Introduction to RTL

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

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

March 2010
Outline

- RTL: The Overall Perspective
- RTL: An External View
- RTL: An Internal View
- RTL: An Example Program to Manipulate RTL
Part 1

RTL: The Overall Perspective
What is RTL?

RTL = Register Transfer Language

Assembly language for an abstract machine with infinite registers
Why RTL?

A lot of work in the back-end depends on RTL. Like,

- Low level optimizations like loop optimization, loop dependence, common subexpression elimination, etc
- Instruction scheduling
- Register Allocation
- Register Movement
Why RTL?

For tasks such as those, RTL supports many low level features, like,

- Register classes
- Memory addressing modes
- Word sizes and types
- Compare and branch instructions
- Calling Conventions
- Bitfield operations
The Dual Role of RTL

- For specifying machine descriptions
  Machine description constructs:
  - `define_insn`, `define_expand`, `match_operand`
- For representing program during compilation
  IR constructs
  - `insn`, `jump_insn`, `code_label`, `note`, `barrier`
Role of Machine Descriptions in Translation

Target Independent → Target Dependent

Parse → Gimplify → Tree SSA → Optimize → Generate RTL → Optimize RTL → Generate ASM

Gimple → RTL

RTL → ASM
Role of Machine Descriptions in Translation

Target Independent → Target Dependent

Parse → Gimplify → Tree SSA → Generate RTL → Optimize RTL → Generate ASM

Gimplify → RTL

RTL → ASM

MD Info Required

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Generating RTL IR During Compilation

C Source Code

- Parser
  - AST
- Gimplifier
  - Gimple
- Linearizer
  - Lower
  - CFG Generator
    - CFG
    - RTL Generator
      - RTL expand

local reg allocator

- Iregs
- global reg allocator
  - Gregs
  - pro_epilogue generation
    - prologue-epilogue
    - Pattern Matcher
      - ASM Program

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
A Target Instruction in Machine Descriptions

```c
(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"
)
```
Define instruction pattern

(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"
)

RTL Expression (RTX):
Semantics of target instruction

target asm inst. =
Concrete syntax for RTX
A Target Instruction in Machine Descriptions

```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"
)
```
An Example of Translation

(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"
)
An Example of Translation

```lisp
(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"
)
```

D.1283 = 10;

```lisp
(set
  (reg:SI 58 [D.1283])
  (const_int 10: [0xa])
)
```

li $t0, 10
The Essence of Retargetability

When are the machine descriptions read?
The Essence of Retargetability

When are the machine descriptions read?

- During the build process
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
Retargetability Mechanism of GCC

Input Language

Compiler Generation Framework

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Selected

Copied

Copied

Generated

Generated

Selected

Copied

Copied

Generated

Generated

Essential Abstractions in GCC
Retargetability Mechanism of GCC

Essential Abstractions in GCC
Retargetability Mechanism of GCC

Input Language

- Language Specific Code
- Language and Machine Independent Generic Code

Compiler Generation Framework

- Machine Dependent Generator Code
- Machine Descriptions

Target Name

- Gimple → PN
  - PN → IR-RTL
  - IR-RTL → ASM

Generated Compiler

- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Development Time

Build Time

Use Time

Essential Abstractions in GCC
Retargetability Mechanism of GCC

Input Language

- Language Specific Code
- Language and Machine Independent Generic Code

Compiler Generation Framework

- Machine Dependent Generator Code
- Machine Descriptions

Target Name

Selected

- Copied
- Copies

Parser

- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Generated Compiler

Gimple → PN

PN → IR-RTL

IR-RTL → ASM

Development Time

Build Time

Use Time

Gimple → IR-RTL

IR-RTL → ASM

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Retargetability Mechanism of GCC

