Workshop on Essential Abstractions in GCC

An Overview of Compilation and GCC

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

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

July 2010

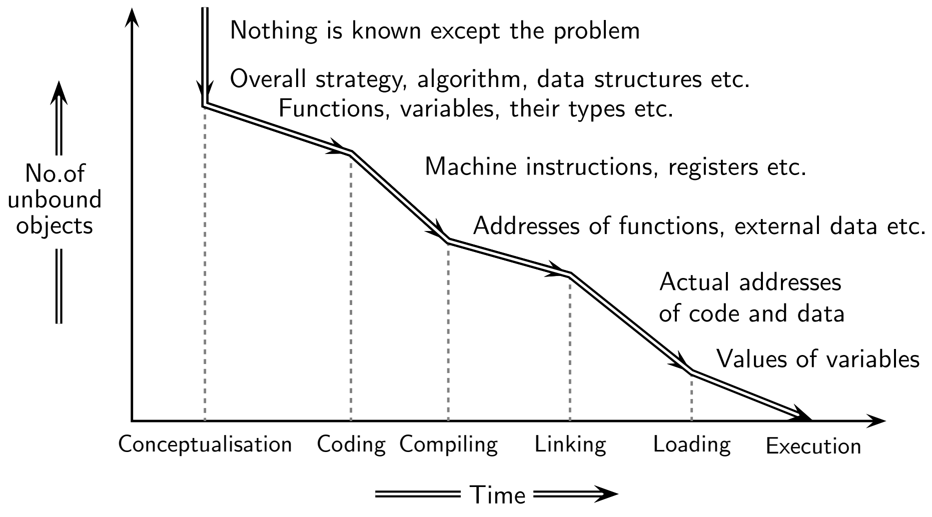
Outline

- Introduction to Compilation
- An Overview of Compilation Phases
- An Overview of GCC

Part 1

Introduction to Compilation

Binding

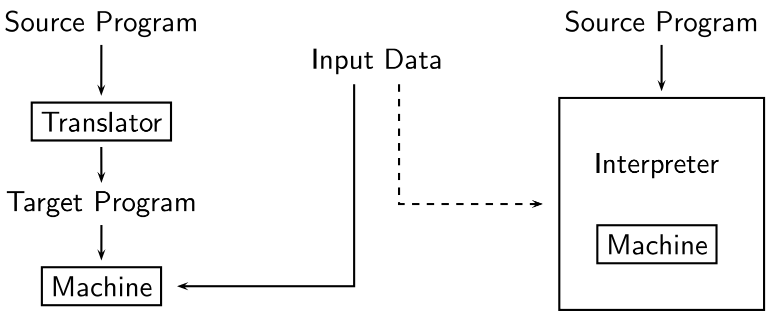


July 2010 Overview: Introduction to Compilation 2/29

Binding

Notes

Implementation Mechanisms



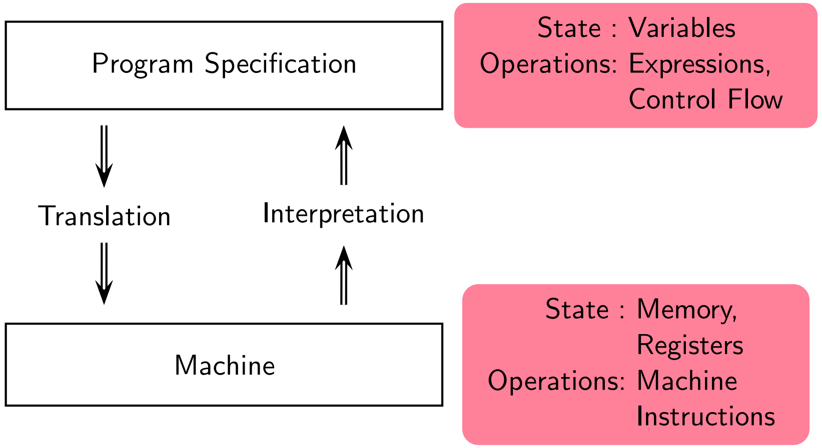
July 2010 Overview: Introduction to Compilation 3/29

Implementation Mechanisms

Notes

Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



Implementation Mechanisms as "Bridges"

