GCC Internals: A Conceptual View – Part II

Abhijat Vichare

CFDVS,
Indian Institute of Technology, Bombay

January 2008

Plan

Part I

- GCC: Conceptual Structure
- C Program through GCC
- Building GCC

Part II

- Gimple
- The MD-RTL and IR-RTL Languages in GCC
- GCC Machine Descriptions

Part I

Gimple

The Goals of GIMPLE are

- Lower control flow
  Program = sequenced statements + unrestricted jump
- Simplify expressions, introduce temporary variables as needed
  Typically: two operand assignments!
- Simplify scope
  move local scope to block begin – including temporaries

Notice

Lowered control flow $\rightarrow$ nearer to register machines + Easier SSA!

Gimple at t_{dev}: GIMPLE code in GCC

Tree manipulation passes (`tree-optimize.c`)

- Gimplifier case analyzes GENERIC nodes, calls corresponding gimplifier.
  {Gimple} = {AST/Generic} - {Control flow nodes}
- Node type specific gimplifiers
- Optimization passes on tree representation, and
- Translate to next IR, i.e. RTL
  - Depth first traverse the "input" Gimple representation
  - Generate a linear list RTL representation

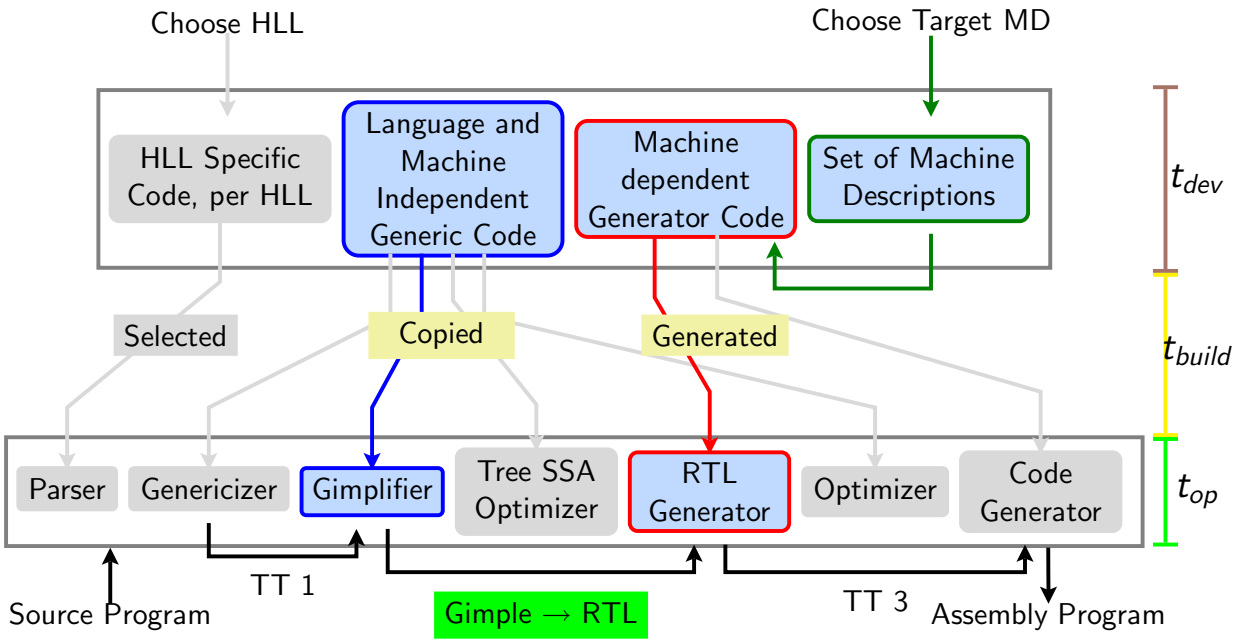
But we have a problem ...

Gimple → RTL Translation Table – Part I

Problem: Gimple (m/c indep.) → RTL (m/c specific)!

To Do : Implement m/c indep. to m/c dep. translation at t_{dev}

Given : the actual target will be known only at t_{build}

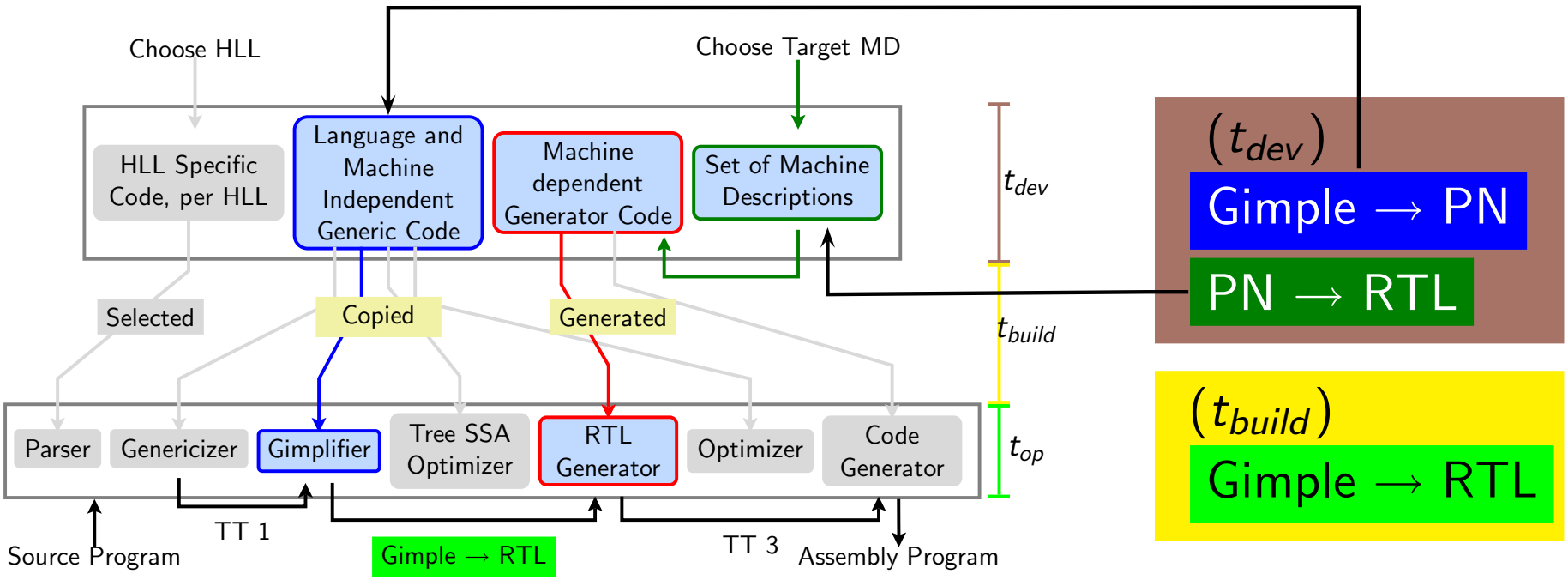


Gimple → RTL Translation Table – Part I

Problem: Gimple (m/c indep.) → RTL (m/c specific)!