Compiler Generation Framework

Input Language
- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

Target Name

Development Time
- Gimple → PN
- PN → IR-RTL

Build Time
- IR-RTL → ASM

Use Time
- Gimple → IR-RTL
- IR-RTL → ASM

Generated Compiler

Gimple → PN

PN → IR-RTL

IR-RTL → ASM

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Part 2

RTL: An External View
RTL for i386: Examples

Translation of $a = a + 1$

**Dump file:** test.c.131r.expand

```plaintext
(stack($fp - 4) = stack($fp - 4) + 1 || flags=?
```


*Plus operation computes $fp - 4$ as the address of variable $a*
RTL for i386: Examples

Translation of $a = a + 1$

**Dump file:** test.c.131r.expand

```
(stack($fp - 4) = stack($fp - 4) + 1 || flags=?

(insn 12 11 10 (parallel[
  (set (mem/c/i:SI (plus:SI
    (reg/f:SI 54 virtual-stack-vars)
    (const int -4 [...])) [...])
  (plus:SI
    (mem/c/i:SI (plus:SI
      (reg/f:SI 54 virtual-stack-vars)
      (const int -4 [...])) [...])
    (const int 1 [...])))))
(clobber (reg:CC 17 flags))
]) -1 (nil))
```

*Set denotes assignment*
RTL for i386: Examples

Translation of $a = a + 1$

**Dump file:** test.c.131r.expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 10 (parallel [  
    ( set (mem/c/i:SI (plus:SI  
        (reg/f:SI 54 virtual-stack-vars)  
        (const int -4 [...])) [...]))  
    (plus:SI  
        (mem/c/i:SI (plus:SI  
            (reg/f:SI 54 virtual-stack-vars)  
            (const int -4 [...])) [...]))  
        (const int 1 [...])))))  
(clobber (reg:CC 17 flags))  
]) -1 (nil))
```

*1 is added to variable $a*
RTL for i386: Examples

Translation of $a = a + 1$

**Dump file:** test.c.131r.expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 10 (parallel [ 
   ( set (mem/c/i:SI (plus:SI 
      (reg/f:SI 54 virtual-stack-vars) 
      (const int -4 [...])) [...] )
   (plus:SI 
      (mem/c/i:SI (plus:SI 
         (reg/f:SI 54 virtual-stack-vars) 
         (const int -4 [...])) [...] )
      (const int 1 [...])))))
   (clobber (reg:CC 17 flags))
]) -1 (nil))
```

Condition Code register is clobbered to record possible side effect of plus
Flags in RTL Expressions

Meanings of some of the common flags

/c  memory reference that does not trap
/i  scalar that is not part of an aggregate
/f  register that holds a pointer
RTL for spim: Examples

Translation of $a = a + 1$

**Dump file:** test.c.131r.expand

\[
\text{r39} = \text{stack}($fp - 4) \\
\text{r40} = \text{r39} + 1 \\
\text{stack}($fp - 4) = \text{r40}
\]

\[
\]

\[
\text{insn 8 7 9 test.c:6 (set (reg:SI 40) (plus:SI (reg:SI 39) (const_int 1 [...])) -1 (nil))}
\]

\[
\text{insn 9 8 0 test.c:6 (set (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars) (const_int -4 [...])) [...]) (reg:SI 40)) -1 (nil))}
\]

In spim, a variable is loaded into register to perform any instruction, hence three instructions are generated.
RTL for spim: Examples

Translation of $a = a + 1$

Dump file: test.c.131r.expand

\[
\text{(insn 7 6 8 test.c:6 (set (reg:SI 39))}
\]
\[
\quad \text{(mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars))}
\]
\[
\quad \quad \text{(const_int -4 [...]}) [...]]) -1 (nil))
\]

\[
\text{(insn 8 7 9 test.c:6 (set (reg:SI 40))}
\]
\[
\quad \text{(plus:SI (reg:SI 39}}
\]
\[
\quad \quad \text{(const_int 1 [...]})]) -1 (nil))
\]

\[
\text{(insn 9 8 0 test.c:6 (set}}
\]
\[
\quad \text{(mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars))}
\]
\[
\quad \quad \text{(const_int -4 [...]}) [...])}
\]
\[
\quad \quad \text{(reg:SI 40)) -1 (nil))}
\]

