu\t %0,%1,%2"
    [(set_attr "type" "arith")
    (set_attr "mode" "SI")])
(define_insn "subdi3"
    [(set (match_operand:DI 0 "register_operand" "=d")
          (minus:DI (match_operand:DI 1 "register_operand" "d")
                    (match_operand:DI 2 "register_operand" "d")))]
    ""
    "dsubu\t %0,%1,%2"
    [(set_attr "type" "arith")
    (set_attr "mode" "DI")])
```


2 July 2011 MD Details: Factoring Out Common Information 9/36

Handling Mode Differences

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Mode Iterators: Abstracting Out Mode Differences

```
(define_mode_iterator GPR [SI (DI "TARGET_64BIT")])
(define_mode_attr d [(SI "") (DI "d")])
(define_insn "sub<mode>3"
    [(set (match_operand:GPR 0 "register_operand" "=d")
          (minus:GPR (match_operand:GPR 1 "register_operand" "d")
                     (match_operand:GPR 2 "register_operand" "d")))]
    ""
    "<d>subu\t %0,%1,%2"
    [(set_attr "type" "arith")
    (set_attr "mode" "<MODE>")])
```


2 July 2011 MD Details: Factoring Out Common Information 10/36

Mode Iterators: Abstracting Out Mode Differences

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Handling Code Differences

```
(define_expand "bunordered"
    [(set (pc) (if_then_else (unordered:CC (cc0) (const_int 0))
                             (label_ref (match_operand 0 ""))
                             (pc)))]
    ""
    { mips_expand_conditional_branch (operands, UNORDERED);
      DONE;
    })

 (define_expand "bordered"
    [(set (pc) (if_then_else (ordered:CC (cc0) (const_int 0))
                             (label_ref (match_operand 0 ""))
                             (pc)))]
    ""
    { mips_expand_conditional_branch (operands, ORDERED);
      DONE;
    })
```


2 July 2011 MD Details: Factoring Out Common Information 11/36

Handling Code Differences

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Code Iterators: Abstracting Out Code Differences

```
(define_code_iterator any_cond [unordered ordered])
(define_expand "b<code>"
    [(set (pc)
          (if_then_else (any_cond:CC (cc0)
                                  (const_int 0))
                        (label_ref (match_operand 0 ""))
                        (pc)))]
    ""
    { mips_expand_conditional_branch (operands, <CODE>);
      DONE;
    })
```


Code Iterators: Abstracting Out Code Differences

Notes

Part 3

Miscellaneous Features

Notes

Defining Attributes

- Classifications are need based
- Useful to GCC phases – e.g. pipelining

Property: Pipelining
Need: To classify target instructions
Construct: `define_attr`

```
;; Instruction type.
(define_attr "type"
  "other,multi, alu,alu1,negnot,   ...   str ,cld, ..."
  (const_string "other") )
```

Fields:
Attribute name, all possible values, one of the possible values, default.

Defining Attributes

Notes

Specifying Instruction Attributes

- Optional field of a `define_insn`
- For an i386, we choose to mark string instructions with the attribute value `str`

```
(define_insn "*strmovdi_rex_1"
  [(set (mem:DI (match_operand:DI 2 ...)]
  "TARGET_64BIT && (TARGET_SINGLE_ ...)"
  "movsq"
  [ (set_attr "type" "str")
   ...
   (set_attr "memory" "both")])
```

NOTE
An instruction may have more than one attribute!

Specifying Instruction Attributes

Notes

Using Attributes

