二十分钟GCC佛脚指南

给准备加入GNU Toolchain小队的你

—

林思南
GNU Toolchain 小队

TOC

Overview - components

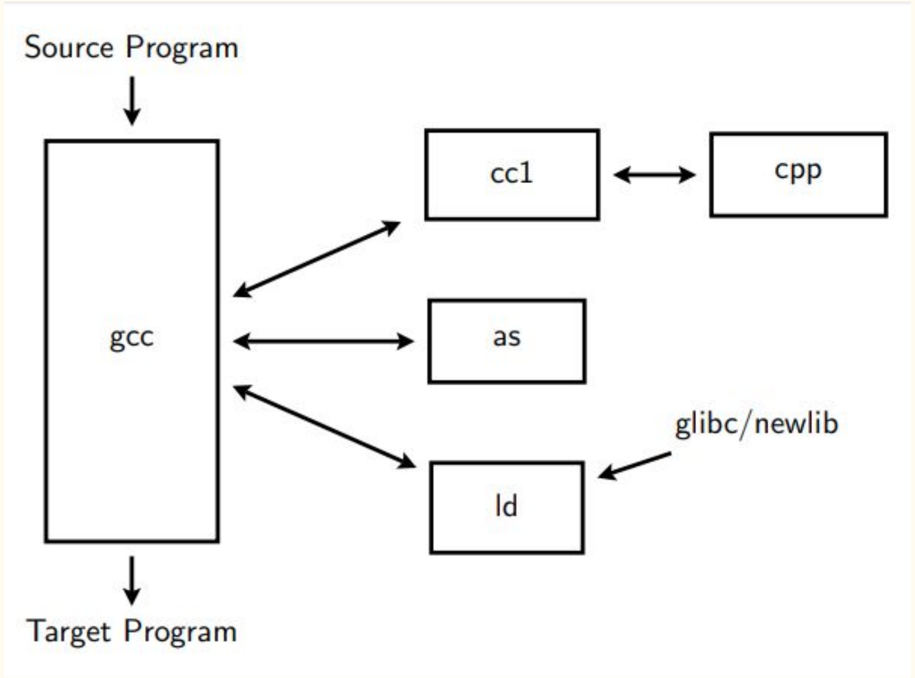


Figure 1. GNU toolchain. Ref [2]

Overview - components

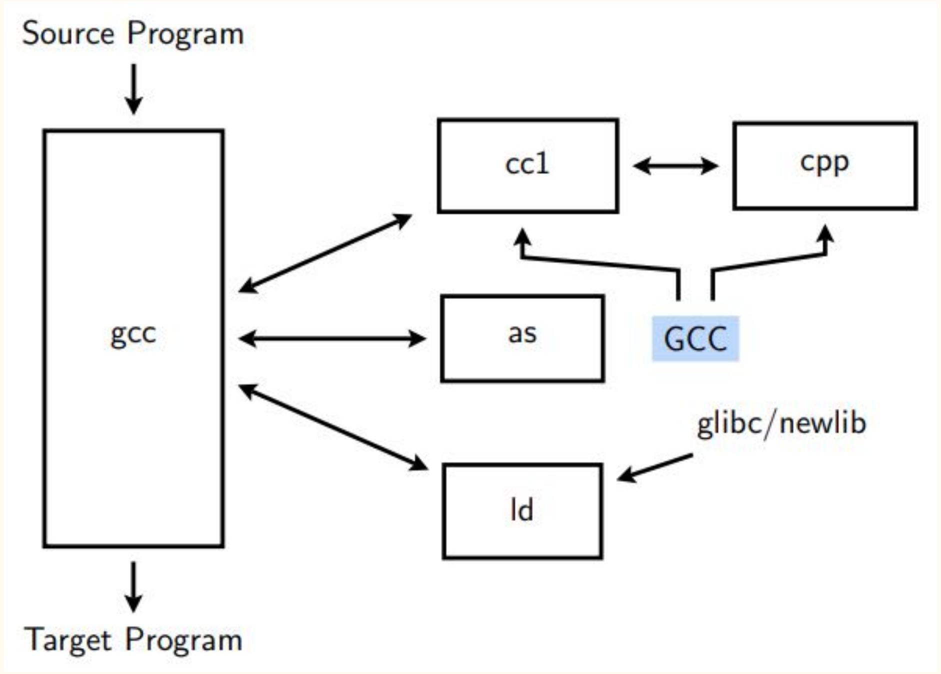


Figure 1. GNU toolchain. Ref [2]

Overview - workflow

前、中和后端三段式的 编译系统

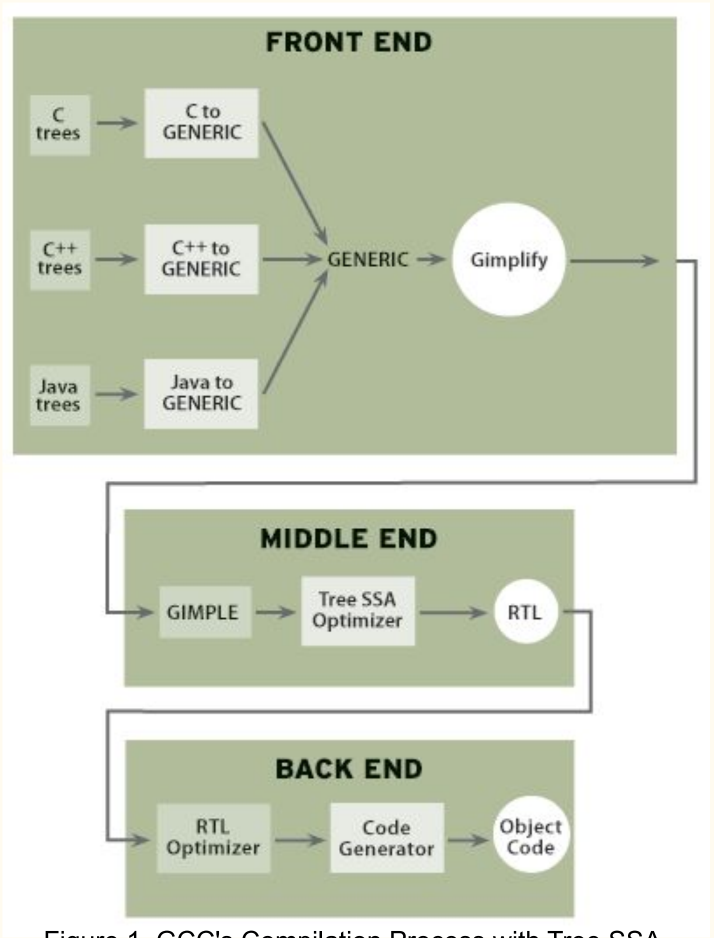


Figure 1. GCC's Compilation Process with Tree SSA.
Ref from [1]

Overview

前、中和后端三段式的编译系统

Immediate Representation

- Abstract Syntax Tree (AST)
- GIMPLE
- Register Transfer Language (RTL)

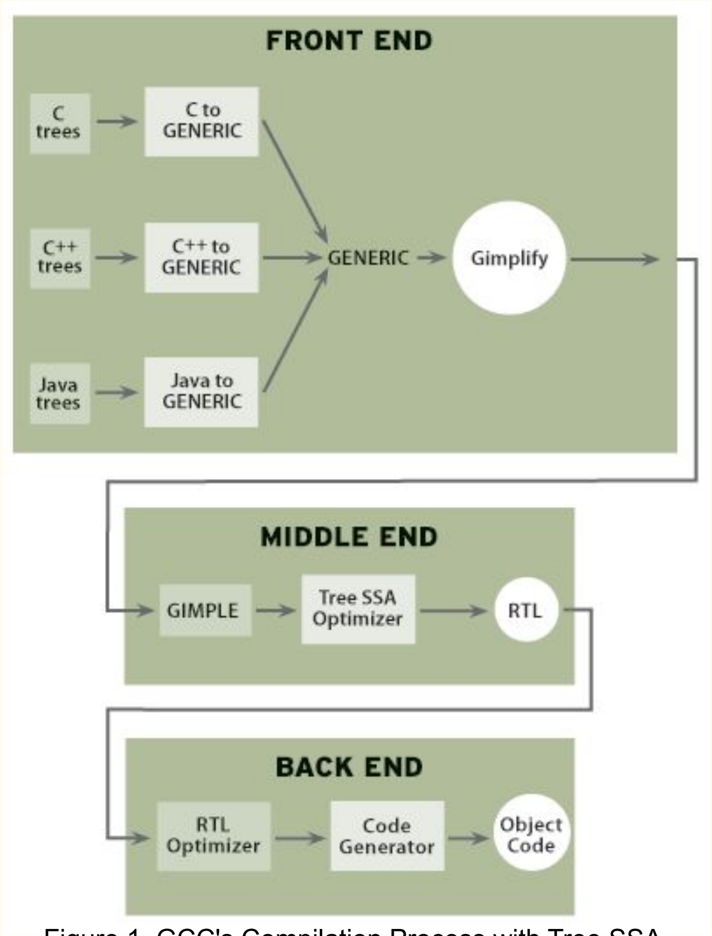


Figure 1. GCC's Compilation Process with Tree SSA.
Ref from [1]

Overview

前、中和后端三段式的 编译系统

Immediate Representation

- Abstract Syntax Tree (AST)
- GIMPLE
- Register Transfer Language (RTL)

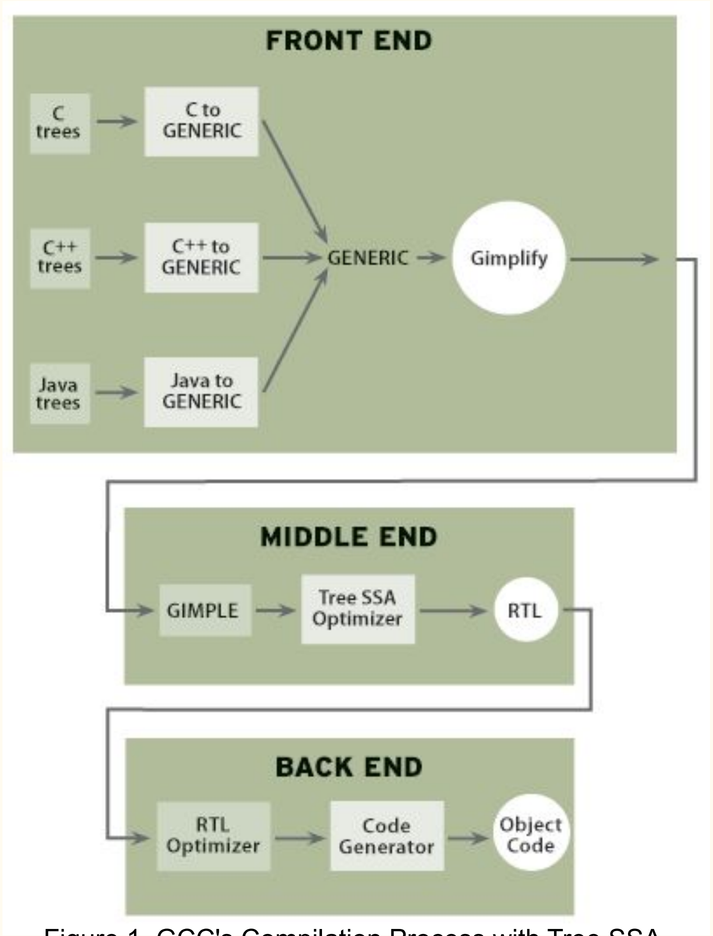


Figure 1. GCC's Compilation Process with Tree SSA.
Ref from [1]

Overview

前、中和后端三段式的编译系统

Immediate Representation

- Abstract Syntax Tree (AST)/Generic
- GIMPLE
- Register Transfer Language (RTL)

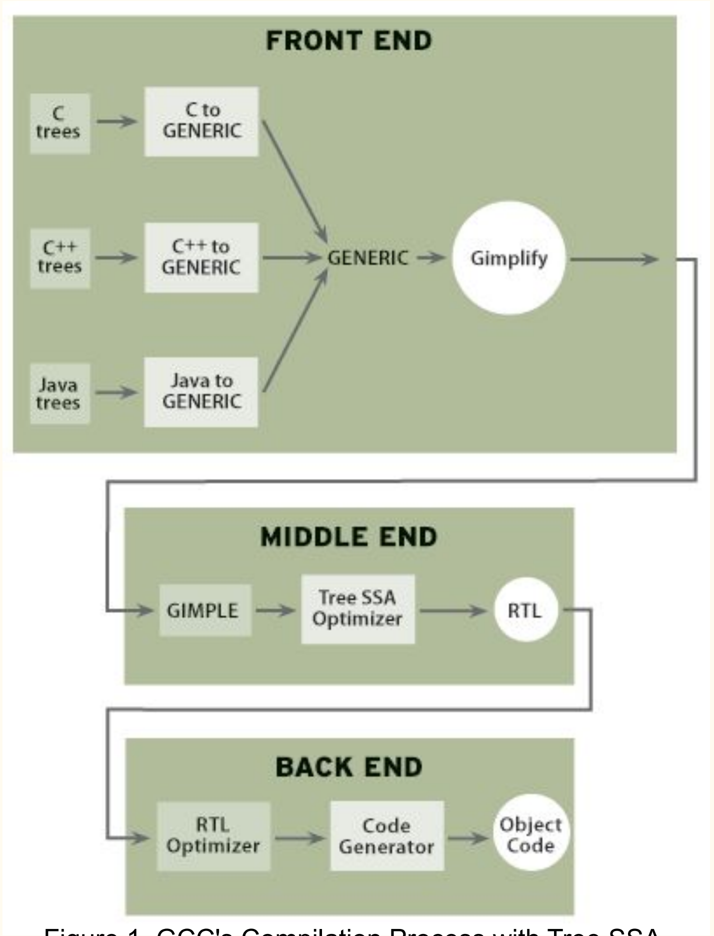


Figure 1. GCC's Compilation Process with Tree SSA.
Ref from [1]

Overview

IR 转换

$$\begin{tikzcd}\text{HLL} \arrow[r, "\text{gimplify}"] & \text{GIMPLE} \arrow[r, "\text{expand}"] & \text{RTL}\end{tikzcd}$$

language-independent &
machine-independent

machine-dependent

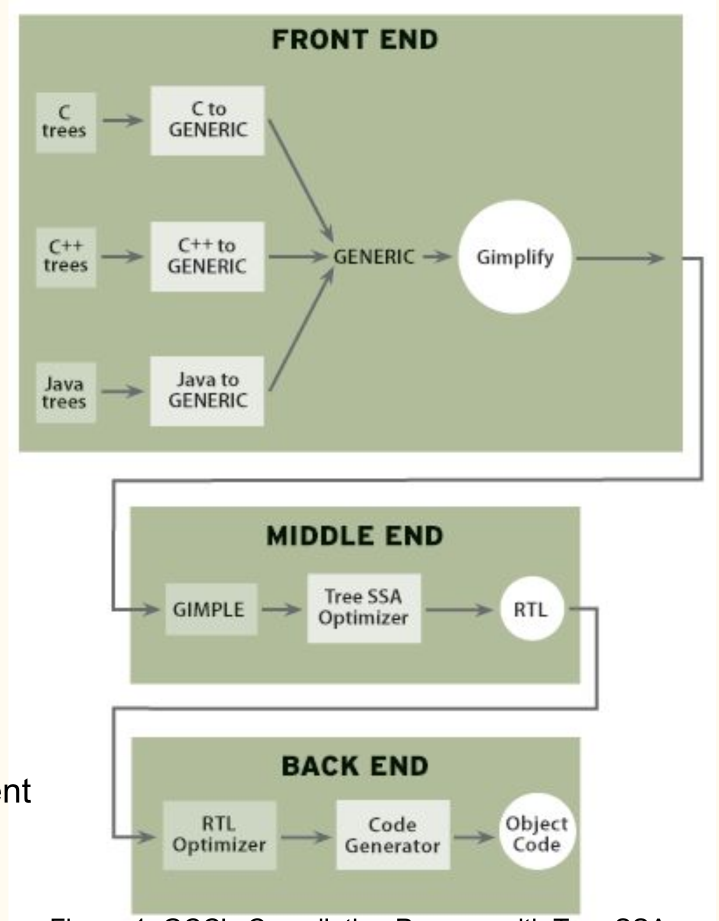


Figure 1. GCC's Compilation Process with Tree SSA.
Ref from [1]

Overview - Front End

目的: 词法分析和语法分析

1. 扫描和分析源代码
2. 生成AST
3. 转换成统一的IR形式, 如GENERIC 或 High Gimple

IR:
Source code -> AST -> Generic -> Gimple

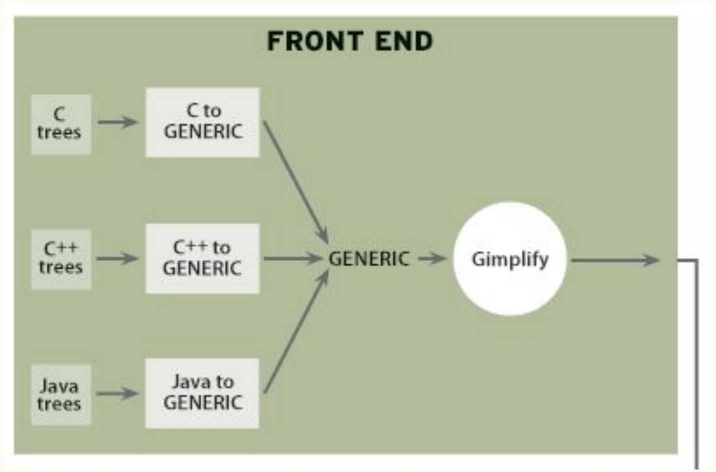
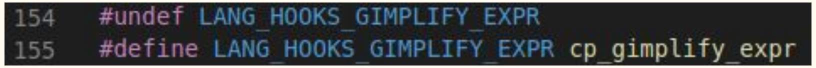


Figure 2. GCC's Frontend. Ref from [1]

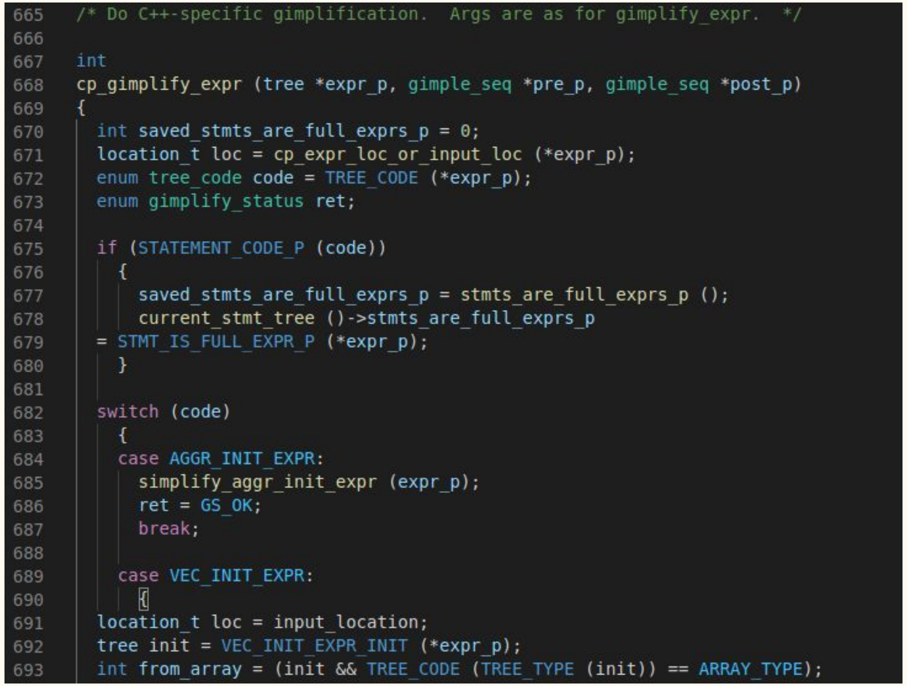
Overview - Front End

通过添加hook，LANG_HOOKS_GIMPLIFY_EXPR，对语言特有表示提供支持，来实现Gimple的转换。如, C++中的向量初始化。



```
#undef LANG_HOOKS_GIMPLIFY_EXPR
#define LANG_HOOKS_GIMPLIFY_EXPR cp_gimplify_expr
```

riscv-gcc/gcc/cp/cp-objcp-common.h



```
/* Do C++-specific gimplification.  Args are as for gimplify_expr.  */

int
cp_gimplify_expr (tree *expr_p, gimple_seq *pre_p, gimple_seq *post_p)
{
  int saved_stmts_are_full_exprs_p = 0;
  location_t loc = cp_expr_loc_or_input_loc (*expr_p);
  enum tree_code code = TREE_CODE (*expr_p);
  enum gimplify_status ret;

  if (STATEMENT_CODE_P (code))
    {
      saved_stmts_are_full_exprs_p = stmts_are_full_exprs_p ();
      current_stmt_tree ()->stmts_are_full_exprs_p
  = STMT_IS_FULL_EXPR_P (*expr_p);
    }

  switch (code)
    {
    case AGGR_INIT_EXPR:
      simplify_aggr_init_expr (expr_p);
      ret = GS_OK;
      break;

    case VEC_INIT_EXPR:
      {
  location_t loc = input_location;
  tree init = VEC_INIT_EXPR_INIT (*expr_p);
  int from_array = (init && TREE_CODE (TREE_TYPE (init)) == ARRAY_TYPE);
```

riscv-gcc/gcc/cp/cp-gimplify.c

Overview - Middle End

目的: 进行语言无关和目标机器无关的优化

1. 对Gimple进行基于静态单赋值形式优化，如死
   码删除，RVO等。
2. 转换为RTL语言，进行更进一步优化。

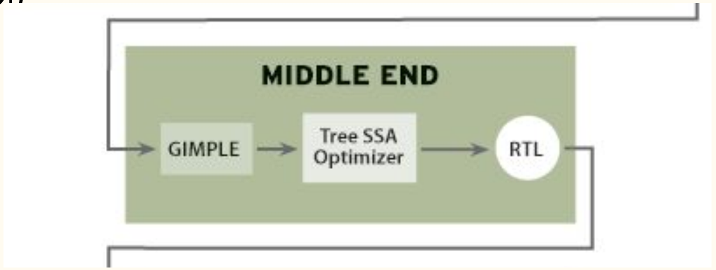


Figure 3. GCC's Middle End. Ref from [1]