Notes

High and Low Level Abstractions

Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
    lw   $t0, 4($fp)  ;    t0 <- b          # Is b smaller
    slti $t0, $t0, 10 ;    t0 <- t0 < 10    # than 10?
    not  $t0, $t0     ;    t0 <- !t0
    bgtz $t0, L0:     ;    if t0>=0 goto L0
    lw   $t0, 4($fp)  ;    t0 <- b          # YES
    b    L1:          ;    goto L1
L0: lw   $t0, 8($fp)  ;L0: t0 <- c          # NO
L1: sw   0($fp), $t0  ;L1: a <- t0
```


High and Low Level Abstractions

Notes

High and Low Level Abstractions

Condition

False Part

True Part

Input C statement

```
a = b<10?b:c;
```

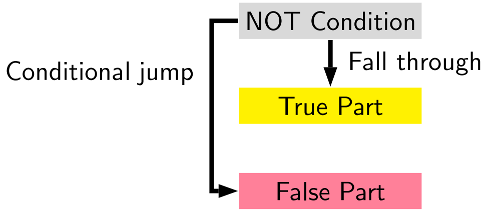
Spim Assembly Equivalent

```
     lw   $t0, 4($fp)  ;    t0 <- b          # Is b smaller
     slti $t0, $t0, 10 ;    t0 <- t0 < 10    # than 10?
     not  $t0, $t0     ;    t0 <- !t0
     bgtz $t0, L0:     ;    if t0>=0 goto L0
     lw   $t0, 4($fp)  ;    t0 <- b          # YES
     b    L1:          ;    goto L1
L0:  lw   $t0, 8($fp)  ;L0: t0 <- c          # NO
L1:  sw   0($fp), $t0  ;L1: a <- t0
```


High and Low Level Abstractions

Notes

High and Low Level Abstractions



Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)  ;    t0 <- b          # Is b smaller
     slti $t0, $t0, 10 ;    t0 <- t0 < 10    # than 10?
     not  $t0, $t0     ;    t0 <- !t0
     bgtz $t0, L0:     ;    if t0>=0 goto L0
     lw   $t0, 4($fp)  ;    t0 <- b          # YES
     b    L1:          ;    goto L1
L0:  lw   $t0, 8($fp)  ;L0: t0 <- c          # NO
L1:  sw   0($fp), $t0  ;L1: a <- t0
```


High and Low Level Abstractions

Notes

Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution

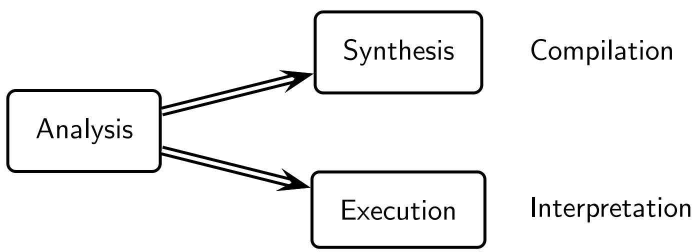
- Translation: Instructions ⟹ Equivalent Instructions