```
(define_insn_reservation "pent_str" 12
  (and (eq_attr "cpu" "pentium")
       (eq_attr "type" "str") )
  "pentium-np*12")
```

Pipeline specification requires the CPU type to be "pentium" and the instruction type to be "str"

Using Attributes

Notes

Some Other RTL Constructs

- `define_split`: Split complex insn into simpler ones e.g. for better use of delay slots
- `define_insn_and_split`: A combination of `define_insn` and `define_split`
  Used when the split pattern matches and insn exactly.
- `define_peephole2`: Peephole optimization over insns that substitutes insns. Run after register allocation, and before scheduling.
- `define_constants`: Use literal constants in rest of the MD.

Some Other RTL Constructs

Notes

Part 4

Improving Machine Descriptions

Notes

The Need for Improving Machine Descriptions

The Problems:

- 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
- Adhoc design decisions
  - Honouring operand constraints delayed to global register allocation
    During GIMPLE to RTL translation, a lot of C code is required
  - Choice of insertion of NOPs

The Need for Improving Machine Descriptions

Notes

Handing Constraints

- `define_insns` patterns have operand predicates and constraints
- While generating an RTL insn from GIMPLE, only the predicates are checked. The constraints are completely ignored
- An insn which is generated in the expander is modified in the reload pass to satisfy the constraints
- It may be possible to generate this final form of RTL during expansion by honouring `constraints`
  - Honouring contraints earlier than the current place
    ⇒ May get rid of some C code in `define_expand`

Handing Constraints

Notes

Design Flaws in Machine Descriptions

Multiple patterns with same structure

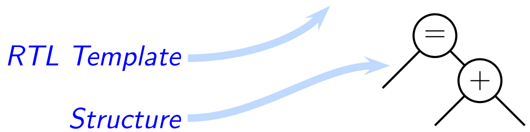
- Repetition of almost similar RTL expressions across multiple `define_insn` an `define_expand patterns`
  - Some Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ everything else may be identical
  - One RTL expression may appears as a sub-expression of some other RTL expression
- Repetition of C code along with RTL expressions in these patterns.

Design Flaws in Machine Descriptions

Notes

Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```



Details

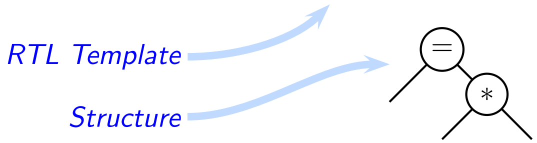
Pattern name	*m*	*p*	*c0*	*c1*	*c2*
define_insn add<mode>3	ANYF	register_operand	=f	f	f
define_expand add<mode>3	GPR	arith_operand			
define_insn *add<mode>3	GPR	arith_operand	=d,d	d,d	d,Q

Redundancy in MIPS Machine Descriptions: Example 1

Notes

Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
          (match_operand:m 2 "register_operand" "c2")))]
```



Details

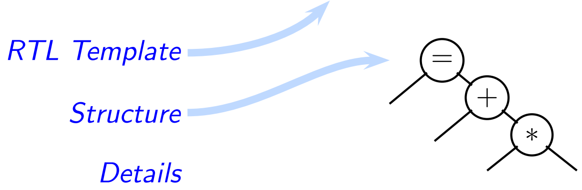
Pattern name	*m*	*c0*	*c1*	*c2*
define_insn *mul<mode>3	SCALARF	=f	f	f
define_insn *mul<mode>3_r4300	SCALARF	=f	f	f
define_insn mulv2sf3	V2SF	=f	f	f
define_expand mul<mode>3	GPR			
define_insn mul<mode>3_mul3_loongson	GPR	=d	d	d
define_insn mul<mode>3_mul3	GPR	d,1	d,d	d,d

Redundancy in MIPS Machine Descriptions: Example 2

Notes

Redundancy in MIPS Machine Descriptions: Example 3

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



Pattern name	*m*	*c0*	*c1*	*c2*	*c3*
*mul_acc_si	SI	=l*?*?,d?	d,d	d,d	0,d
*mul_acc_si_r3900	SI	=l*?*?,d*?,d?	d,d,d	d,d,d	0,1,d
*macc	SI	=l,d	d,d	d,d	0,1
*madd4<mode>	ANYF	=f	f	f	f
*madd3<mode>	ANYF	=f	f	f	0

