

Retargetting GCC

Abhijat Vichare

(Contact: amv@cfdvs.iitb.ac.in)

CFDVS,
Indian Institute of Technology, Bombay



June 2007

Outline

- GCC MD: General issues
 - ▶ Why ?
 - ▶ Target independence to target specific

- Techniques of introducing target instructions

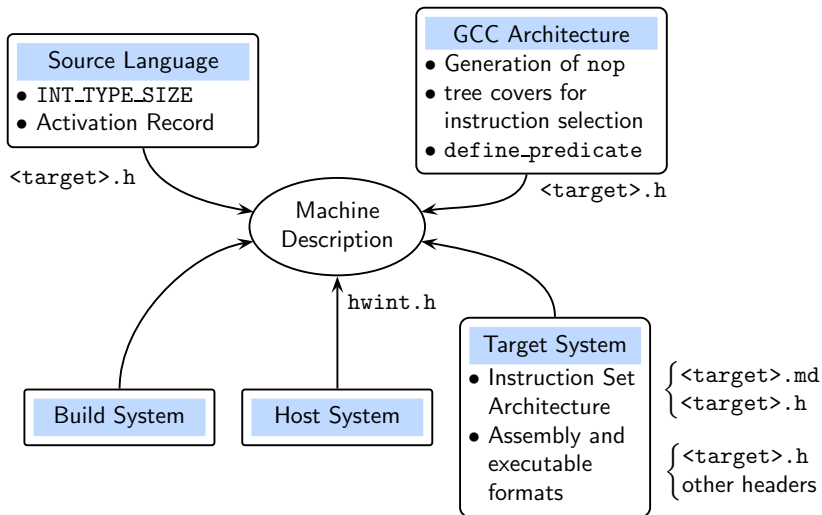
- Techniques of specifying other properties



Part 1

MD: General Issues

Examples of Influences on the Machine Descriptions

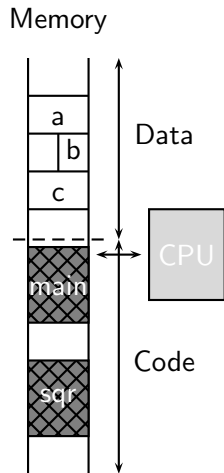


HLL Influences on Machine Description

```
int main ()
{
    int a;
    char b;
    float c;

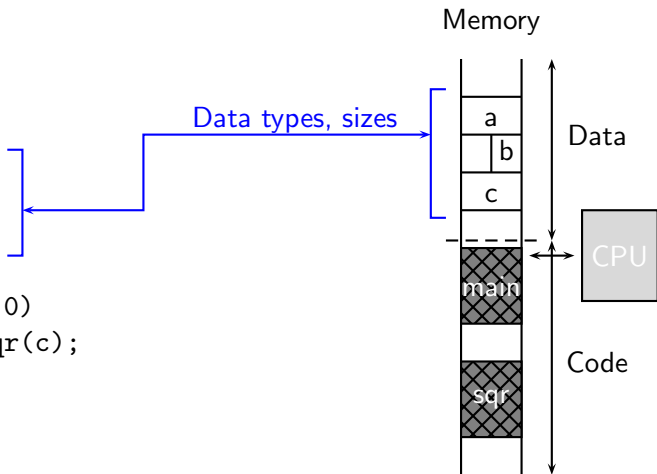
    while (a != 0)
        a = b + sqr(c);

    return 0;
}
```