To Do : Implement m/c indep. to m/c dep. translation at t_{dev}

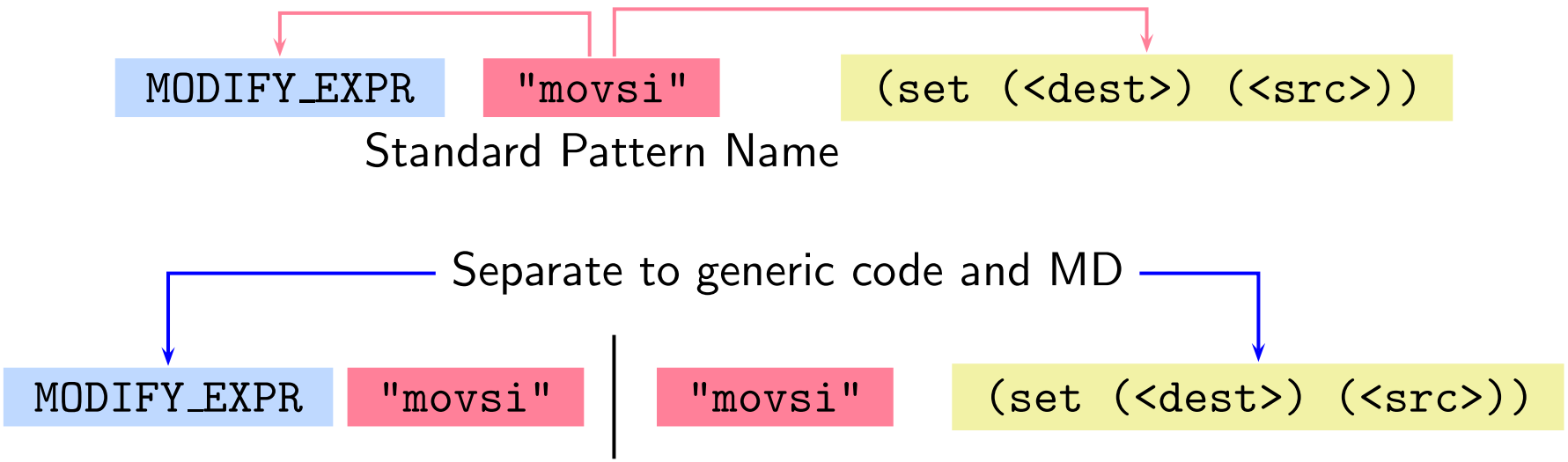
Given : the actual target will be known only at t_{build}

