Workshop on Essential Abstractions in GCC

Introduction to 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

- 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 \texttt{nop}
- Tree covers for instruction selection
- \texttt{define\_predicate}

Machine Description
- \texttt{<target>.h}
- \texttt{hwint.h}

Target System
- Instruction Set Architecture
- Assembly and executable formats

Build System
- \texttt{<target>.h}

Host System
- \texttt{<target>.h}
- \texttt{other\_headers}

Notes
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 → 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 \(\text{define}_\text{insn}\)
    - One GIMPLE \(\rightarrow\) More than one IR-RTL \(\text{define}_\text{expand}\)
  - Processor characteristics useful to GCC \(\text{define}_\text{cpu}\)
  - Target ASM syntax \(\text{define}_\text{insn}\)
  - IR-RTL \(\rightarrow\) IR-RTL transformations \(\text{define}_\text{split}\)
  - Target Specific Optimizations
    - \(\text{define}_\text{peephole2}\)
- Programming Conveniences
  (eg. \text{define}_\text{insn_and_split}, \text{define}_\text{constants}, \text{define}_\text{cond_exec}, \text{define}_\text{automaton})

File Organization of GCC MD

The GCC MD comprises of

- \(<\text{target}>\).h: A set of C macros that describe
  - HLL properties: e.g. \text{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.
- \(<\text{target}>\).md: Target instructions described using MD constructs.

\(<\text{target}>\).md: Target instructions described using MD constructs. (Our main interest!)

- \(<\text{target}>\).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

Target Independent

<table>
<thead>
<tr>
<th>Parse</th>
<th>Simplify</th>
<th>Tree SSA</th>
<th>Optimize</th>
<th>Generate RTL</th>
<th>Optimize RTL</th>
<th>Generate ASM</th>
</tr>
</thead>
</table>

Target Dependent

GIMPLE → RTL

RTL → ASM

MD Info Required
Observe that

- RTL is a target specific IR
- GIMPLE → non strict RTL → 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 ⇔ Unique target ASM string

Consider generating RTL expressions of GIMPLE nodes

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

---

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 instruction pattern**

```c
(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"
)
```

**Standard Pattern Name**

**RTL Expression (RTX):** Semantics of target instruction

**target asm inst. =** Concrete syntax for RTX

---

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
### Instruction Specification and Translation

- **Target Independent**
  - Parse
  - Simplify
  - Tree SSA Optimize

- **Target Dependent**
  - Gimplify
  - Generate RTL
  - Optimize RTL
  - Generate ASM

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

GCC Solution: Standard Pattern Names

#### Notes

- **General Move Instruction**

  ```c
  (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"
  )
  ```

- This `define_insn` can generate data movement patterns of all combinations
- Even Mem $\rightarrow$ 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]);
   }
)
```

Relationship Between `<target>.md`, `<target>.c`, and `<target>.h` Files

Example:
- Register class constraints are used in `<target>.md` file
- Register class is defined in `<target>.h` file
- Checks for register class are implemented in `<target>.c` file
Register Class Constraints in `<target>.md` File

```markdown
;; Here z is the constraint character defined in
;; REG_CLASS_FROM_LETTER_P
;; The register $zero is used here.
(define_insn "IITB_move_zero"
  [(set
     (match_operand:SI 0 "nonimmediate_operand" "=r,m")
     (match_operand:SI 1 "zero_register_operand" "z,z")
   )]
  "move \t%0,%1sw \t%1, %m0"
)
```

The Register Class letter code

```
/* From spim.h */
#define REG_CLASS_FROM_LETTER_P
  reg_class_from_letter
enum reg_class
{
  \NO_REGS, \ZERO_REGS,
  CALLER_SAVED_REGS, \CALLEE_SAVED_REGS,
  BASE_REGS, \GENERAL_REGS,
  ALL_REGS, \LIM_REG_CLASSES
};

#define REG_CLASS_CONTENTS
{0x00000000, 0x00000003, 0xff00ffff, 0x00ff0000,
  0xf0000000, 0x0cfffff3, 0xffffffff}
```

The Register Classes

The Register Class Enumeration
enum reg_class
reg_class_from_letter (char ch)
{
    switch(ch)
    {
        case 'b': return BASE_REGS;
        case 'x': return CALLEE_SAVED_REGS;
        case 'y': return CALLER_SAVED_REGS;
        case 'z': return ZERO_REGS;
    }
    return NO_REGS;
}

Get the enumeration from the Register class letter
Instruction Specification and Translation: A Recap

- Target Independent
  - Parse → Simplify → Tree SSA → Generate RTL → Optimize RTL → Generate ASM

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

GCC Solution: Standard Pattern Names

GIMPLE ASSIGN

(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 */
  "li %0, %1"
)

Development

Use

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The Essence of Retargetability

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
 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.

\texttt{define_insn} and \texttt{define_expand} are used to convert a GIMPLE representation to RTL.