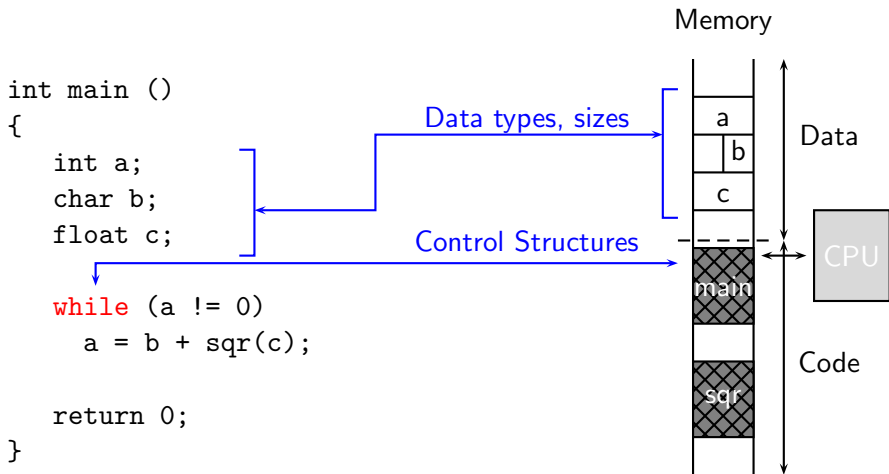


HLL Influences on Machine Description

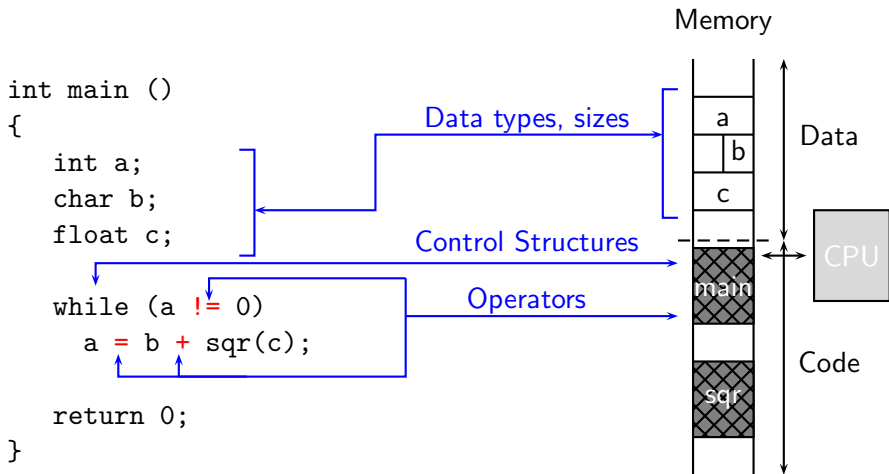
```
int main ()  
{  
    int a;  
    char b;  
    float c;  
  
    while (a != 0)  
        a = b + sqr(c);  
  
    return 0;  
}
```