  Interpretation: Instructions ⟹ Actions Implied by Instructions

Implementation Mechanisms

Notes

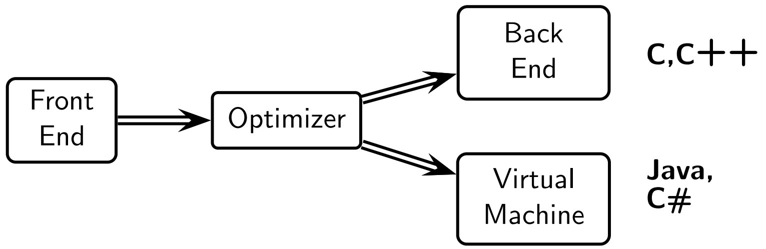
Language Implementation Models



Language Implementation Models

Notes

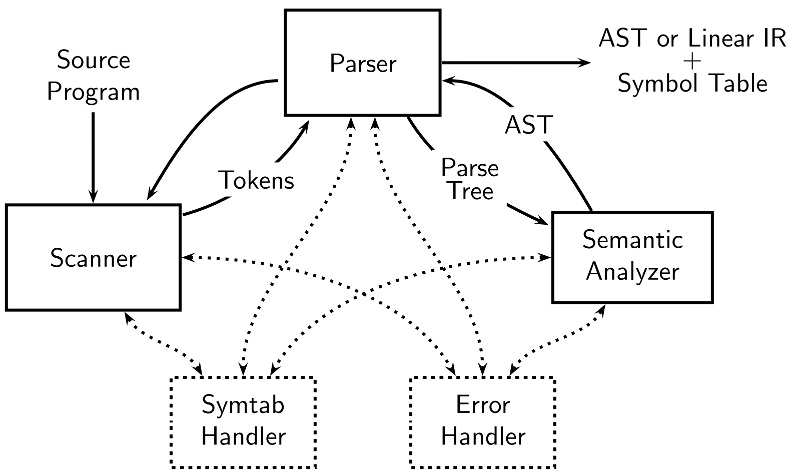
Language Processor Models



Language Processor Models

Notes

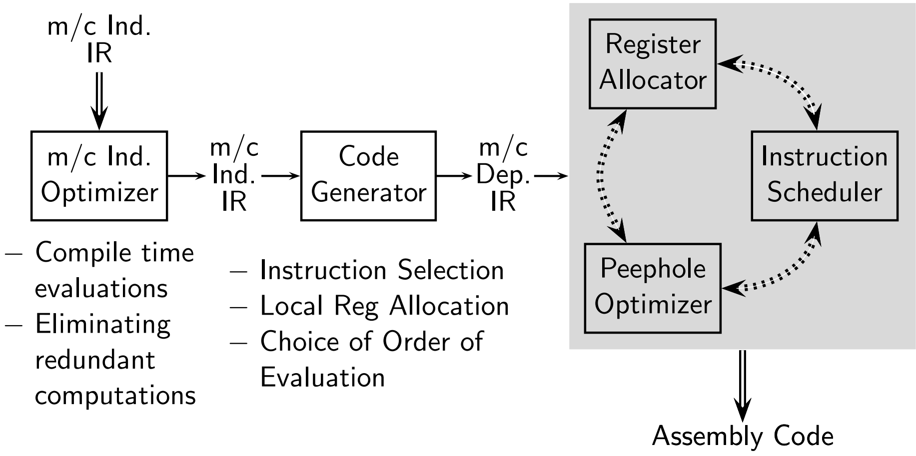
Typical Front Ends



Typical Front Ends

Notes

Typical Back Ends



July 2010 Overview: Introduction to Compilation 10/29

Typical Back Ends

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

July 2010 Overview: Introduction to Compilation 10/29

Typical Back Ends

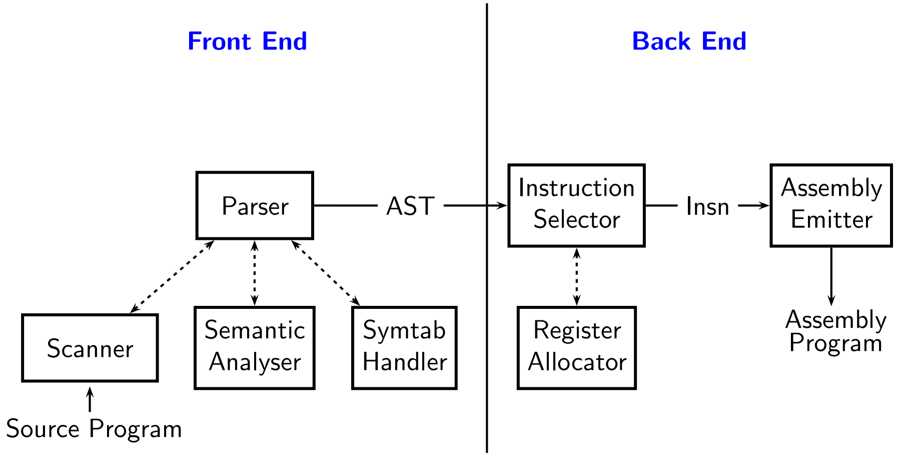
Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 2