2 July 2011 MD Details: Improving Machine Descriptions 22/36

Redundancy in MIPS Machine Descriptions: Example 3

Notes

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

2 July 2011 MD Details: Improving Machine Descriptions 23/36

Insufficient Iterator Mechanism

Notes

Many Similar Patterns Cannot be Combined

```
(define_expand "iordi3"
   [(set (match_operand:DI 0 "nonimmediate_operand" "")
      (ior:DI (match_operand:DI 1 "nonimmediate_operand" "")
          (match_operand:DI 2 "x86_64_general_operand" "")))
   (clobber (reg:CC FLAGS_REG))]
   "TARGET_64BIT"
   "ix86_expand_binary_operator (IOR, DImode, operands); DONE;")

(define_insn "*iordi_1_rex64"
   [(set (match_operand:DI 0 "nonimmediate_operand" "=rm,r")
      (ior:DI (match_operand:DI 1 "nonimmediate_operand" "%0,0")
          (match_operand:DI 2 "x86_64_general_operand" "re,rme" )))
   (clobber (reg:CC FLAGS_REG))]
   "TARGET_64BIT
   && ix86_binary_operator_ok (IOR, DImode, operands)"
   "or{q}\t{%2, %0|%0, %2}"
   [(set_attr "type" "alu")
   (set_attr "mode" "DI")])
```


Many Similar Patterns Cannot be Combined

Notes

Measuring Redundancy in RTL Templates

MD File	Total number of patterns	Number of primitive trees	Number of times primitive trees are used to create composite trees
i386.md	1303	349	4308
arm.md	534	232	1369
mips.md	337	147	921

Measuring Redundancy in RTL Templates

Notes

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

specRTL: Key Observations

Notes

Specification Goals of specRTL

Support all of the following

- Separation of shapes from target specific details
- Creation of new shapes by composing shapes
- Associtiating concrete details with shapes
- Overriding concrete details

Specification Goals of specRTL

Notes

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

Software Engineering Goals of specRTL

Notes

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.)
- *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: Key Idea

Notes

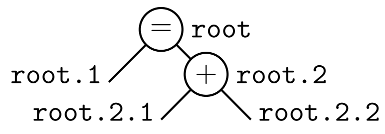
Meeting the Specification Goals: Operations

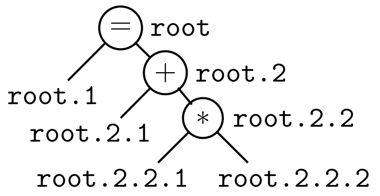
- Creating new shapes by composing shapes: `extends`
- Associtiating concrete details with shapes: `instantiates`
- Overriding concrete details: `overrides`

Meeting the Specification Goals: Operations

Notes

Creating Abstract Patterns

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

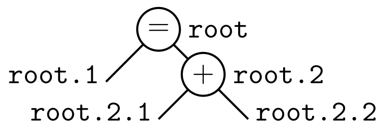
abstract set_macc extends
   set_plus
{
  root.2.2 = mult;
}
	

Creating Abstract Patterns

Notes

Creating Concrete Patterns

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



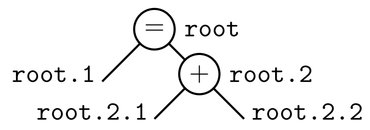

```
concrete add<mode>3.insn instantiates set_plus
{ set_plus(register_operand:ANYF:"=f",
           register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
concrete add<mode>3.expand instantiates set_plus
{ set_plus(register_operand:GPR:"",
           register_operand:GPR:"",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


Creating Concrete Patterns

Notes

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

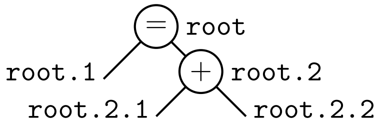
```
(define_insn "add<mode>3"
[(set (match_operand:ANYF 0 "register_operand" "=f")
      (plus:ANYF (match_operand:ANYF 1 "register_operand" "f")
                 (match_operand:ANYF 2 "register_operand" "f")))]
/* Conventional Machine Description Fragments */
)
```


Generating Conventional Machine Descriptions

Notes

Overriding Details

```
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"); }
```


Overriding Details

Notes

Current Status and Plans for Future Work

- specRTL parser has been augmented with semantic checks
  Emitting conventional machine descriptions is pending
- i386 move instructions and spim add instructions have been rewritten
  Other instructions are being rewritten
- Suggestions have been received to improve the syntax

Current Status and Plans for Future Work

Notes

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

2 July 2011 MD Details: Improving Machine Descriptions 36/36

Conclusions

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay