Workshop on Essential Abstractions in GCC

The Retargetability Model of GCC

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

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

2 July 2012

Outline

- A Recap
- Generating the code generators
- Using the generator code generators

Outline

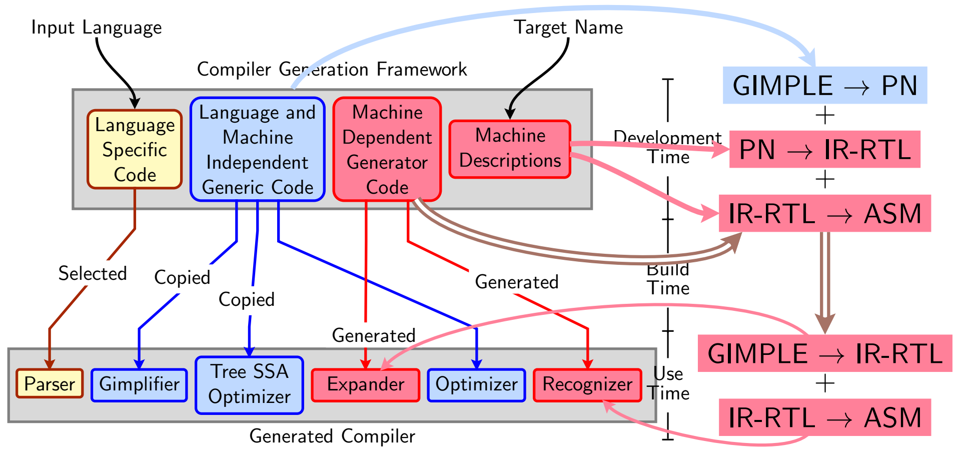
Notes

Part 1

A Recap

Notes

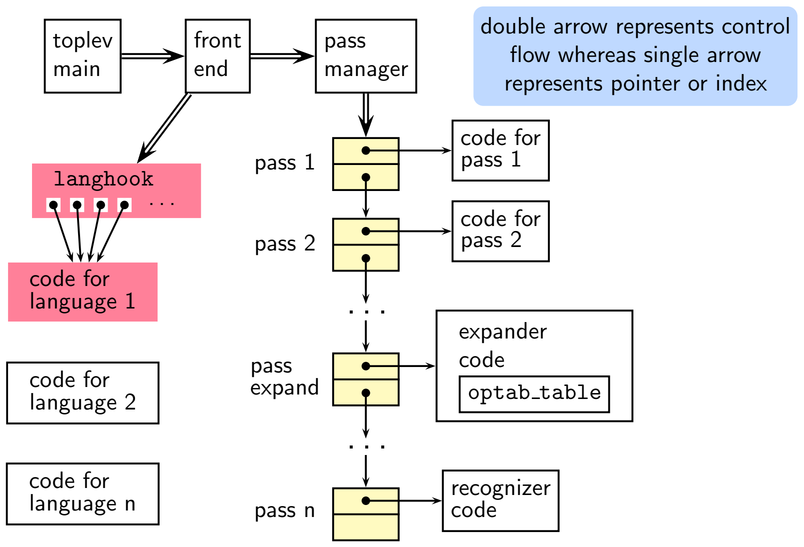
Retargetability Mechanism of GCC



Retargetability Mechanism of GCC

Notes

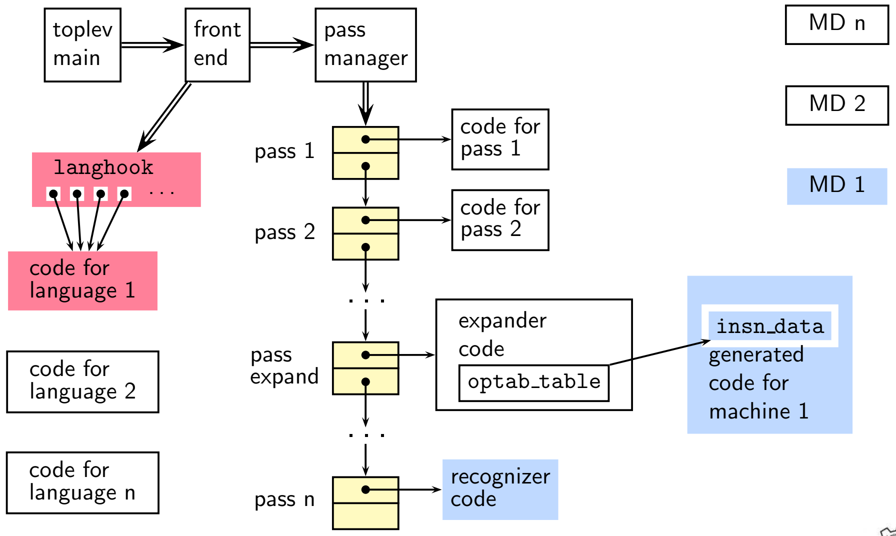
Plugin Structure in `cc1`



Plugin Structure in `cc1`

Notes

Plugin Structure in `cc1`



Plugin Structure in `cc1`

Notes

What is “Generated”?