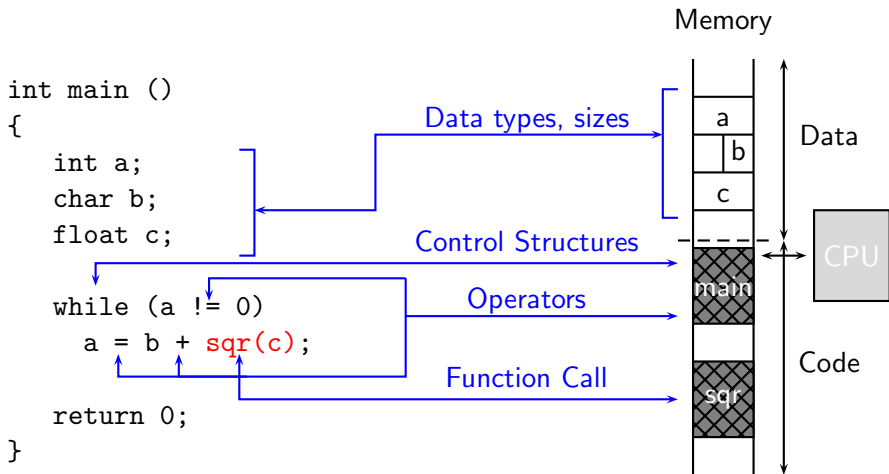
HLL Influences on Machine Description



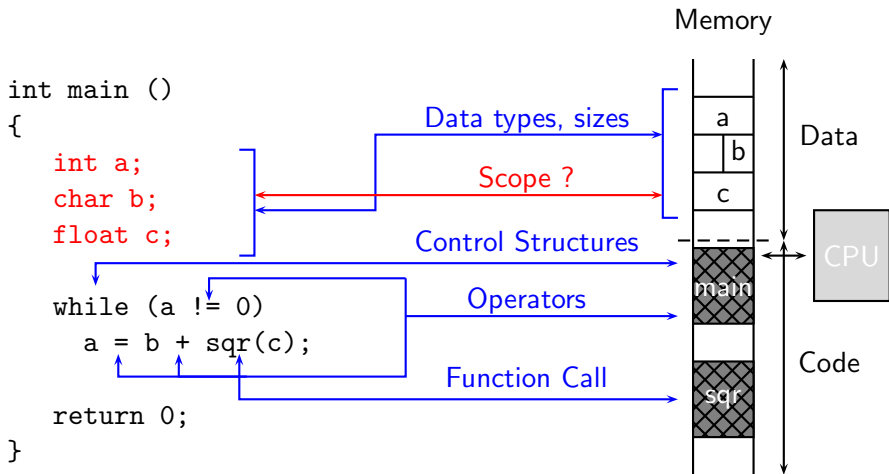
HLL Influences on Machine Description



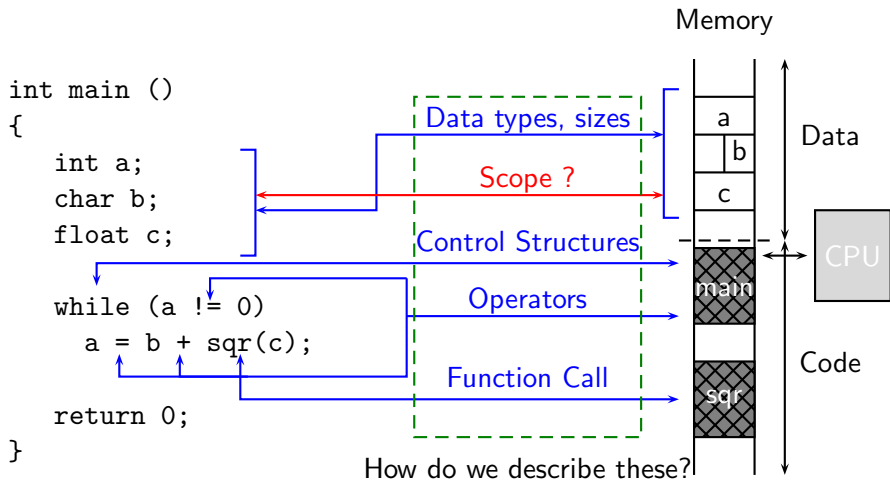
HLL Influences on Machine Description



HLL Influences on Machine Description



HLL Influences on Machine Description



GCC Architecture Influence on MD

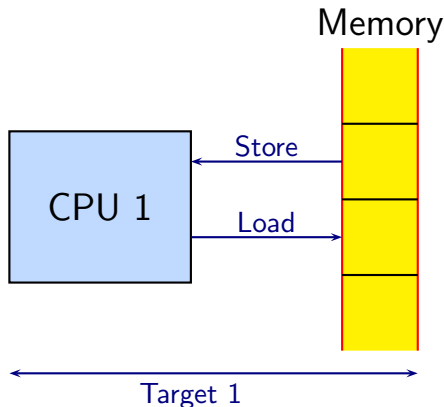
- Standard pattern names with predefined semantics are used in MD
- A new Standard Pattern Name – e.g. `cbranch`
- A new MD RTL – e.g. `define_predicate`
- Macros to be added, removed, or changed in future!



Target Influences on MD

NOTE: Target System = Target ISA + Target System Software

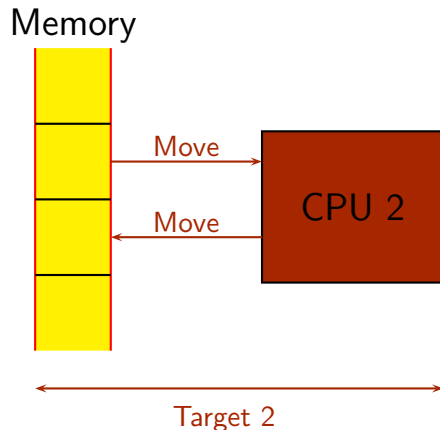
Illustration of ISA Influence



Target Influences on MD

NOTE: Target System = Target ISA + Target System Software

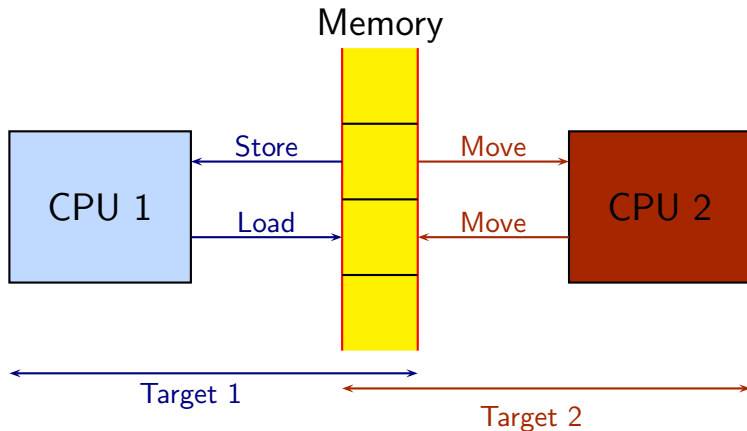
Illustration of ISA Influence



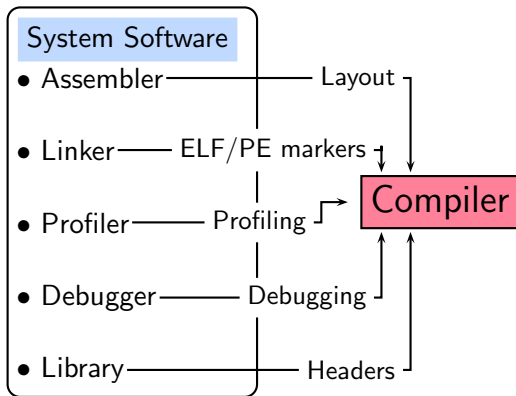
Target Influences on MD

NOTE: Target System = Target ISA + Target System Software

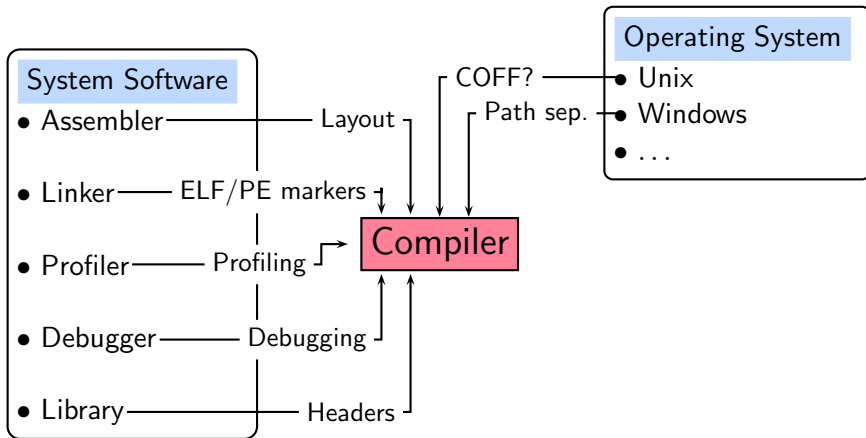
Illustration of ISA Influence



System Influences



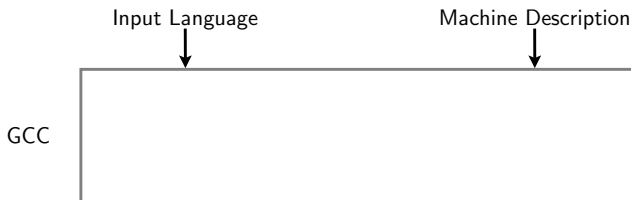
System Influences



Part 2

Phase Sequence and MD

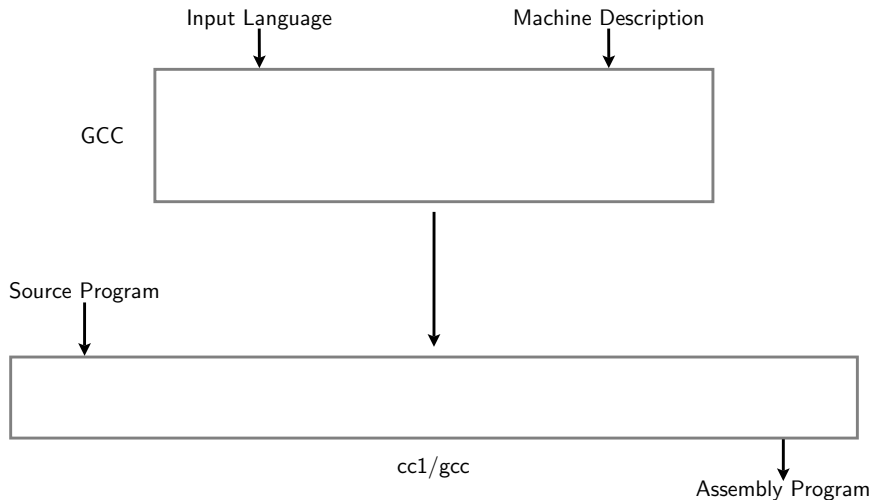
Compiler Generation Framework: Summary



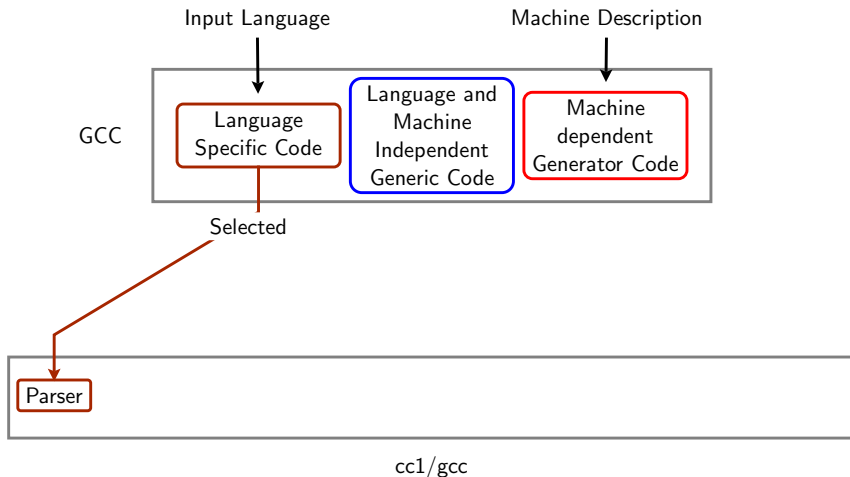
cc1/gcc



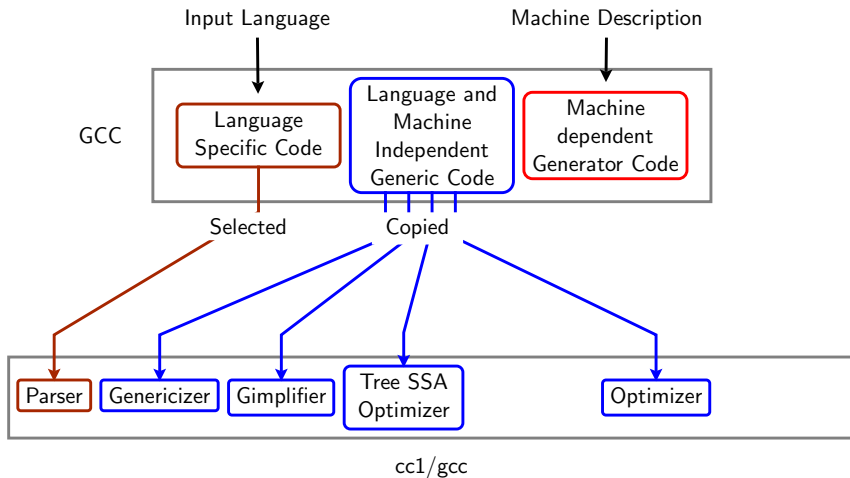
Compiler Generation Framework: Summary



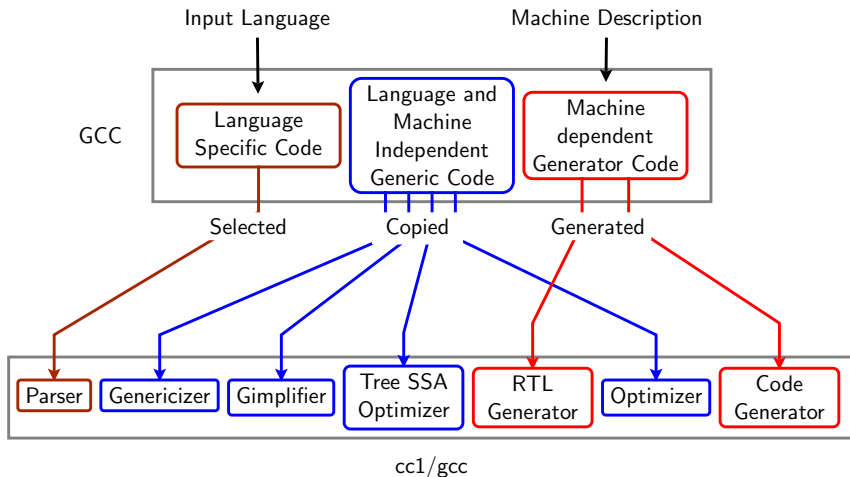