An Overview of Compilation Phases

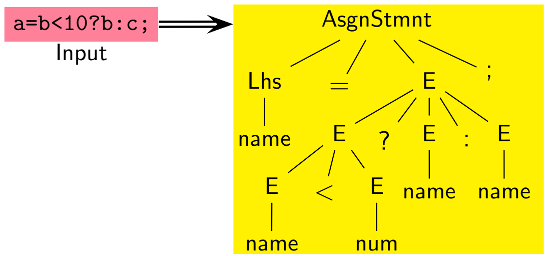
The Structure of a Simple Compiler



The Structure of a Simple Compiler

Notes

Translation Sequence in Our Compiler: Parsing



Parse Tree

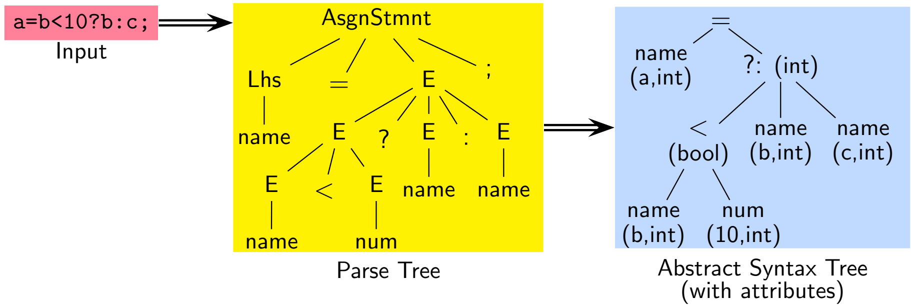
Issues:

- Grammar rules, terminals, non-terminals
- Order of application of grammar rules
  eg. is it `(a = b<10?)` followed by `(b:c)`?
- Values of terminal symbols
  eg. string "10" vs. integer number 10.

Translation Sequence in Our Compiler: Parsing

Notes

Translation Sequence in Our Compiler: Semantic Analysis



Issues:

- Symbol tables

  Have variables been declared? What are their types? What is their scope?
- Type consistency of operators and operands

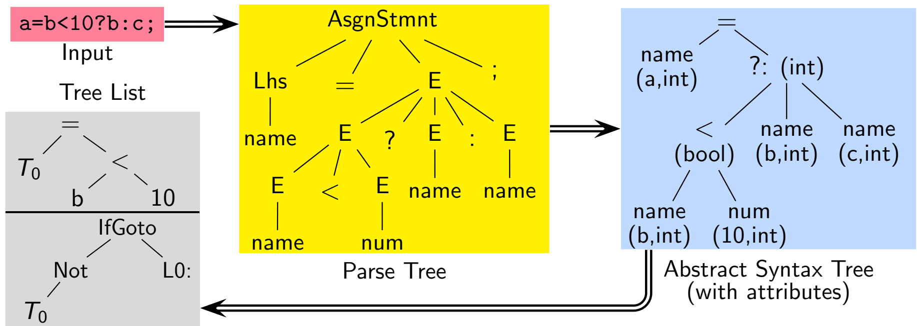
  The result of computing `b<10?` is bool and not int

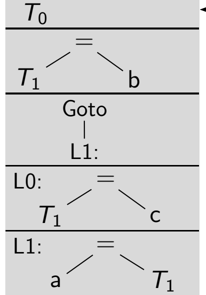
July 2010 Overview: An Overview of Compilation Phases 13/29

Translation Sequence in Our Compiler: Semantic Analysis

Notes

Translation Sequence in Our Compiler: IR Generation





Issues:

- Convert to maximal trees which can be implemented without altering control flow

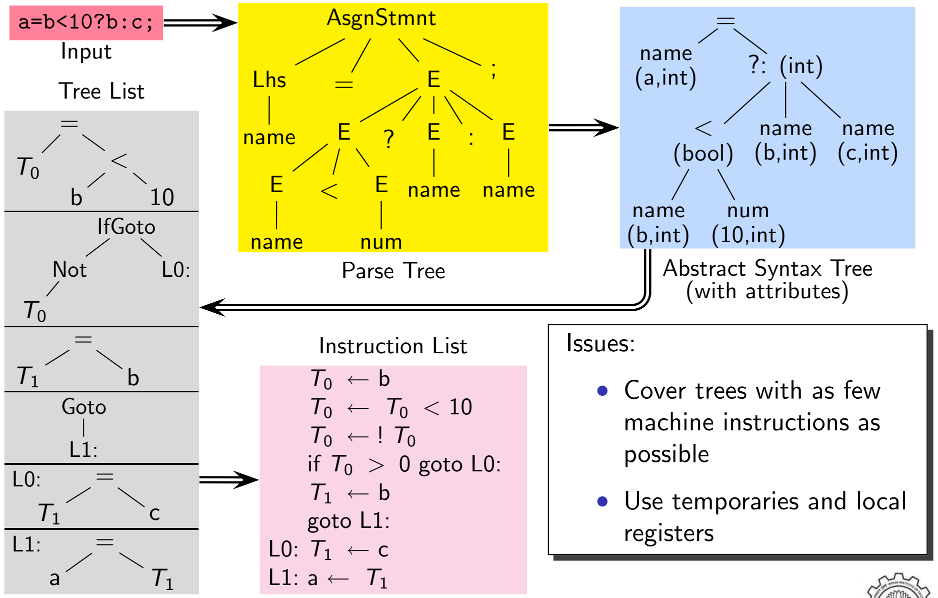
  Simplifies instruction selection and scheduling, register allocation etc.
- Linearise control flow by flattening nested control constructs