In spim, a variable is loaded into register to perform any instruction, hence three instructions are generated.

\[
r_{39} = \text{stack($fp - 4$)}
\]
\[
r_{40} = r_{39} + 1
\]
\[
\text{stack($fp - 4$)} = r_{40}
\]
RTL for spim: Examples

Translation of \( a = a + 1 \)

**Dump file:** test.c.131r.expand

\[
\begin{align*}
\text{r39} &= \text{stack}($fp - 4) \\
\text{r40} &= \text{r39} + 1 \\
\text{stack}($fp - 4) &= \text{r40}
\end{align*}
\]

(Insn 7 6 8 test.c:6 (set (reg:SI 39)
  (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
    (const_int -4 [...]))) [...] -1 (nil))

(Insn 8 7 9 test.c:6 (set (reg:SI 40)
  (plus:SI (reg:SI 39)
    (const_int 1 [...]))) -1 (nil))

(Insn 9 8 0 test.c:6 (set
  (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
    (const_int -4 [...]))) [...] 
  (reg:SI 40)) -1 (nil))

In spim, a variable is loaded into register to perform any instruction, hence three instructions are generated
What does this represent?

(jump_insn 15 14 16 4 p1.c:6 (set (pc)
 (if_then_else (lt (reg:CCGC 17 flags)
 (const_int 0 [0x0]))
 (label_ref 12)
 (pc))) (nil)
 (nil))
RTL for i386: Examples

What does this represent?

```
(jump_insn 15 14 16 4 p1.c:6 (set (pc)
   (if_then_else (lt (reg:CCGC 17 flags)
      (const_int 0 [0x0]))
    (label_ref 12)
    (pc))) (nil)
    (nil))
```

```
pc = r17 <0 ? label(12) : pc
```
RTL for i386: Examples

Translation of if (a > b)

Dump file: test.c.131r.expand

(insn 8 7 9 test.c:7 (set (reg:SI 61)
   (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
   (const_int -8 [0xffffffff8])) [0 a+0 S4 A32])) -1 (nil))

(insn 9 8 10 test.c:7 (set (reg:CCGC 17 flags)
   (compare:CCGC (reg:SI 61)
   (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
   (const_int -4 [0xffffffffc])) [0 b+0 S4 A32])))) -1 (nil)

(jump_insn 10 9 0 test.c:7 (set (pc)
   (if_then_else (le (reg:CCGC 17 flags)
   (const_int 0 [0x0]))
   (label_ref 0)
   (pc))) -1 (nil))
RTL for i386: Examples

Translation of if (a > b)

**Dump file:** test.c.131r.expand

(insn 8 7 9 test.c:7 (set (reg:SI 61)
    (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
        (const_int -8 [0xfffffffff8])) [0 a+0 S4 A32])) -1 (nil))
(insn 9 8 10 test.c:7 (set (reg:CCGC 17 flags)
    (compare:CCGC (reg:SI 61)
        (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
            (const_int -4 [0xfffffffff0])) [0 b+0 S4 A32])))) -1 (nil))
(jump_insn 10 9 0 test.c:7 (set (pc)
    (if_then_else (le (reg:CCGC 17 flags)
        (const_int 0 [0x0]))
        (label_ref 0)
        (pc)))) -1 (nil))
RTL for i386: Examples

Translation of if (a > b)

**Dump file:** test.c.131r.expand

\[
\text{(insn 8 7 9 test.c:7 (set (reg:SI 61))}
\]
\[
\left( \text{mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)}
\right.
\]
\[
\left( \text{const_int -8 [0xffffffff8]} \right)
\]
\[
\left[ 0 \text{ a+0 S4 A32]} \right)
\]
\[
\right) -1 (\text{nil})
\]

\[
\text{(insn 9 8 10 test.c:7 (set (reg:CCGC 17 flags))}
\]
\[
\left( \text{compare:CCGC (reg:SI 61)}
\right.
\]
\[
\left( \text{mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)}
\right.
\]
\[
\left( \text{const_int -4 [0xffffffffc]} \right)
\]
\[
\left[ 0 \text{ b+0 S4 A32]} \right)
\]
\[
\right) -1 (\text{nil})
\]