Compiler Generation Framework: Summary



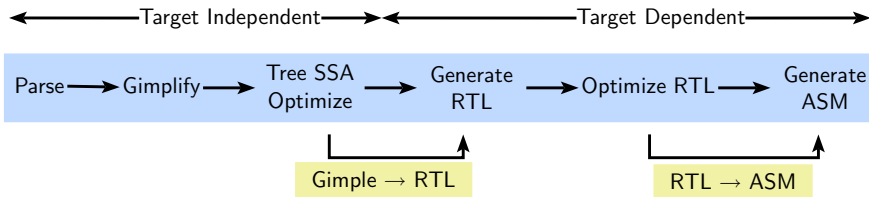
Compiler Generation Framework: Summary



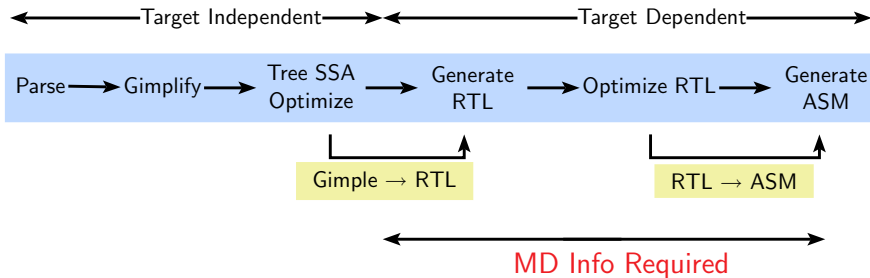
Compiler Generation Framework: Summary



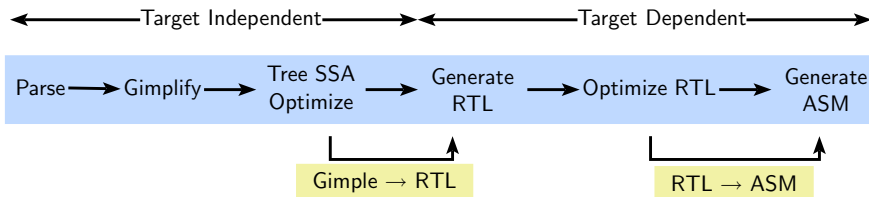
The GCC Phase Sequence



The GCC Phase Sequence



<target>.md ISA Specification Syntax: An Introduction



```
(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"
)
```