July 2010 Overview: An Overview of Compilation Phases 14/29

Translation Sequence in Our Compiler: IR Generation

Notes

Translation Sequence in Our Compiler: Instruction Selection



Issues:

- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

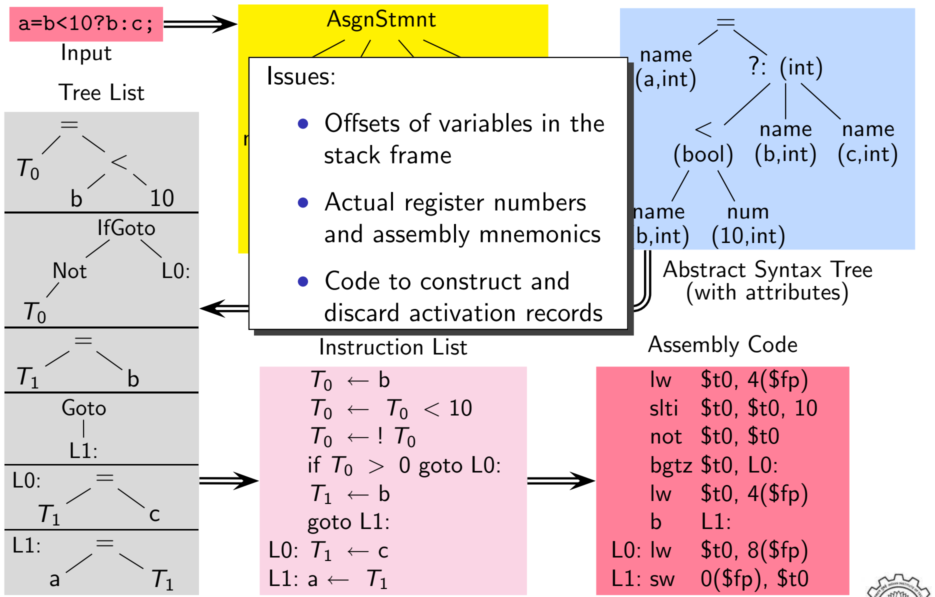
July 2010 Overview: An Overview of Compilation Phases 15/29

Translation Sequence in Our Compiler: Instruction Selection

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Translation Sequence in Our Compiler: Emitting Instructions



Issues:

- Offsets of variables in the stack frame
- Actual register numbers and assembly mnemonics
- Code to construct and discard activation records

July 2010 Overview: An Overview of Compilation Phases 16/29

Translation Sequence in Our Compiler: Emitting Instructions

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 3

GCC ≡ The Great Compiler Challenge

What is GCC?

- For the GCC developer community: The GNU Compiler Collection
- For other compiler writers: The Great Compiler Challenge ☺

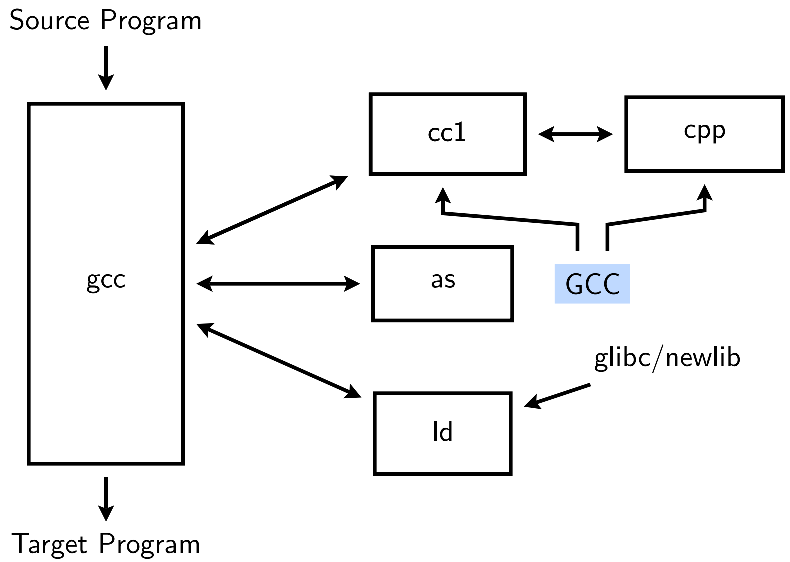
July 2010 Overview: GCC ≡ The Great Compiler Challenge 17/29

What is GCC?