- Info about instructions supported by chosen target, e.g.
  - Listing data structures (e.g. instruction pattern lists)
  - Indexing data structures, since diff. targets give diff. lists.
- C functions that generate RTL internal representation
- Any useful “attributes”, e.g.
  - Semantic groupings: arithmetic, logical, I/O etc.
  - Processor unit usage groups for pipeline utilisation

What is “Generated”?

Notes

Information Supplied by Machine Descriptions

- The target instructions – as ASM strings
- A description of the semantics of each
- A description of the features of each like
  - Data size limits
  - One of the operands must be a register
  - Implicit operands
  - Register restrictions

Information supplied	in `define_insn` as
The target instruction	ASM string
A description of it's semantics	RTL Template
Operand data size limits	predicates
Register restrictions	constraints

Information Supplied by Machine Descriptions

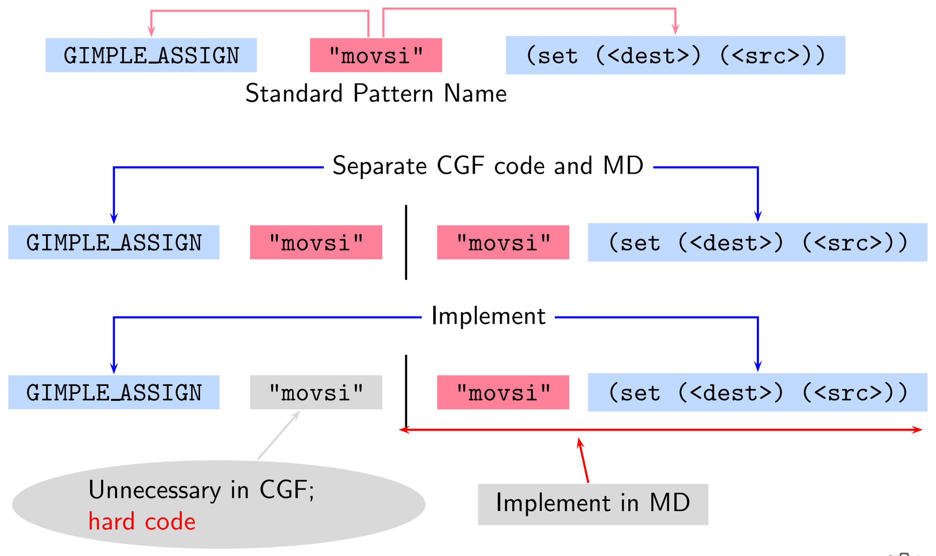
Notes

Part 2

Generating the Code Generators

Notes

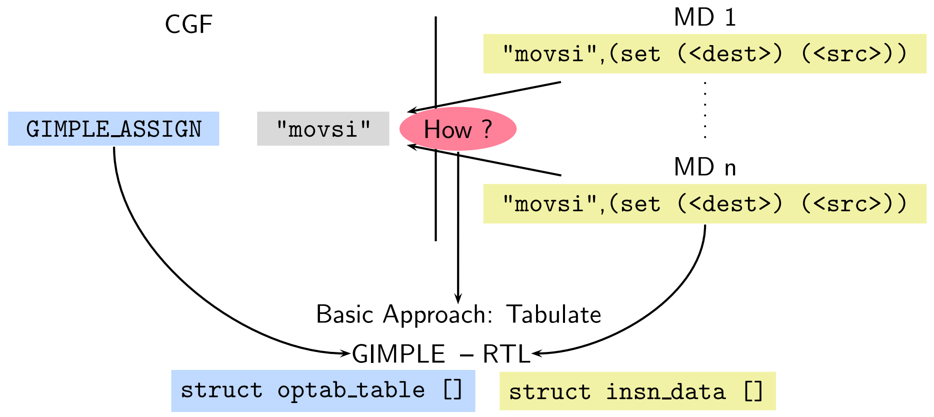
Using Target Specific RTL as IR



Using Target Specific RTL as IR

Notes

Retargetability ⇒ Multiple MD vs. One CGF!