IR:
Gimple -> RTL

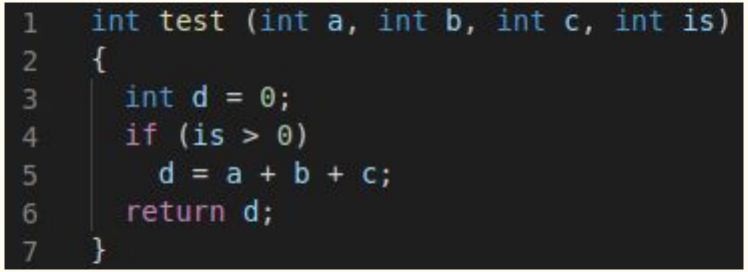
GIMPLE: 以McCAT编译器的SIMPLE IL为原型的IR，三地址码

Overview - Middle End

Middle End中的多种Gimple

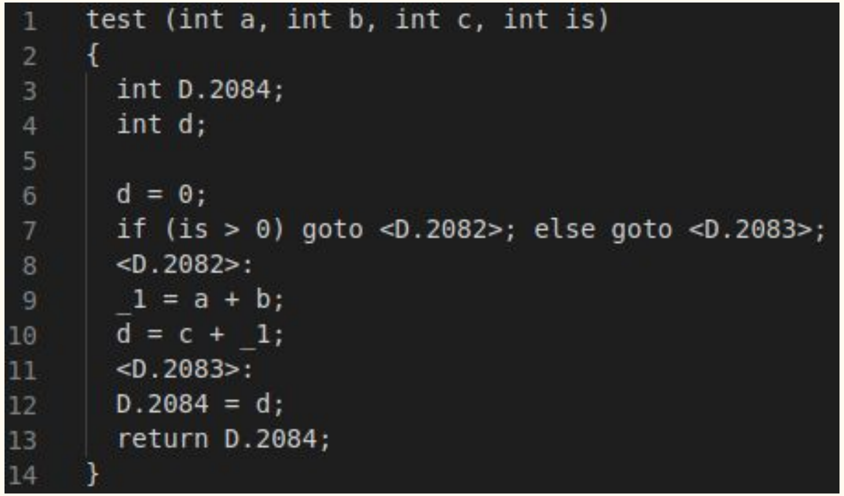
- High Gimple: Front End的输出语言

复杂表达式被切分成三地址 码，显示表示中间变量 (D.2081)



```c
int test (int a, int b, int c, int is)
{
  int d = 0;
  if (is > 0)
    d = a + b + c;
  return d;
}
```

Source code



```
test (int a, int b, int c, int is)
{
  int D.2084;
  int d;

  d = 0;
  if (is > 0) goto <D.2082>; else goto <D.2083>;
  <D.2082>:
  _1 = a + b;
  d = c + _1;
  <D.2083>:
  D.2084 = d;
  return D.2084;
}
```

Gimple

gcc -S sample.c -fdump-tree-gimple

Overview - Middle End

Middle End中的多种Gimple

- High Gimple: 复杂表达式被切分成三地址码，显示表示中间变量
- Low Gimple: 控制流结构线性化得,包括嵌套函数、异常处理和循环
- SSA Gimple: 以静态单赋值形式重写的Gimple

(High GIMPLE -> Low Gimple) -> (SSA tree form / CFG -> optimization) ->

(Basic blocks expand to RTL)

See riscv-gcc/gcc/cfgexpand.c

Overview - Back End

目的: 进行语言无关和目标机器无关的优化

1. 执行RTL优化pass。
2. 生成对应的汇编指令。

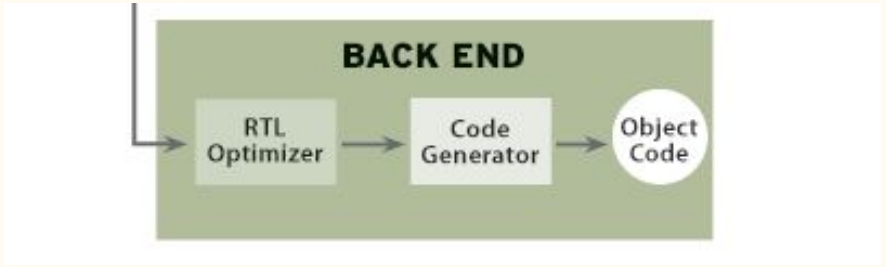


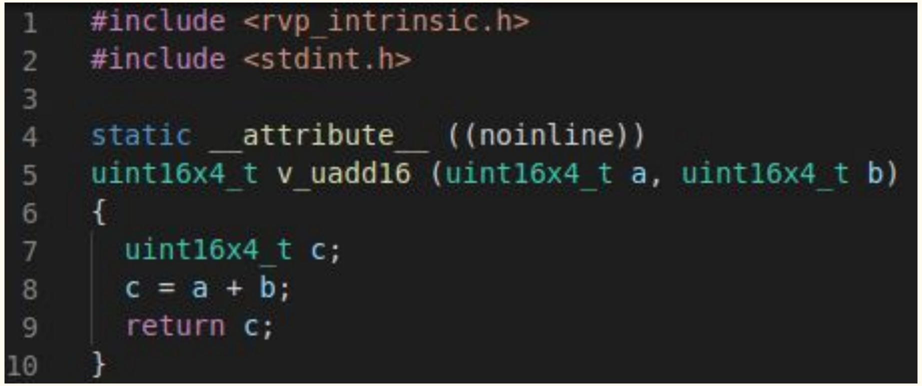
Figure 4. GCC's Backend End. Ref from [1]

IR:
RTL -> ASM

RTL: 接近汇编语言的IR, 底层特性支持(寄存器 类型, Word Size)

GCC workflow

source code



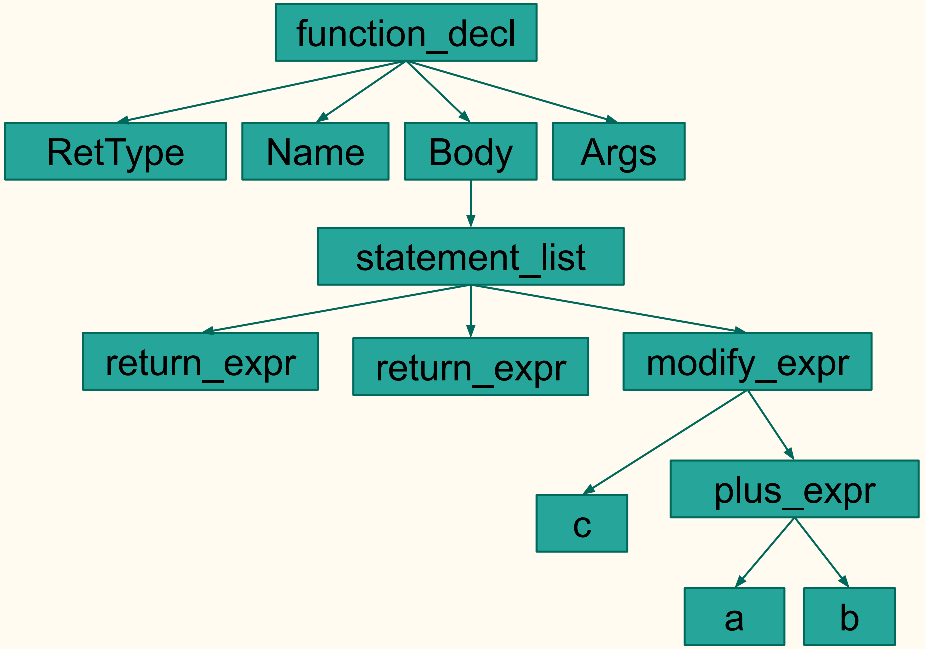
```
#include <rvp_intrinsic.h>
#include <stdint.h>

static __attribute__ ((noinline))
uint16x4_t v_uadd16 (uint16x4_t a, uint16x4_t b)
{
    uint16x4_t c;
    c = a + b;
    return c;
}
```

sample.c

GCC workflow

source code

➡ AST



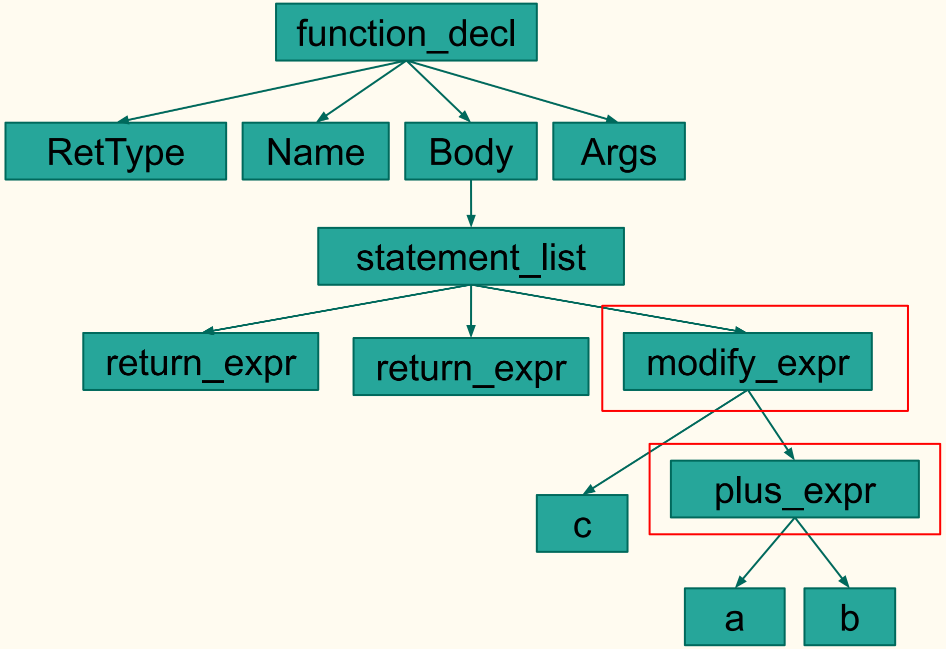
simple.c.004t.original

```
gcc -fdump-tree-original-raw -g -S sample.c
```

GCC workflow

source code

AST



simple.c.004t.original

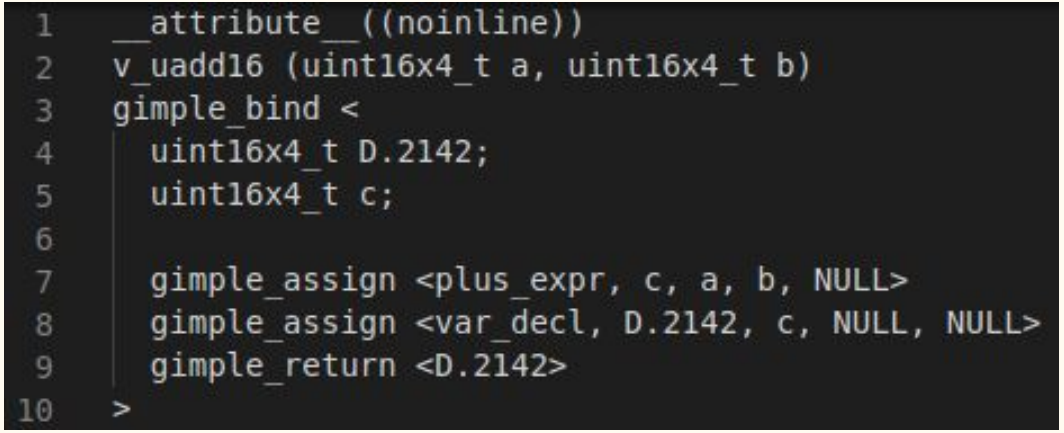
```
gcc -fdump-tree-original-raw -g -S sample.c
```

GCC workflow

source code

AST

Gimple


```
__attribute__((noinline))
v_uadd16 (uint16x4_t a, uint16x4_t b)
gimple_bind <
  uint16x4_t D.2142;
  uint16x4_t c;

  gimple_assign <plus_expr, c, a, b, NULL>
  gimple_assign <var_decl, D.2142, c, NULL, NULL>
  gimple_return <D.2142>
>
```

simple.c.005t.gimple

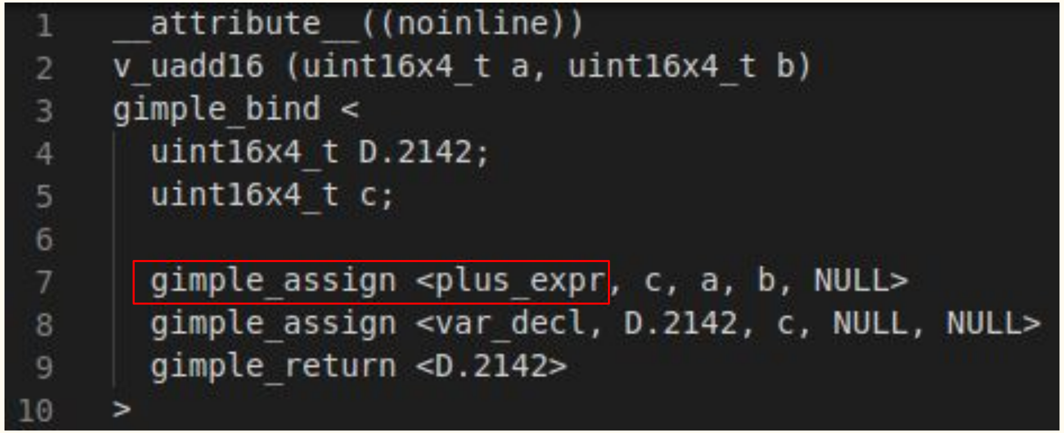
gcc -fdump-tree-original-raw -g -S sample.c

GCC workflow

source code

AST

Gimple



```
__attribute__((noinline))
v_uadd16 (uint16x4_t a, uint16x4_t b)
gimple_bind <
  uint16x4_t D.2142;
  uint16x4_t c;

  gimple_assign <plus_expr, c, a, b, NULL>
  gimple_assign <var_decl, D.2142, c, NULL, NULL>
  gimple_return <D.2142>
>
```

simple.c.005t.gimple

gcc -fdump-tree-gimple-raw -g -S sample.c

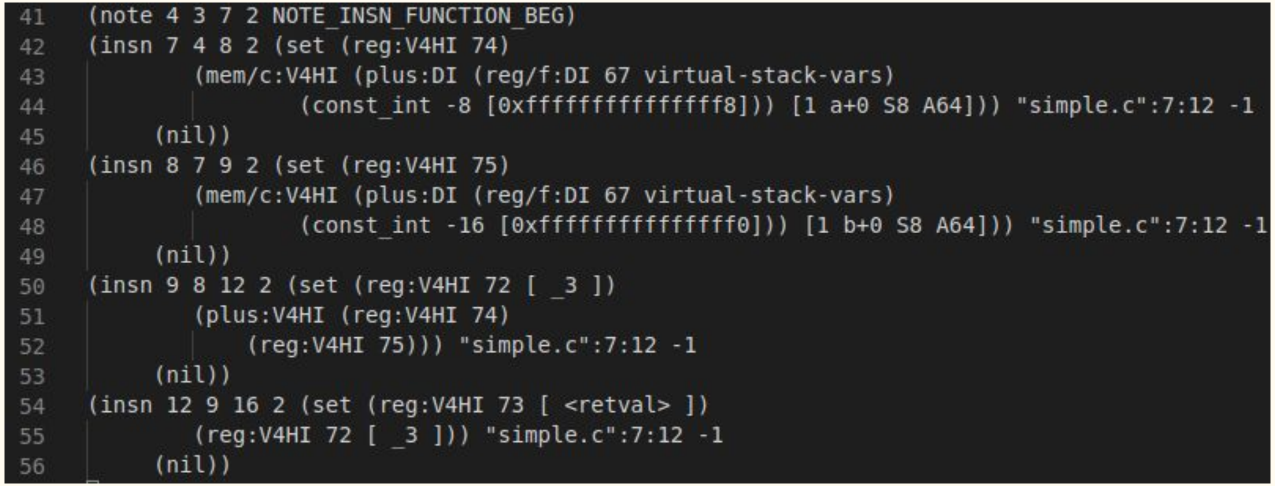
GCC workflow

source code

AST

Gimple

RTL



```
(note 4 3 7 2 NOTE_INSN_FUNCTION_BEG)
(insn 7 4 8 2 (set (reg:V4HI 74)
        (mem/c:V4HI (plus:DI (reg/f:DI 67 virtual-stack-vars)
                (const_int -8 [0xfffffffffffffff8])) [1 a+0 S8 A64])) "simple.c":7:12 -1
     (nil))
(insn 8 7 9 2 (set (reg:V4HI 75)
        (mem/c:V4HI (plus:DI (reg/f:DI 67 virtual-stack-vars)
                (const_int -16 [0xfffffffffffffff0])) [1 b+0 S8 A64])) "simple.c":7:12 -1
     (nil))
(insn 9 8 12 2 (set (reg:V4HI 72 [ _3 ])
        (plus:V4HI (reg:V4HI 74)
            (reg:V4HI 75))) "simple.c":7:12 -1
     (nil))
(insn 12 9 16 2 (set (reg:V4HI 73 [ <retval> ])
        (reg:V4HI 72 [ _3 ])) "simple.c":7:12 -1
     (nil))
```

simple.c.237r.expand

gcc -fdump-rtl-expand -g -S sample.c

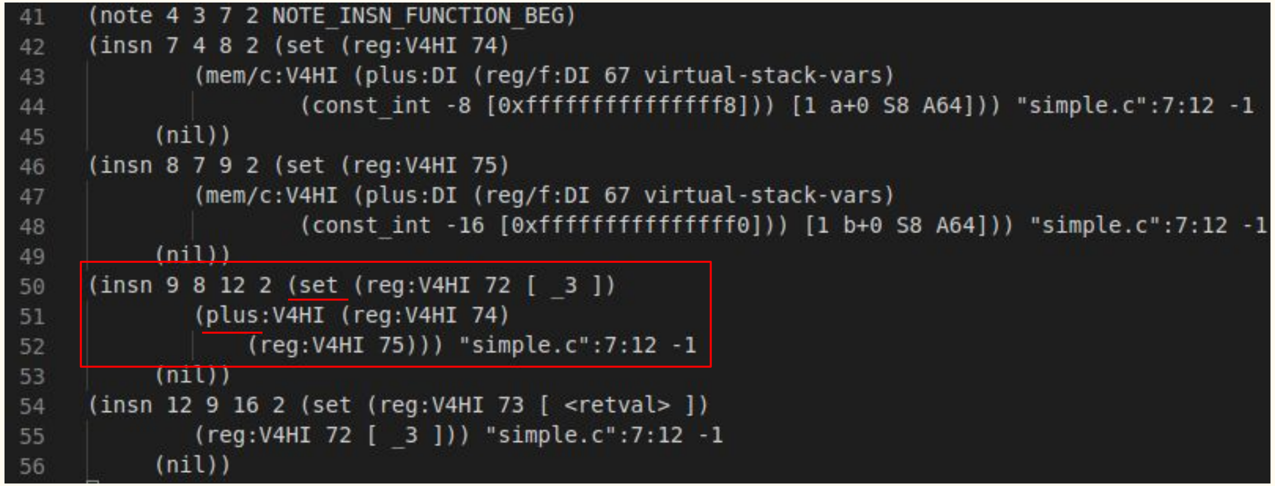
GCC workflow

source code

AST

Gimple

➡ RTL



```
(note 4 3 7 2 NOTE_INSN_FUNCTION_BEG)
(insn 7 4 8 2 (set (reg:V4HI 74)
        (mem/c:V4HI (plus:DI (reg/f:DI 67 virtual-stack-vars)
                (const_int -8 [0xfffffffffffffff8])) [1 a+0 S8 A64])) "simple.c":7:12 -1
     (nil))
(insn 8 7 9 2 (set (reg:V4HI 75)
        (mem/c:V4HI (plus:DI (reg/f:DI 67 virtual-stack-vars)
                (const_int -16 [0xfffffffffffffff0])) [1 b+0 S8 A64])) "simple.c":7:12 -1
     (nil))
(insn 9 8 12 2 (set (reg:V4HI 72 [ _3 ])
        (plus:V4HI (reg:V4HI 74)
            (reg:V4HI 75))) "simple.c":7:12 -1
     (nil))
(insn 12 9 16 2 (set (reg:V4HI 73 [ <retval> ])
        (reg:V4HI 72 [ _3 ])) "simple.c":7:12 -1
     (nil))
```

simple.c.237r.expand

gcc -fdump-rtl-expand -g -S sample.c

GCC workflow

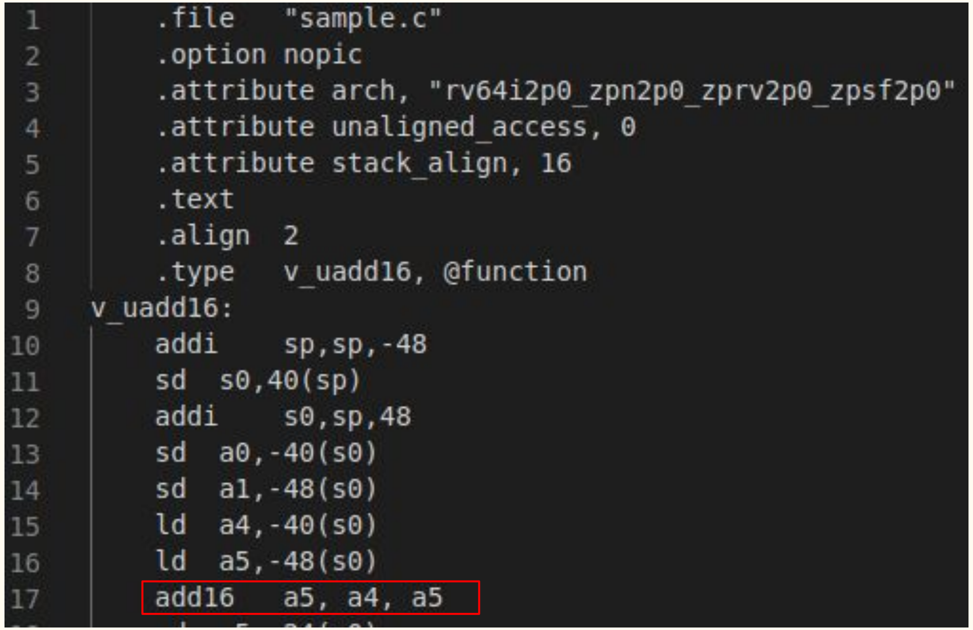
source code

AST

Gimple

RTL

ASM



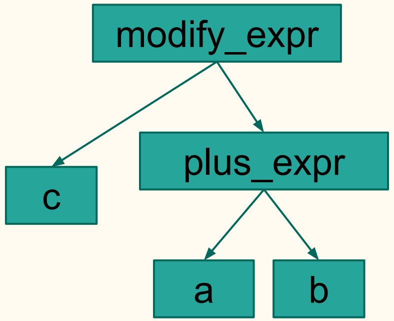
```
	.file	"sample.c"
	.option nopic
	.attribute arch, "rv64i2p0_zpn2p0_zprv2p0_zpsf2p0"
	.attribute unaligned_access, 0
	.attribute stack_align, 16
	.text
	.align	2
	.type	v_uadd16, @function
v_uadd16:
	addi	sp,sp,-48
	sd	s0,40(sp)
	addi	s0,sp,48
	sd	a0,-40(s0)
	sd	a1,-48(s0)
	ld	a4,-40(s0)
	ld	a5,-48(s0)
	add16	a5, a4, a5
```