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

The GNU Tool Chain



The GNU Tool Chain

Notes

Why is Understanding GCC Difficult?

Some of the obvious reasons:

- Comprehensiveness

  GCC is a production quality framework in terms of completeness and practical usefulness

- Open development model

  Could lead to heterogeneity. Design flaws may be difficult to correct

- Rapid versioning

  GCC maintenance is a race against time. Disruptive corrections are difficult

Why is Understanding GCC Difficult?

Notes

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control
  Design, implement, test, release
- Bazaar: Total Decentralization
  Release early, release often, make users partners in software development

  "Given enough eyeballs, all bugs are shallow"
  Code errors, logical errors, and architectural errors

A combination of the two seems more sensible

Open Source and Free Software Development Model

Notes

The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

- GCC Steering Committee: Free Software Foundation has given charge
  - Major policy decisions
  - Handling Administrative and Political issues
- Release Managers:
  - Coordination of releases
- Maintainers:
  - Usually area/branch/module specific
  - Responsible for design and implementation
  - Take help of reviewers to evaluate submitted changes

The Current Development Model of GCC

Notes

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC, NEC SX architecture

July 2010 Overview: GCC ≡ The Great Compiler Challenge 22/29

Comprehensiveness of GCC: Wide Applicability

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Comprehensiveness of GCC: Size

Count		gcc-4.3.0	gcc-4.4.2	gcc-4.5.0
Lines	The main source	2029115	2187216	2320963
	Libraries	1546826	1633558	1671501
	Subdirectories	3527	3794	4055
Files	Total number of files	57660	62301	77782
	C source files	15477	18225	20024
	Header files	9646	9213	9389
	C++ files	3708	4232	4801
	Java files	6289	6340	6340
	Makefiles and templates	163	163	169
	Configuration scripts	52	52	56
	Machine description files	186	206	229

(Line counts estimated by David A. Wheeler's `sloccount` program)

July 2010 Overview: GCC ≡ The Great Compiler Challenge 23/29

Comprehensiveness of GCC: Size

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Why is Understanding GCC Difficult?

Deeper reason: GCC is not a *compiler* but a *compiler generation framework*

There are two distinct gaps that need to be bridged:

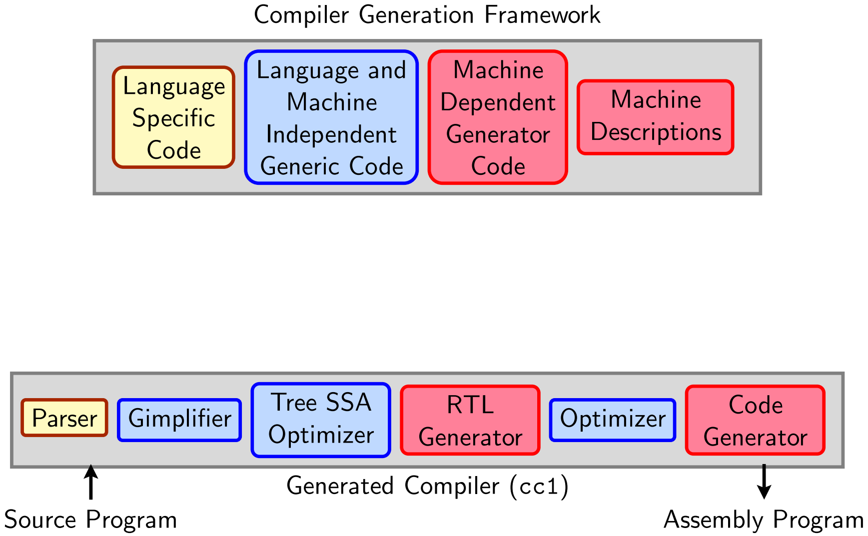
- Input-output of the generation framework: The target specification and the generated compiler
- Input-output of the generated compiler: A source program and the generated assembly program

July 2010 Overview: GCC ≡ The Great Compiler Challenge 24/29
Why is Understanding GCC Difficult?