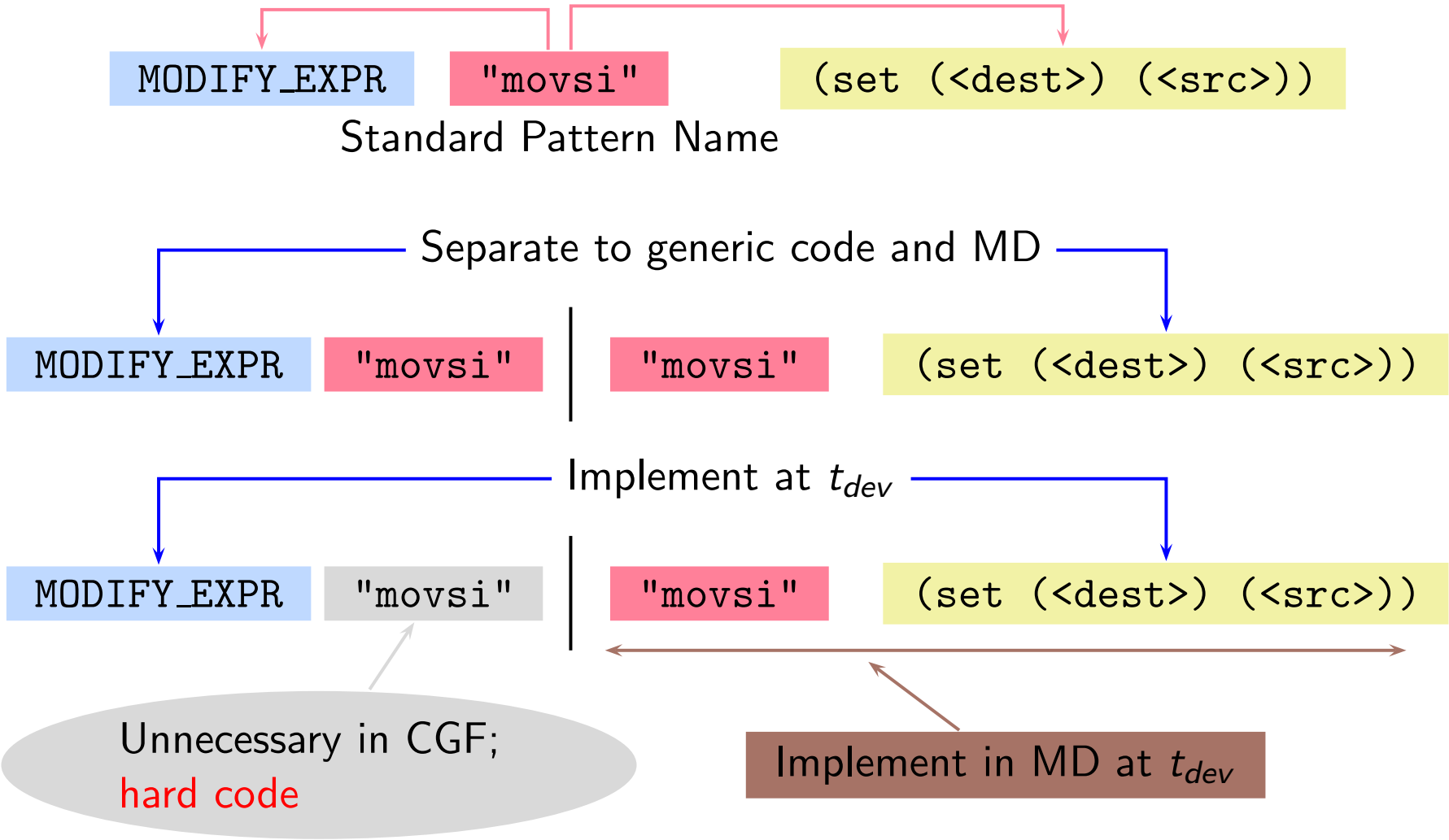


MODIFY_EXPR

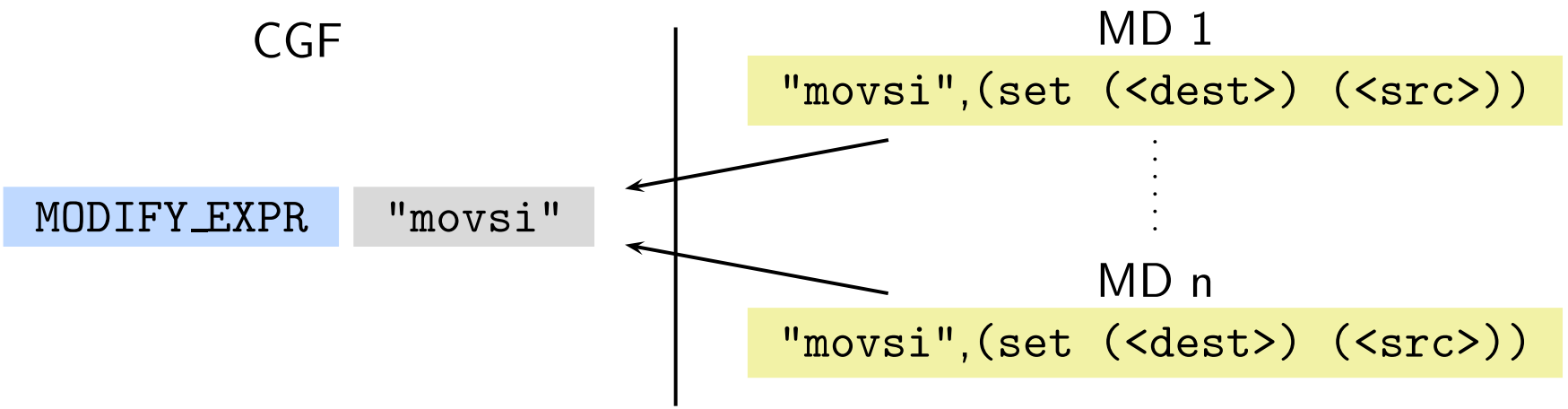
```
(set (<dest>) (<src>))
```


MODIFY_EXPR
"movsi"
(set (<dest>) (<src>))
Standard Pattern Name

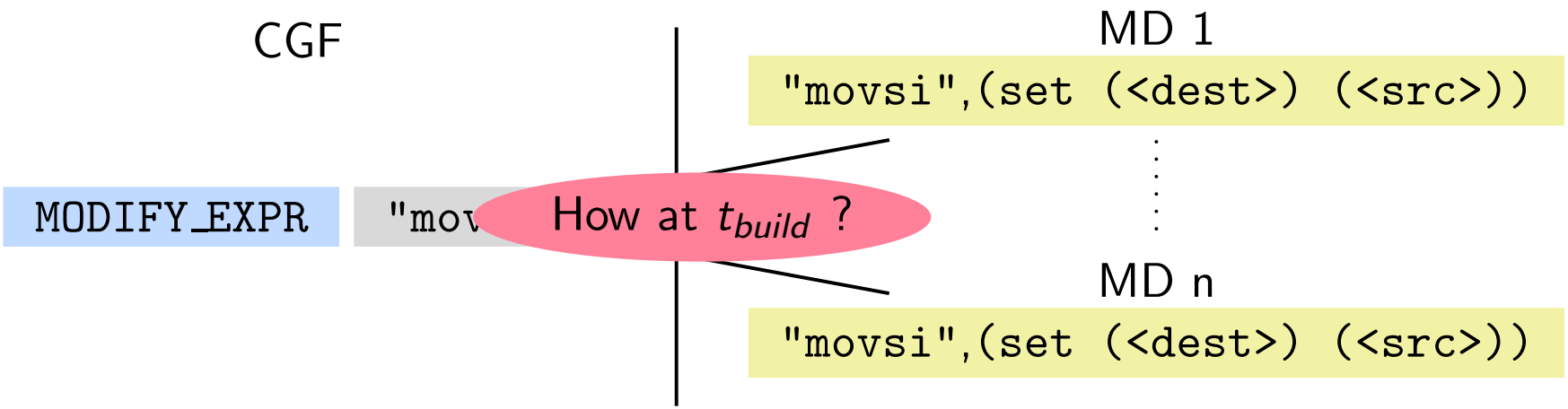

MODIFY_EXPR
"movsi"
(set (<dest>) (<src>))
Standard Pattern Name
Separate to generic code and MD
MODIFY_EXPR
"movsi"
"movsi"
(set (<dest>) (<src>))


MODIFY_EXPR
"movsi"
(set (<dest>) (<src>))
Standard Pattern Name
Separate to generic code and MD
MODIFY_EXPR
"movsi"
"movsi"
(set (<dest>) (<src>))
Implement at t_{dev}
MODIFY_EXPR
"movsi"
"movsi"
(set (<dest>) (<src>))
Unnecessary in CGF;
hard code
Implement in MD at t_{dev}

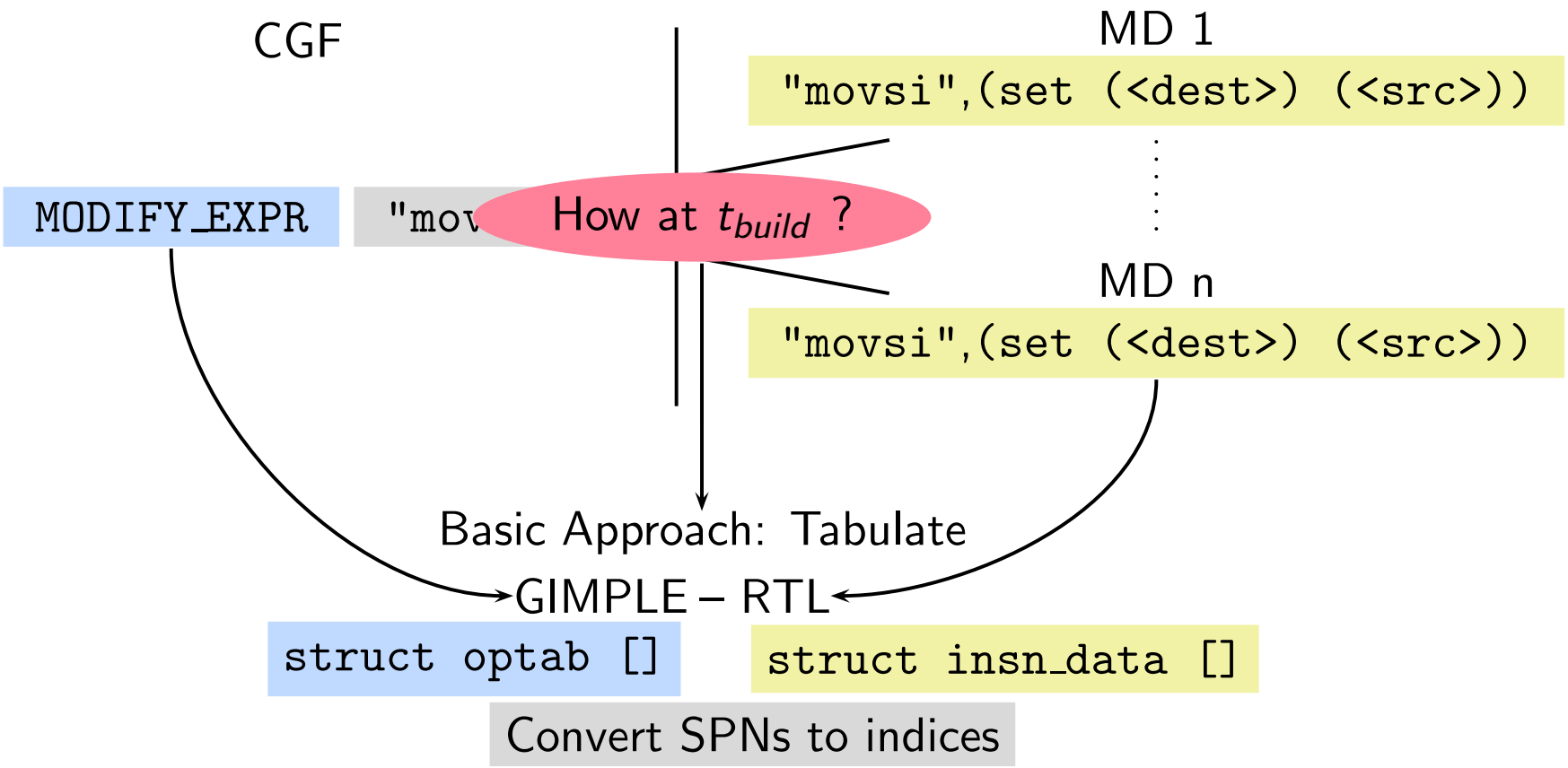
Retargetability ⇒ Multiple MD vs. One CGF! (3:1:10)