sample.s

gcc -S sample.c

GCC workflow

c=a+b



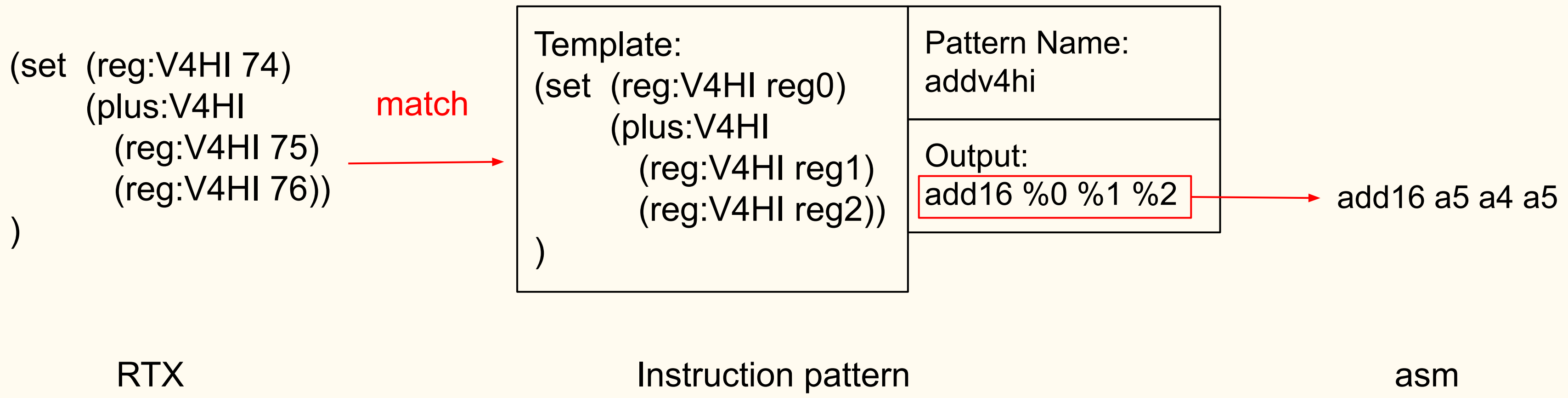
```
(set
    (reg:V4HI 74)
        (plus:V4HI
            (reg:V4HI 75)
            (reg:V4HI 76))
)
```

source code

tree node

register translate
expression (RTX)

GCC workflow



```
(set  (reg:V4HI 74)
      (plus:V4HI
         (reg:V4HI 75)
         (reg:V4HI 76))
)
```

```
Template:
(set  (reg:V4HI reg0)
      (plus:V4HI
         (reg:V4HI reg1)
         (reg:V4HI reg2))
)
```

```
Pattern Name:
addv4hi
```

```
Output:
add16 %0 %1 %2
```

```
add16 a5 a4 a5
```

GCC Backend

- 编译器直接生成指令
- 使用intrinsic

GCC Backend

后端CodeGen开发：如何通过添加和修改dev-time的指令模板描述生成想要的RTL模板。

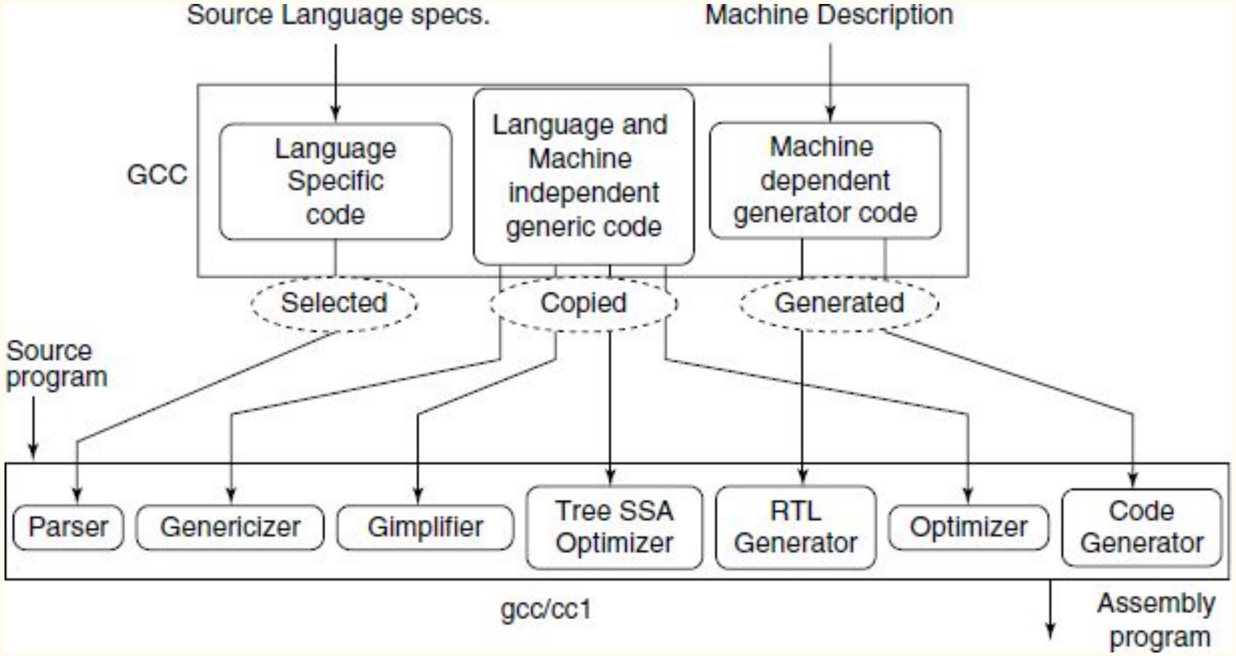


Figure 4. GCC compiler generation framework

GCC Backend

后端CodeGen开发：如何通过添加和修改dev-time的指令模板描述生成想要的指令模板。

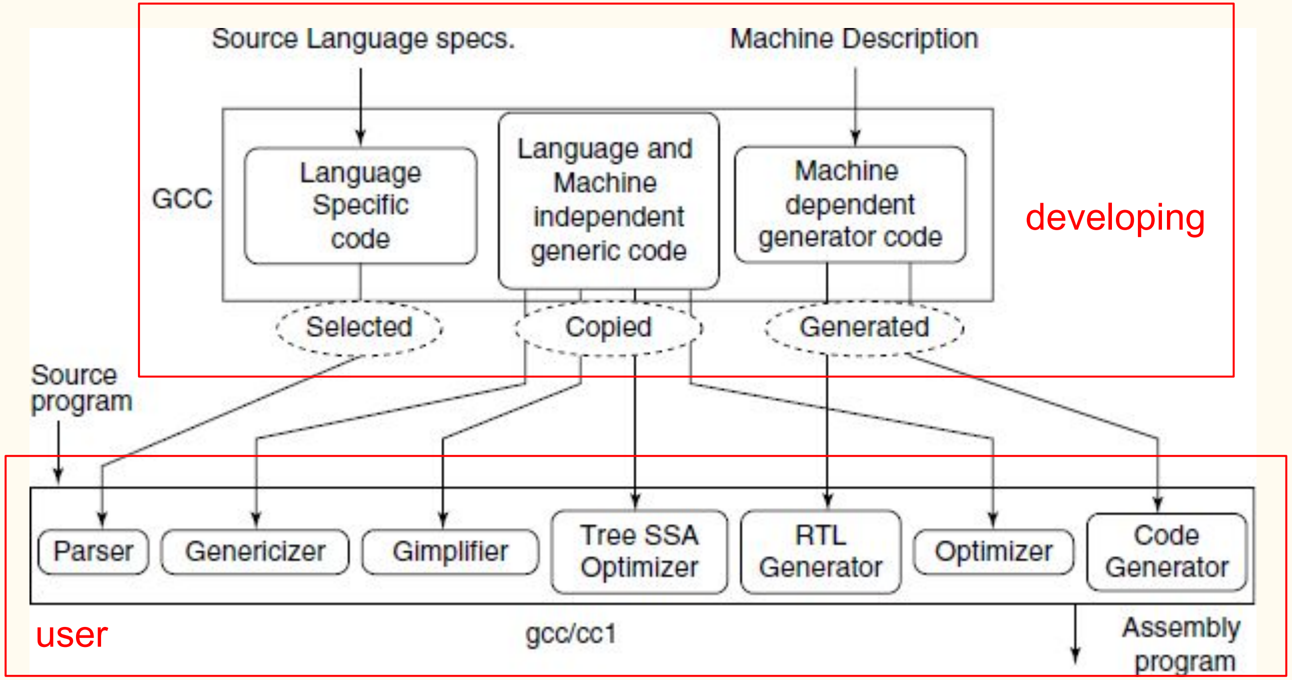
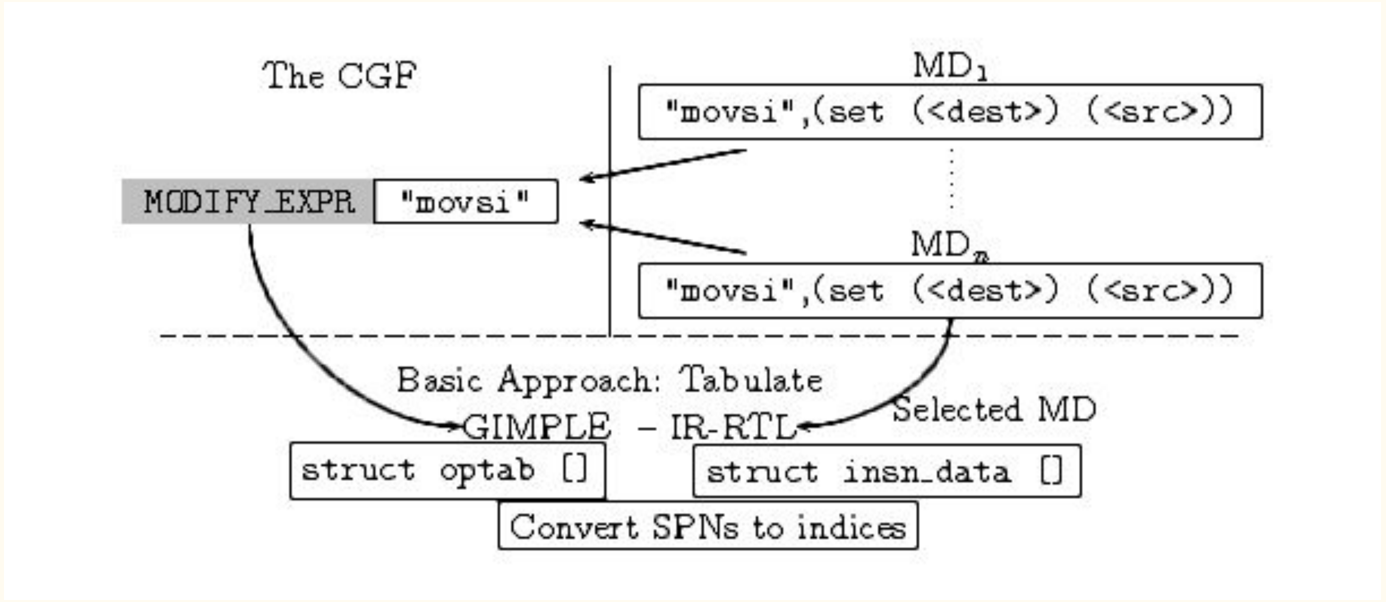