\[
\text{(jump_insn 10 9 0 test.c:7 (set (pc))}
\]
\[
\left( \text{if_then_else (le (reg:CCGC 17 flags)}
\right.
\]
\[
\left( \text{const_int 0 [0x0]} \right)
\]
\[
\left( \text{label_ref 0)}
\right.
\]
\[
\left( \text{pc))) -1 (\text{nil})
\right)
\]
Part 3

RTL: An Internal View
 RTL Objects

• Types of RTL Objects
  ▶ Expressions
  ▶ Integers
  ▶ Wide Integers
  ▶ Strings
  ▶ Vectors

• Internal representation of RTL Expressions
  ▶ Expressions in RTX are represented as trees
  ▶ A pointer to the C data structure for RTL is called rtx
RTX Expressions are classified into RTX codes:

- Expression codes are names defined in `rtl.def`
- RTX codes are C enumeration constants
- Expression codes and their meanings are machine-independent
- Extract the code of a RTX with the macro `GET_CODE(x)`
RTL Classes

RTL expressions are divided into few classes, like:

- **RTX_UNARY**: NEG, NOT, ABS
- **RTX_BIN_ARITH**: MINUS, DIV
- **RTX_COMM_ARITH**: PLUS, MULT
- **RTX_OBJ**: REG, MEM, SYMBOL_REF
- **RTX_COMPARE**: GE, LT
- **RTX_TERNARY**: IF_THEN_ELSE
- **RTX_INSN**: INSN, JUMP_INSN, CALL_INSN
- **RTX_EXTRA**: SET, USE
RTX Codes

The RTX codes are defined in rtl.def using cpp macro call DEF_RTL_EXPR, like:

- DEF_RTL_EXPR(INSN, "insn", "iuuBieie", RTX_INSN)
- DEF_RTL_EXPR(SET, "set", "ee", RTX_EXTRA)
- DEF_RTL_EXPR(PLUS, "plus", "ee", RTX_COMM_ARITH)
- DEF_RTL_EXPR(IF_THEN_ELSE, "if_then_else", "eee", RTX_TERNARY)

The operands of the macro are:

- Internal name of the rtx used in C source. It's a tag in enumeration "enum rtx_code"
- name of the rtx in the external ASCII format
- Format string of the rtx, defined in rtx_format[]
- Class of the rtx
RTX Formats

DEFRTL_EXPR(INSN, "insn", "iuuBieie", RTX_INSN)

- i: Integer
- u: Integer representing a pointer
- B: Pointer to basic block
- e: Expression
RTL statements

- RTL statements are instances of type rtx
- RTL insns contain embedded links
- Types of RTL insns:
  - INSN: Normal non-jumping instruction
  - JUMP_INSNN: Conditional and unconditional jumps
  - CALL_INSNN: Function calls
  - CODE_LABEL: Target label for JUMP_INSNN
  - BARRIER: End of control Flow
  - NOTE: Debugging information
Basic RTL APIs

- **XEXP, XINT, XWINT, XSTR**
  - Example: `XINT(x, 2)` accesses the 2nd operand of `rtx x` as an integer
  - Example: `XEXP(x, 2)` accesses the same operand as an expression

- Any operand can be accessed as any type of RTX object
  - So operand accessor to be chosen based on the format string of the containing expression

- Special macros are available for Vector operands
  - `XVEC(exp, idx)`: Access the vector-pointer which is operand number `idx` in `exp`
  - `XVECLEN(exp, idx)`: Access the length (number of elements) in the vector which is in operand number `idx` in `exp`. This value is an int
  - `XVECEXP(exp, idx, eltnum)`: Access element number “eltnum” in the vector which is in operand number `idx` in `exp`. This value is an RTX
RTL Insns