Retargetability ⇒ Multiple MD vs. One CGF! (3:1:10)



Retargetability ⇒ Multiple MD vs. One CGF! (3:1:10)



Part II

The MD-RTL and IR-RTL Languages in GCC

Goal 1: Specify target instruction semantics at t_{dev}

Capture target instruction semantics as RTXs

Use MD constructs and operators $\rightarrow$ The MD-RTL Language

Goal 2: Represent input program at t_{op}

- Lower data
- "Express" the "captured" target semantics in IR
- Goal: Every RTX of last RTL pass = unique ASM string.

Use IR constructs and operators $\rightarrow$ The IR-RTL Language

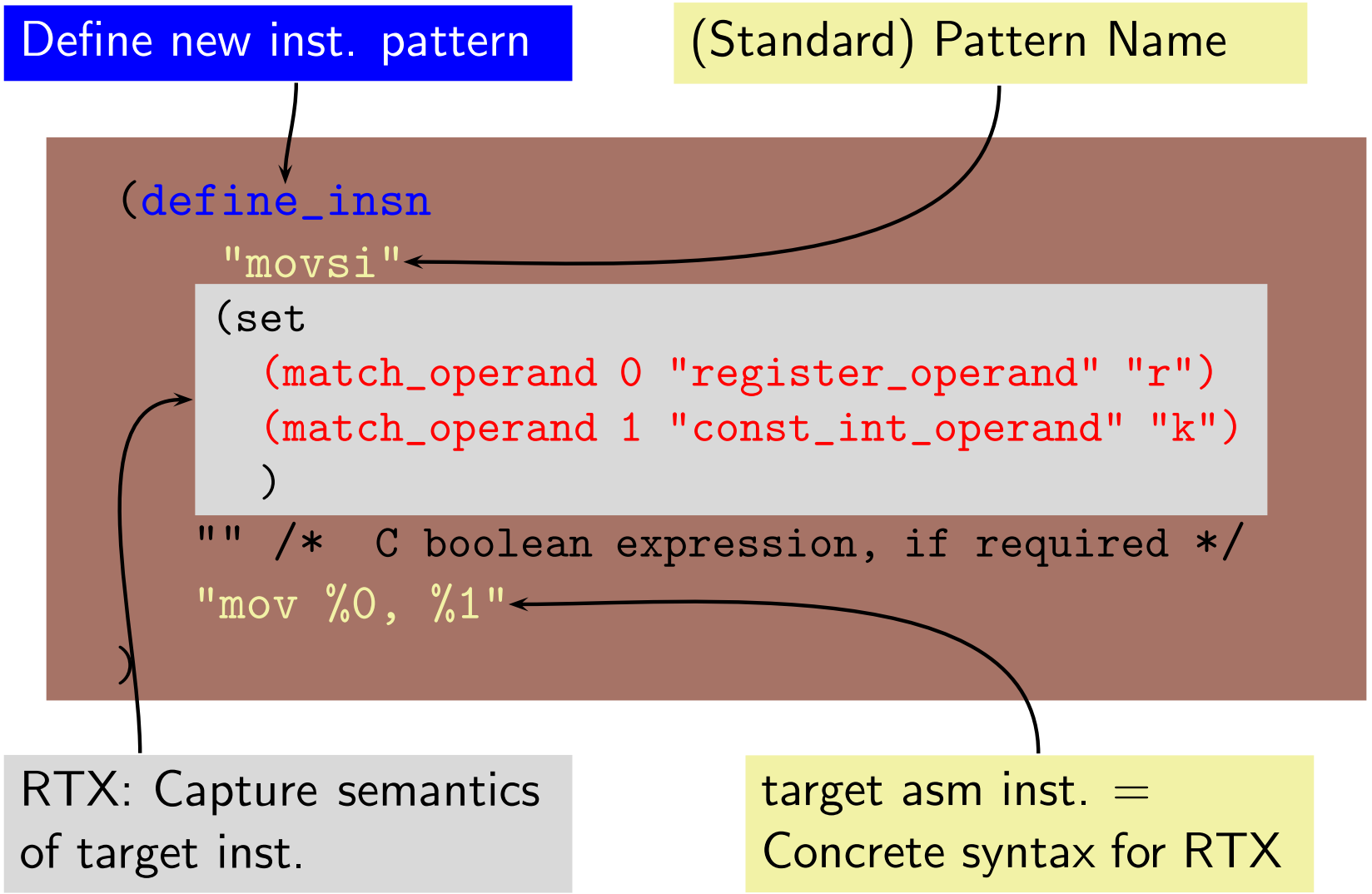
Notice

Lowered data and procedures $\rightarrow$ nearer to typical hardware

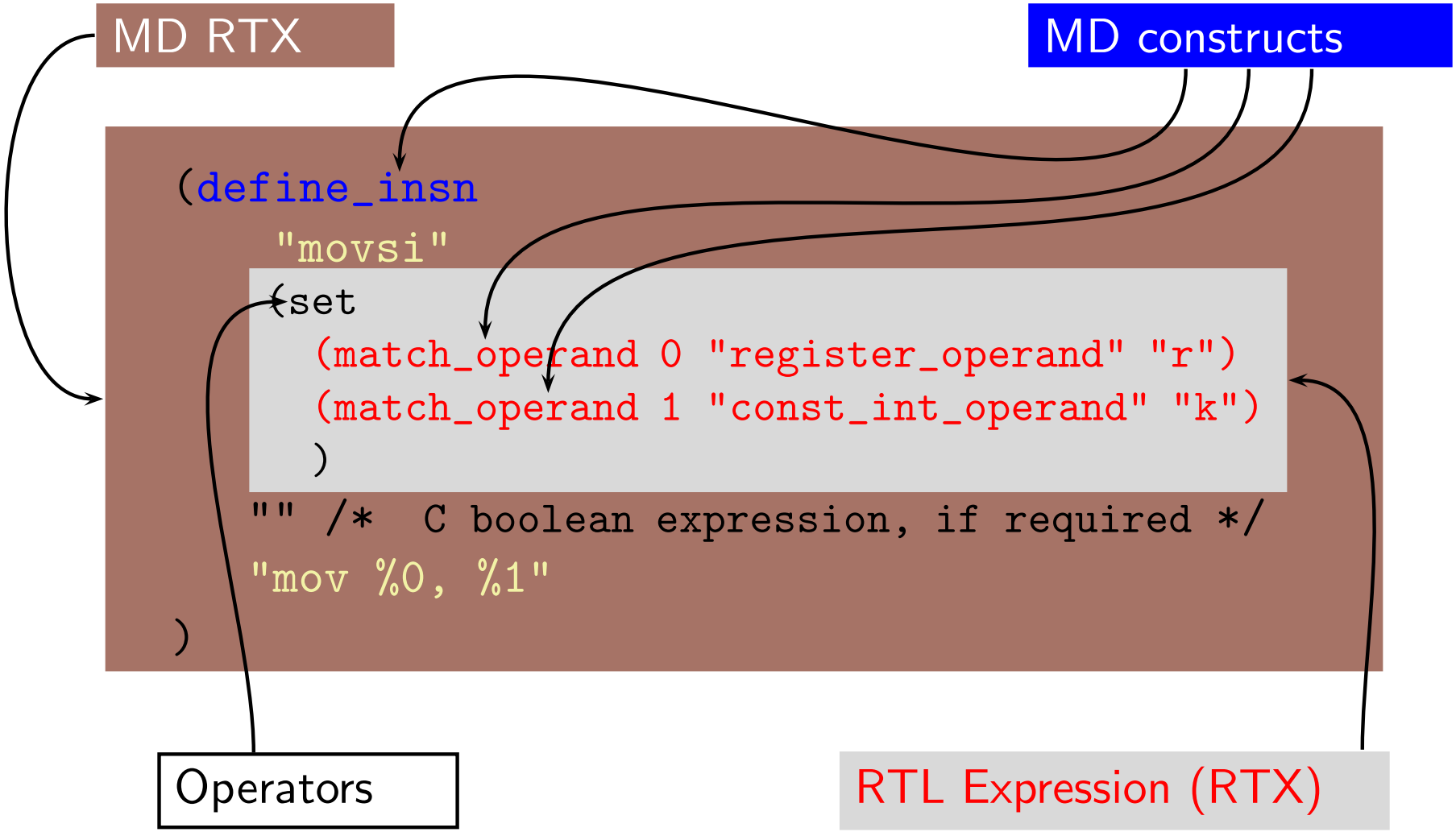
Goal 1 Example: Specifying target inst. semantics

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

Goal 1 Example: Specifying target inst. semantics



Goal 1 Example: Specifying target inst. semantics



From Goal 1 to Goal 2: RTL at t_{build}

Alert: At t_{build}

Convert MD-RTL at $t_{develop}$ to RTL data structures, and compile.

The Data Structure for RTL Objects (In `rtl.h`)

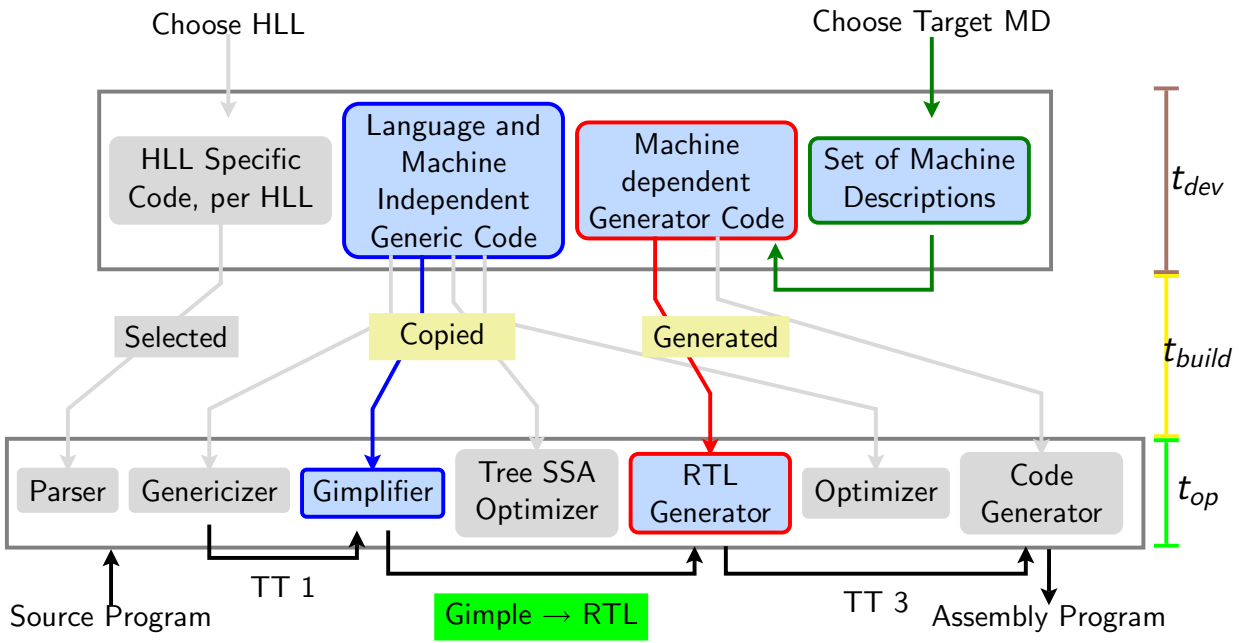
```
struct rtx_def {  /* RTL codes (e.g. SET)  enum-med
   from $GCCHOME/gcc/rtl.def */
  ENUM_BITFIELD(rtx_code)        code : 16;
  ENUM_BITFIELD(machine_mode)    mode : 8;
  unsigned int                   jump : 1;
  /* ... a few such flags */
  union u {
    rtunion          fld[1];
    HOST_WIDE_INT    hwint[1];
  };
};
```

Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form

```
(define_insn
    "movsi"
    ( set  (op0) (op1))
    ""
    "mov %0, %1")
```

t_{dev}



Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form

```
(define_insn
    "movsi"
    ( set  (op0) (op1))
    ""
    "mov %0, %1")
```

t_{dev}

```
rtx
 gen_movsi  (rtx operand0, rtx operand1)
{
    ...
    emit_insn (gen_rtx_ SET  (VOIDmode, op0, op1));
    ...
}
```

t_{build}

Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form

```
(define_insn
   "movsi"
   ( set  (op0) (op1))
   ""
   "mov %0, %1")
```

t_{dev}

```
rtx
gen_movsi  (rtx operand0, rtx operand1)
{
   ...
   emit_insn (gen_rtx_ SET  (VOIDmode, op0, op1));
   ...
}
```

t_{build}

Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form

```
(define_insn
    "movsi"
    ( set  (op0) (op1))
    ""
    "mov %0, %1")
```

t_{dev}

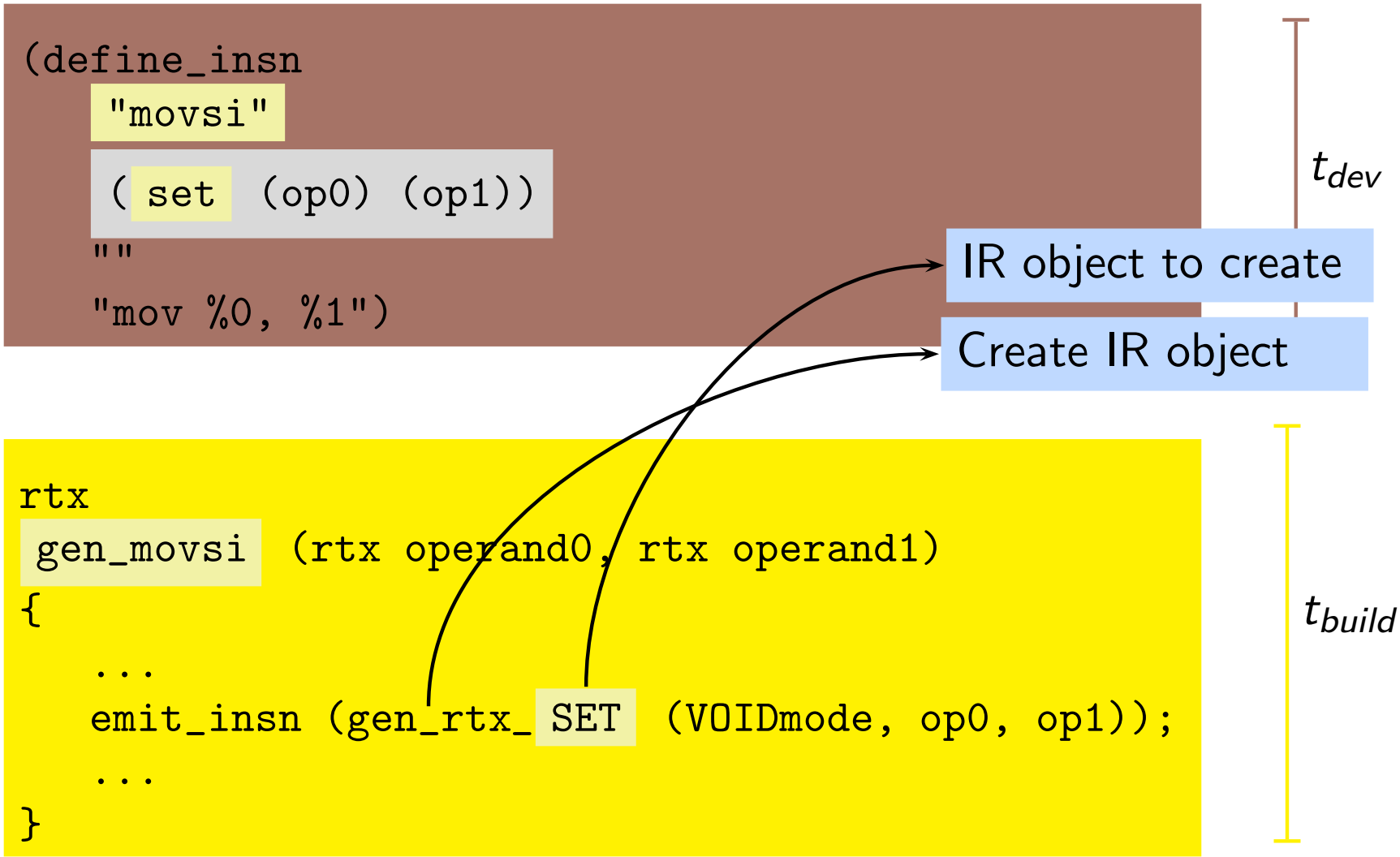
IR object to create

```
rtx
 gen_movsi  (rtx operand0, rtx operand1)
{
    ...
    emit_insn (gen_rtx_ SET  (VOIDmode, op0, op1));
    ...
}
```

t_{build}

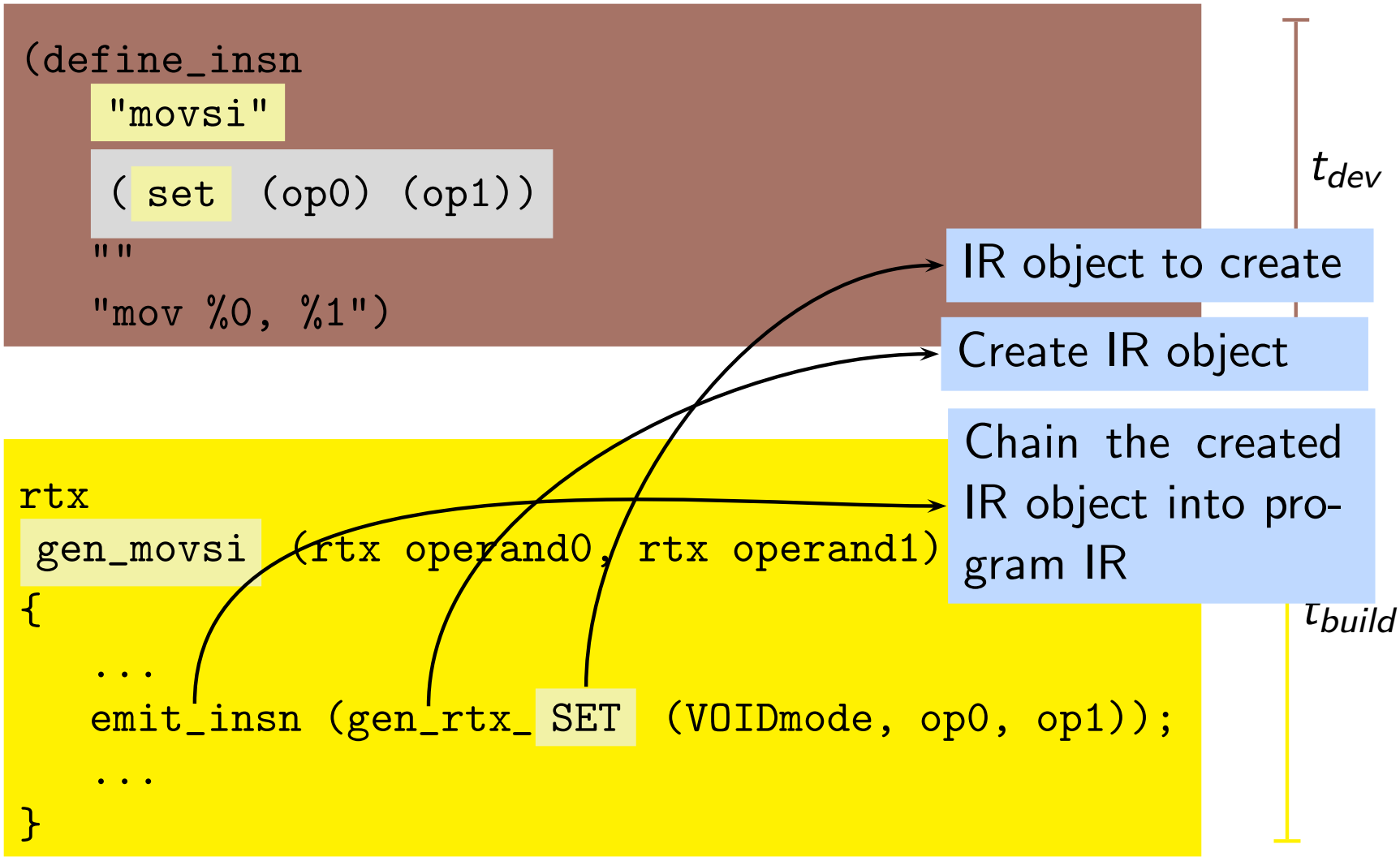
Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form