<target>.md ISA Specification Syntax: An Introduction

← Target Independent → Target Dependent →



- GIMPLE: target independent
- RTL: target dependent

Gimple → RTL

RTL → ASM

- **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")))
  "" /* C boolean expression, if required */
  "mov %0, %1"
)
  
```



<target>.md ISA Specification Syntax: An Introduction

← Target Independent → Target Dependent →



- GIMPLE: target independent
- RTL: target dependent

Gimple → RTL

RTL → ASM

- **Need:** associate the semantics

⇒ GCC Solution: **Standard Pattern Names**

RTL Template

ASM

MODIFY_EXPR

```

(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"
)
  
```



Part 3

Organisation of GCC MD

GCC MD File Contents

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



GCC MD File Contents

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 RTL.
(Our main interest!)
- **<target>.c**: Optional, but usually required.
C functions that implement target specific code (e.g. target specific activation layout).



The <target>.md File

```
;; 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,%1
  sw \t%1, %m0"
)
```

The Register Class letter code



The <target>.h File

```

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

#define REG_CLASS_CONTENTS \
{0x00000000, 0x00000001, 0x0200ff00, 0x00ff0000, \
 0xf2ffffffc, 0xffffffffe, 0xffffffff}

```


The Register Classes

The Register Class Enumeration



The <target>.c File

```
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



Part 4

Basic Techniques in GCC MD

Recapitulate

We have seen that

- RTL is a **target specific** IR
- GIMPLE \rightarrow non strict RTL \rightarrow strict RTL.
- **SPN**: “(Semantic) Glue” between GIMPLE and RTL
- $S_m C^{T_1}$ **selects** in two phases
 - ▶ operator match + coarse operand match, and
 - ▶ refine the operand match
- **Finally**: Strict RTL \Leftrightarrow Unique target ASM string

To start: Consider generating RTL expressions of GIMPLE nodes

Two constructs available: `define_insn` & `define_expand`



Running Example

Consider a data move operation

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

We will consider the following cases:

- Target has simple mov instruction
No need to carefully examine MODIFY_EXPR operands!
- Target "mov" requires source to be a register!
Carefully examine MODIFY_EXPR operands **before** emitting the RTX!



The define_insn Construct

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



The define_insn Construct

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

Standard Pattern Name _____



The define_insn Construct

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

RTL Template



The define_insn Construct

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

Operand Matchers



The define_insn Construct

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

Operand nos. _____



The define_insn Construct

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

Predicates



The define_insn Construct

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

Constraints



The define_insn Construct

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

C Condition



The define_insn Construct

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

Target ASM string



Use of the `define_insn` Construct

The `define_insn` just defined

- Could instantiate an RTX during GIMPLE \rightarrow RTL as:

```
(set (reg 24) (reg 37))
```

where

- ▶ (reg 24) is an instance of the first `match_operand`, and
- ▶ (reg 37), an instance of the second.
- Can generate data movement patterns of all combinations e.g. `reg \rightarrow reg`, `reg \rightarrow mem`, `mem \rightarrow reg`, `reg \rightarrow const` etc.
- **Even `mem \rightarrow mem` is possible.**

We need a mechanism to:

Generate more restricted data movement RTX instances!

(e.g. Target “`mov`” requires source to be a register.)



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 (!no_new_pseudos)  
        operands[1] = force_reg (SImode, operands[1]);  
  }  
)
```



Part 5

Specifying Other Properties

Defining Attributes

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

Property: Pipelining

Need: To classify target instructions

Construct: `define_attr`



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") )
```



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,



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,



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,



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.



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!



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”



A Few 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_peephole**: (Old) Peephole optimization over insns that substitutes target ASM text.
- **define_peephole2**: (New) Peephole optimization over insns that substitutes insns. Run after register allocation, and before scheduling.
- **define_constants**: Use literal constants in rest of the MD.



Part 6

Summary

Summary

- GCC MD is influenced by:
The HLLs, GCC architecture, and properties of target, host and build systems.
- Retargetting GCC: 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.
- Many other constructs are used.
For example, `define_attr` - `set_attr` - `eq_attr` allow us to create, set and use attributes that can describe the system.



THANK YOU