- A function’s code is a doubly linked chain of INSN objects
- Insns are rtxs with special code
- Each insn contains atleast 3 extra fields:
  - Unique id of the insn, accessed by INSN_UID(i)
  - PREV_INSN(i) accesses the chain pointer to the INSN preceeding i
  - NEXT_INSN(i) accesses the chain pointer to the INSN succeeding i
- The first insn is accessed by using get_insns()
- The last insn is accessed by using get_lastInsn()
Part 4

RTL: An Example Program to Manipulate RTL
Problem statement: Counting the number of SET objects in a basic block by adding a new RTL pass

- Add your new pass after `passexpand`
- `new_rtl_pass_main` is the main function of the pass
- Iterate through different instructions in the doubly linked list of instructions and for each expression, call `eval_rtx(insn)` for that expression which recurse in the expression tree to find the set statements
int new_rtl_pass_main(void){
    basic_block bb;
    rtx last, insn, opd1, opd2;
    int bbno, code, type;
    count = 0;
    for (insn=get_insns(), last=get_last_insn(),
         last=NEXT_INSN(last); insn!=last; insn=NEXT_INSN(insn))
    {
        int is_insn;
        is_insn = INSN_P (insn);
        if(flag_dump_new_rtl_pass)
            print_rtl_single(dump_file, insn);
        code = GET_CODE(insn);
        if(code==NOTE){ ... }
        if(is_insn)
            { rtx subexp = XEXP(insn,5);
              eval_rtx(subexp);
            }
    }
    ...
int new_rtl_pass_main(void){
    basic_block bb;
    rtx last, insn, opd1, opd2;
    int bbno, code, type;
    count = 0;
    for (insn=get_insns(), last=get_last_insn(),
        last=NONCE_INSN(last); insn!=last; insn=NONCE_INSN(insn))
    {
        int is_insn;
        is_insn = INSN_P (insn);
        if(flag_dump_new_rtl_pass)
            print_rtl_single(dump_file, insn);
        code = GET_CODE(insn);
        if(code==NOTE){ ... }
        if(is_insn)
            { rtx subexp = XEXP(insn, 5);
              eval_rtx(subexp);
            }
    }
    ...
void eval_rtx(rtx exp)
{
    rtx temp;
    int veclen, i,
    int rt_code = GET_CODE(exp);
    switch(rt_code)
    {
    case SET:
        if(flag_dump_new_rtl_pass){
            fprintf(dump_file, "\nSet statement %d : \t", count+1);
            print_rtl_single(dump_file, exp);
        }
        count++; break;
    case PARALLEL:
        veclen = XVECLEN(exp, 0);
        for(i = 0; i < veclen; i++)
        {
            temp = XVECEXP(exp, 0, i);
            eval_rtx(temp);
        }
        break;
    default: break;
    }
}
void eval_rtx(rtx exp)
{
    rtx temp;
    int veclen, i,
    int rt_code = GET_CODE(exp);
    switch(rt_code)
    {
        case SET:
            if(flag_dump_new_rtl_pass){
                fprintf(dump_file,"\nSet statement %d : \t",count+1);
                print_rtl_single(dump_file,exp);
            }
            count++; break;
        case PARALLEL:
            veclen = XVECLEN(exp, 0);
            for(i = 0; i < veclen; i++)
            {
                temp = XVECEXP(exp, 0, i);
                eval_rtx(temp);
            }
            break;
        default: break;
    }
}
void eval_rtx(rtx exp) {
    rtx temp;
    int veclen,i,
    int rt_code = GET_CODE(exp);
    switch(rt_code) {
        case SET:
            if(flag_dump_new_rtl_pass){
                fprintf(dump_file, "\nSet statement %d : \t",count+1);
                print_rtl_single(dump_file,exp);
            }
            count++; break;
        case PARALLEL:
            veclen = XVECLLEN(exp, 0);
            for(i = 0; i < veclen; i++)
                {temp = XVECEXP(exp, 0, i);
                 eval_rtx(temp);
                }
            break;
        default: break;
    }
}