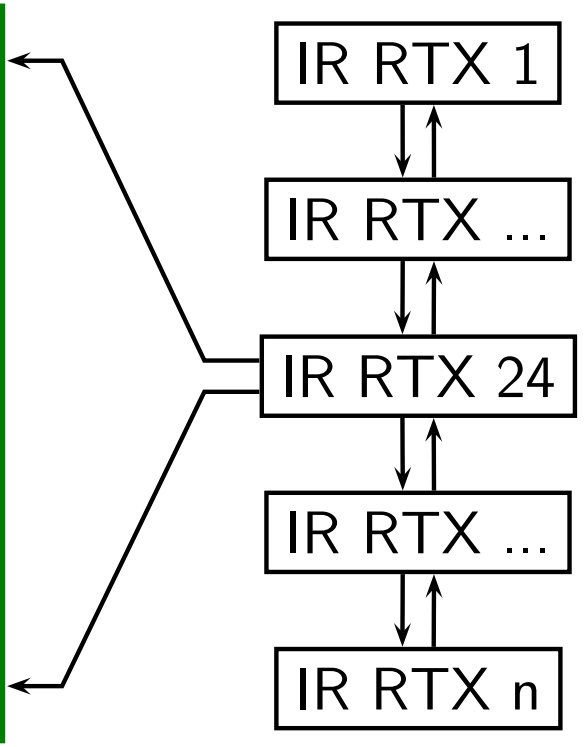
Goal 1 to Goal 2: Gimple → RTL Translation Table

Conversion of RTL from Textual to Internal Form



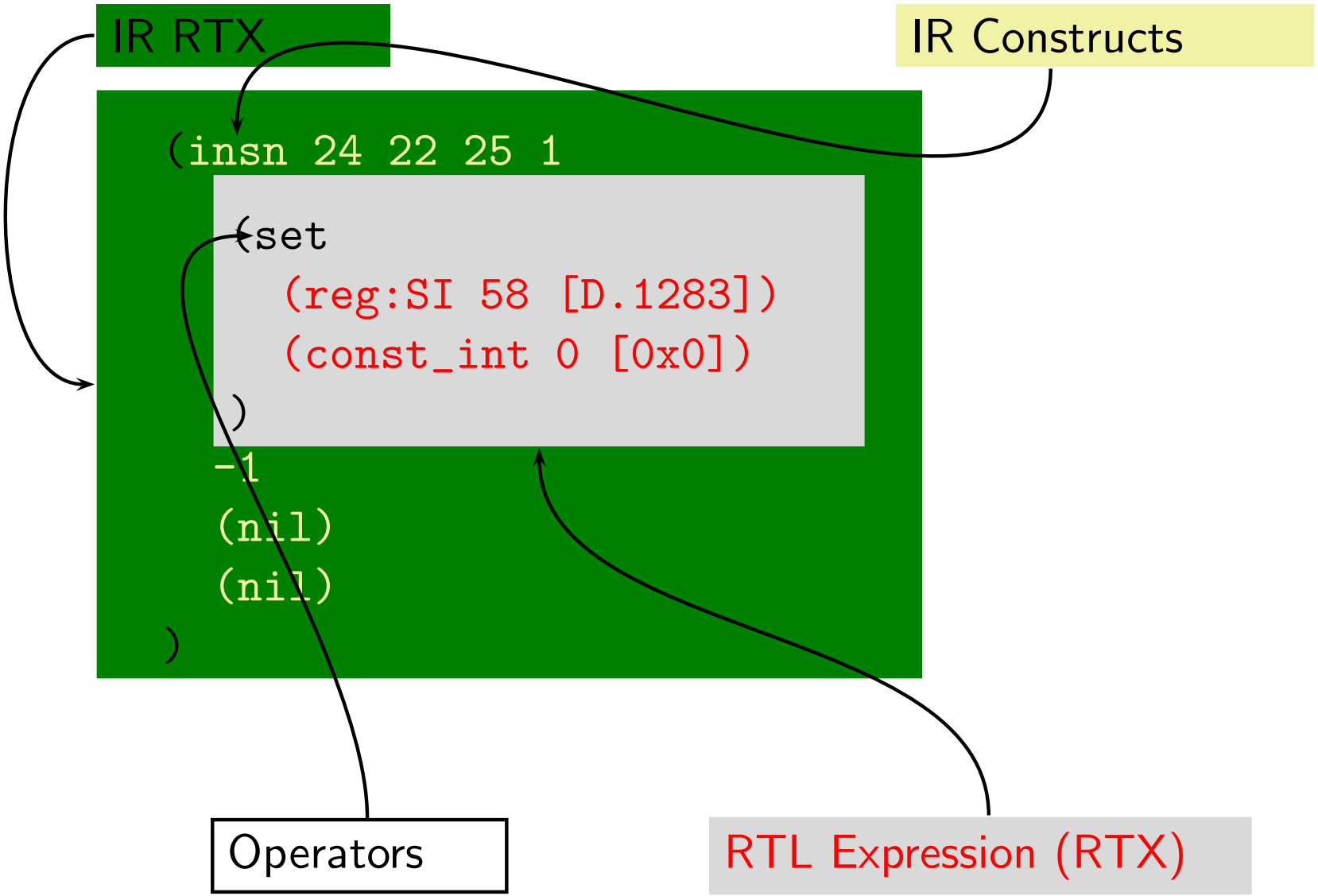
RTL at t_{op}: Example of RTL in use as IR

```
(insn 24 22 25 1
    (set
      (reg:SI 58 [D.1283])
      (const_int 0 [0x0])
    )
   -1
   (nil)
   (nil)
)
```