CGF needs:

An interface immune to MD authoring variations

Retargetability ⇒ Multiple MD vs. One CGF!

Notes

MD Information Data Structures

Two principal data structures

- `struct optab` – Interface to CGF
- `struct insn_data` – All information about a pattern
  - Array of each pattern read
  - Some patterns are SPNs
  - Each pattern is accessed using the generated index

Supporting data structures

- `enum insn_code`: Index of patterns available in the given MD

Note

Data structures are named in the CGF, but populated at build time.
Generating target specific code = populating these data structures.

MD Information Data Structures

Notes

Operation Table

- One optab for every standard pattern name

```
struct optab_d
{
  enum rtx_code code;
  char libcall_suffix;
  const char *libcall_basename;
  void (*libcall_gen)(struct optab_d *, const char *name, char suffix,
                      enum machine_mode);
  struct optab_handlers handlers[NUM_MACHINE_MODES];
};
typedef struct optab_d * optab;
```

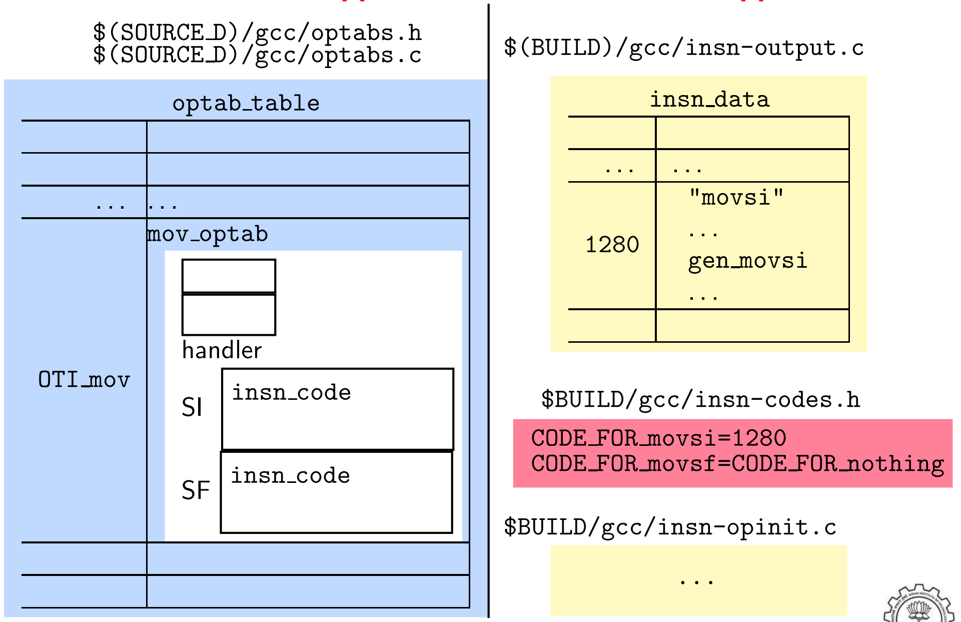
Operation Table

Notes

Instruction Data

- One entry for every pattern defined in .md file
- `struct insn_data_d`
  - Name
  - Information about assembly code generation
    - Single string
    - Multiple string
    - Function returning the required string
    - No assembly code
  - A gen function (as generated in insn-emit.c)
  - Output format (1=single, 2=multi, 3=function, 0=none).