Figure 4. GCC compiler generation framework

GCC Backend



通过GCC standard pattern name的指令自动生成, image from ref [8]

GCC backend

```
(define_insn "addv4hi3"
  [(set (match_operand:V4HI 0 "register_operand"  "=r")
            (plus:V4HI
               (match_operand:V4HI 1 "register_operand" " r")
                 (match_operand:V4HI 2 "register_operand" "
r")))]
  "TARGET_ZPN"
  "add16\t%0, %1, %2"
  [(set_attr "type" "simd")
   (set_attr "mode" "V4HI")])
```

$\longrightarrow$ gen_insn $\longrightarrow$

```
enum insn_code {
  ,...,
  CODE_FOR_addv4hi3 = 317,
  ...
 }
```

riscv-gcc/gcc/config/riscv/rvp.md

riscv-gcc/gcc/gencodes.c

build-gcc-newlib-stage1/gcc/insn-codes.c

GCC backend

```
(define_insn "addv4hi3"
  [(set (match_operand:V4HI 0 "register_operand"  "=r")
           (plus:V4HI
              (match_operand:V4HI 1 "register_operand" " r")
                (match_operand:V4HI 2 "register_operand" "
r")))]
  "TARGET_ZPN"
  "add16\t%0, %1, %2"
  [(set_attr "type" "simd")
   (set_attr "mode" "V4HI")])
```

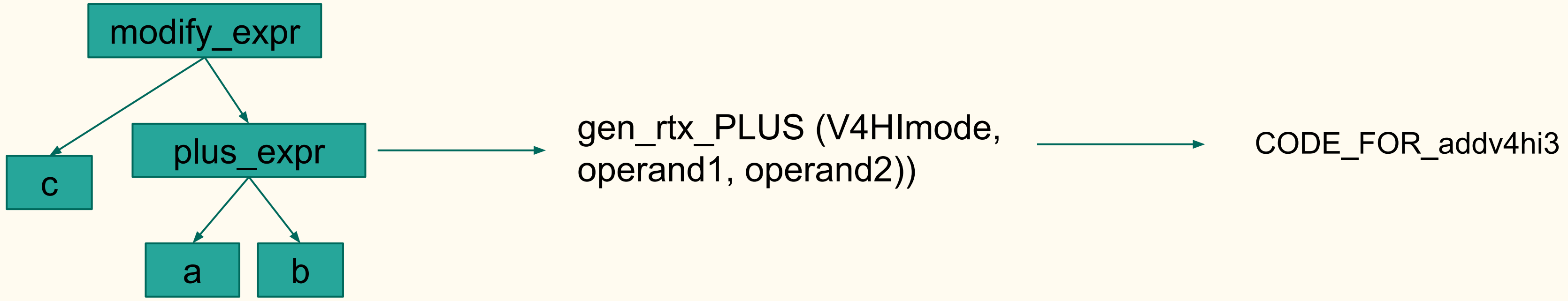
riscv-gcc/gcc/config/riscv/rvp.md

→ gen_insn →

riscv-gcc/gcc/genemit.c

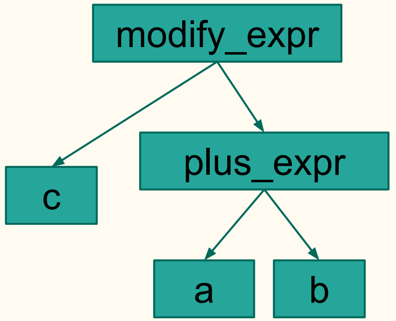
```
/* ./riscv-gcc/gcc/config/riscv/rvp.md:102 */
rtx
gen_addv4hi3 (rtx operand0,
        rtx operand1,
        rtx operand2)
{
  return gen_rtx_SET (operand0,
         gen_rtx_PLUS (V4HImode,
              operand1,
              operand2));
}
```

build-gcc-newlib-stage1/gcc/insn-emit.c


modify_expr
c
plus_expr
a
b
gen_rtx_PLUS (V4HImode, operand1, operand2))
CODE_FOR_addv4hi3

GCC workflow

c=a+b



```
(set
    (reg:V4HI 74)
        (plus:V4HI
            (reg:V4HI 75)
            (reg:V4HI 76))
)
```

source code

tree node

register translate expression (RTX)

GCC backend: generate code from intrinsic

```
riscv_builtins[] = {
{ CODE_FOR_addv4hi3 , "__builtin_riscv_add16 ",                \
    RISCV_BUILTIN_DIRECT, RISCV_UIXLEN_FTYPE_UIXLEN_UIXLEN, \
riscv_builtin_avail_zpn64 }, …,
}
```

riscv-gcc/gcc/config/riscv/riscv-builtins.c

Reference

1. https://web.archive.org/web/20160410185222/https://www.redhat.com/magazine/002dec04/features/gcc/#bibliography
2. https://reup.dmcs.pl/wiki/images/2/2e/Gcc-internals-1.pdf
3. https://www.airs.com/dnovillo/Papers/osdl2006.pdf
4. 《编译系统透视》机械工业出版社
5. https://gcc.gnu.org/onlinedocs/gccint
6. https://docplayer.net/9914280-Generic-and-gimple-a-new-tree-representation-for-entire-functions.html
7. https://wiki.aalto.fi/display/t1065450/Advanced+compilers+2015
8. https://reup.dmcs.pl/wiki/images/0/04/Gcc-internals-2.pdf