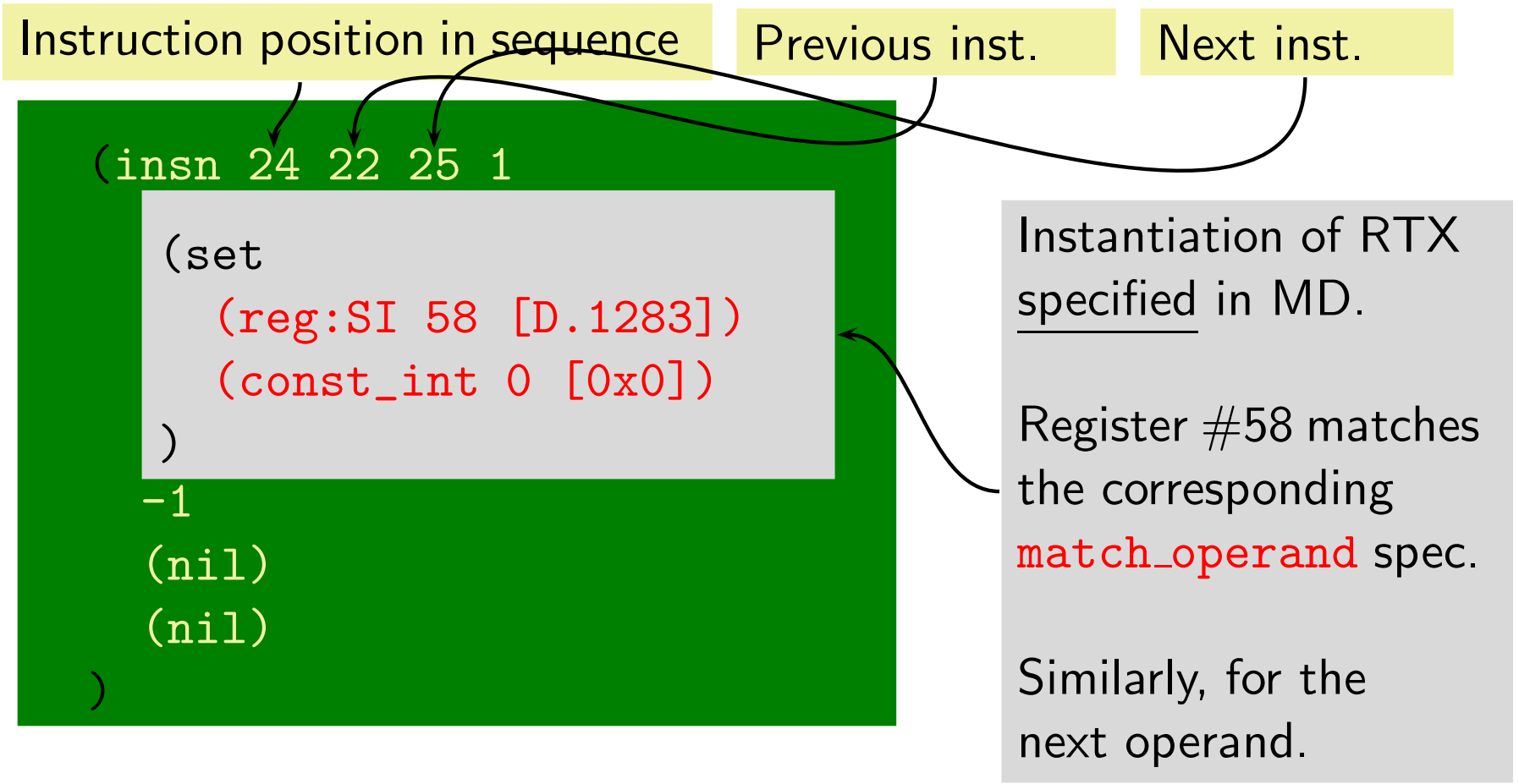


See: RTL dump in “C Program through GCC”

RTL at t_{op}: Example of RTL in use as IR



RTL at t_{op}: Example of RTL in use as IR



RTL at t_{op}: Example of RTL in use as IR

```
(insn 24 22 25 1
    (set
      (reg:SI 58 [D.1283])
      (const_int 0 [0x0])
    )
    -1
    (nil)
    (nil)
)
```

Note:
RTX is incomplete.

Hard register for pseudoregister #58 unknown (yet).
Allocate: #58 = eax

RTX is now complete.

Instantiated RTX is target dependent because the RTX specified in MD captures target instruction semantics.

Part III

GCC Machine Descriptions

Why MD at t_{dev}?

MDs are written in MD-RTL

Main Purpose: Gimple $\longrightarrow$ IR-RTL Translation at t_{run}

Gimple	**IR-RTL**
m/c indep.	m/c specific.
Known at t_{dev}	Unknown at t_{dev}, known at t_{build}
Single "instance"	Per target "instance"

Other Purposes

- IR-RTL Manipulations
  - Gimple $\longrightarrow$ IR-RTL
  - IR-RTL $\longrightarrow$ IR-RTL (combine or split)
  - IR-RTL $\longrightarrow$ Target ASM
- Support
  - Programmer support through convenience constructs
  - Additional heuristics for Instruction selection

The Information in MD

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

Syntactic Entities in GCC MD

Two kinds:

- Necessary Specifications
  - Processor instructions useful to GCC
    - One Gimple → One IR-RTL — `define_insn`
    - One Gimple → More than one IR-RTL — `define_expand`
  - Processor characteristics useful to GCC — `define_cpu_unit`
  - Target ASM syntax — part of `define_insn`
  - IR-RTL → IR-RTL transformations — `define_split`
  - Target Specific Optimizations — `define_peephole2`
- Programming Conveniences
  - `define_insn_and_split`
  - `define_constants`
  - `define_cond_exec`
  - `define_automaton`

Tip

See: `$GCCHOME/gcc/rtl.def`.

Reading a GCC MD

Focus on the RTX!

- Target semantics are in the RTX!
- Target ASM syntax is one argument of `define_insn`
- The set of `define_insns` must be "complete".
- All IR-RTL manipulations must have a `define_insn`.

Example

```
(define_insn "trap"  ; "trap" pattern in  i386.md
  [(trap_if (const_int 1) (const_int 5))]
  "" "int $5")

(define_insn "trap"  ; "trap" pattern in  mips.md
  [(trap_if (const_int 1) (const_int 0))]
  "" "break 0")  ; some target asm details ignored
```