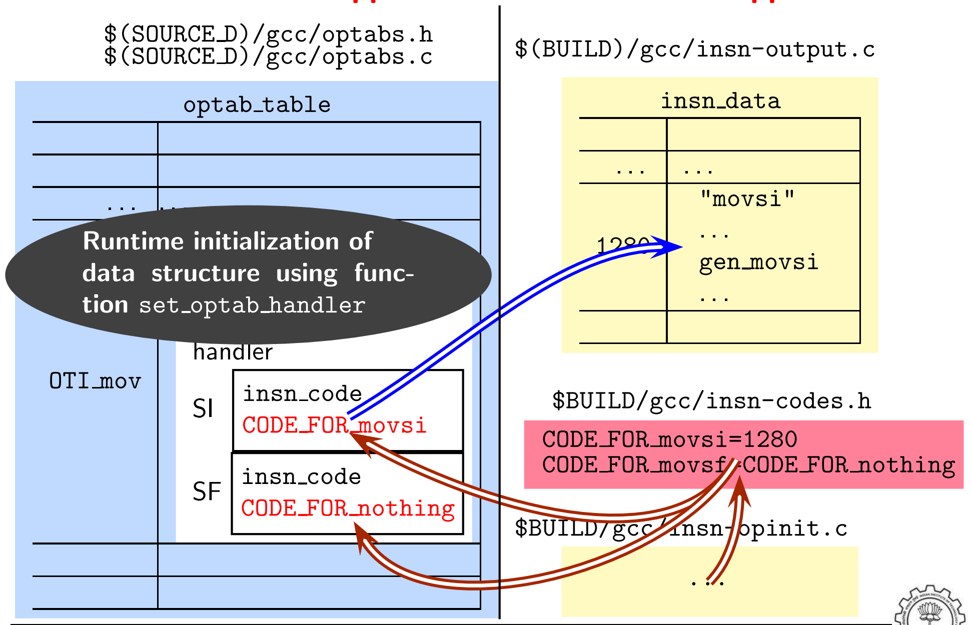
Assume `movsi` is supported but `movsf` is not supported...



Assume movsi is supported but movsf is not supported...

Notes

Assume movsi is supported but movsf is not supported...



Assume movsi is supported but movsf is not supported...

Notes

GCC Generation Phase – Revisited

Generator	Generated from MD	Information	Description
genopinit	insn-opinit.c	void init_all_optabs (void);	Operations Table Initialiser
gencodes	insn-codes.h	enum insn_code = {... CODE_FOR_movsi = 1280, ...}	Index of patterns
genooutput	insn-output.c	struct insn_data [CODE].genfun = /* fn ptr */	All insn data e.g. gen function
genemit	insn-emit.c	rtx gen_rtx_movsi (/* args */) {/* body */}	RTL emission functions

GCC Generation Phase – Revisited

Notes

Explicit Calls to gen<SPN> functions

- In some cases, an entry is not made in insn_data table for some SPNs.
- gen functions for such SPNs are explicitly called.
- These are mostly related to
  - Function calls
  - Setting up of activation records
  - Non-local jumps
  - etc. (i.e. deeper study is required on this aspect)

Explicit Calls to gen<SPN> functions

Notes

Handling C Code in define_expand

```
(define_expand "movsi"
   [( set  (op0) (op1))]
   ""
   "{  /* C CODE OF DEFINE EXPAND */  }")


rtx
gen_movsi (rtx operand0, rtx operand1)
{
   ...
   {
      /* C CODE OF DEFINE EXPAND */
   }
   emit_insn (gen_rtx_ SET  (VOIDmode, operand0, operand1)
   ...
}
```

Handling C Code in `define_expand`

Notes

Part 3

Using the Code Generators

`cc1` Control Flow: GIMPLE to RTL Expansion (`pass_expand`)

```
gimple_expand_cfg
  expand_gimple_basic_block(bb)
     expand_gimple_cond(stmt)
     expand_gimple_stmt(stmt)
         expand_gimple_stmt_1 (stmt)
             expand_expr_real_2
                expand_expr   /* Operands */
                    expand_expr_real
                optab_for_tree_code
                expand_binop /* Now we have rtx for operands */
                   expand_binop_directly
                     /* The plugin for a machine */
                     code=optab_handler(binoptab,mode)
                     GEN_FCN
                     emit_insn
```

Notes

cc1 Control Flow: GIMPLE to RTL Expansion (pass_expand)

Notes

RTL Generation

```
expand_binop_directly
   ... /* Various cases of expansion */
/* One case: integer mode move */
icode = mov_optab->handler[SImode].insn_code
if (icode != CODE_FOR_nothing) {
   ... /* preparatory code */
   emit_insn (GEN_FCN(icode)(dest,src));
}
```

RTL Generation

Notes

RTL to ASM Conversion

- Simple pattern matching of IR RTLs and the patterns present in all named, un-named, standard, non-standard patterns defined using define_insn.
- A DFA (deterministic finite automaton) is constructed and the first match is used.

RTL to ASM Conversion

Notes

Part 4

Conclusions

A Comparison with Davidson Fraser Model

- Retargetability in Davidson Fraser Model
  - Manually rewriting expander and recognizer
  - Simple enough for machines of 1984 era
- Retargetability in GCC

  Automatic construction possible by separating machine specific details in carefully designed data structures
  - List insns as they appear in the chosen MD
  - Index them
  - Supply index to the CGF

Notes

A Comparison with Davidson Fraser Model

Notes