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

The Architecture of GCC

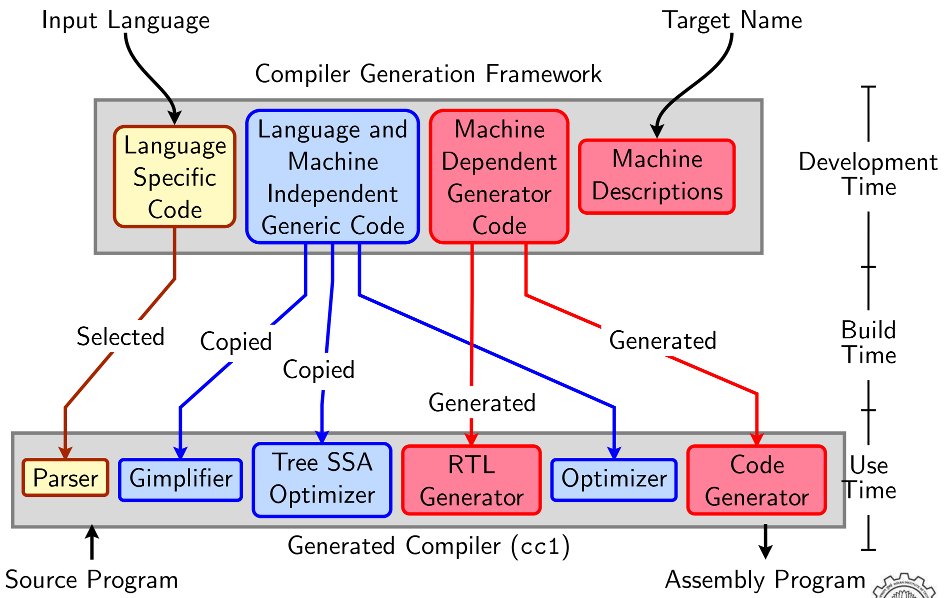


July 2010 Overview: GCC ≡ The Great Compiler Challenge 25/29
The Architecture of GCC

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

The Architecture of GCC



The Architecture of GCC

Notes

An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```

- The required C statements are generated during the build

An Example of The Generation Related Gap

Notes

Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.5.0/gcc using cscope

```
  File             Line
0 collect2.c       1111 main (int argc, char **argv)
1 fp-test.c          85 main (void )
2 gcc.c            6803 main (int argc, char **argv)
3 gcov-dump.c        76 main (int argc ATTRIBUTE_UNUSED, char **argv)
4 gcov-iov.c         29 main (int argc, char **argv)
5 gcov.c            355 main (int argc, char **argv)
6 genattr.c          89 main (int argc, char **argv)
7 genattrtab.c     4439 main (int argc, char **argv)
8 genautomata.c    9475 main (int argc, char **argv)
9 genchecksum.c      67 main (int argc, char ** argv)
a gencodes.c         51 main (int argc, char **argv)
b genconditions.c  209 main (int argc, char **argv)
c genconfig.c       261 main (int argc, char **argv)
d genconstants.c     50 main (int argc, char **argv)
e genemit.c         825 main (int argc, char **argv)
f genextract.c      401 main (int argc, char **argv)
```


Another Example of The Generation Related Gap

Notes

Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.5.0/gcc using cscope

```
  File             Line
g genflags.c        250 main (int argc, char **argv)
h gengenrtl.c       350 main (int argc, char **argv)
i gengtype.c       3694 main (int argc, char **argv)
j genmddeps.c        45 main (int argc, char **argv)
k genmodes.c       1376 main (int argc, char **argv)
l genopinit.c       469 main (int argc, char **argv)
m genoutput.c      1023 main (int argc, char **argv)
n genpeep.c         353 main (int argc, char **argv)
o genpreds.c       1404 main (int argc, char **argv)
p genrecog.c       2722 main (int argc, char **argv)
q lto-wrapper.c     412 main (int argc, char *argv[])
r main.c             33 main (int argc, char **argv)
s mips-tdump.c     1393 main (int argc, char **argv)
t mips-tfile.c      655 main (void )
u mips-tfile.c     4695 main (int argc, char **argv)
v tlink.c            61 const char *main;
```


Another Example of The Generation Related Gap

Notes

The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism

  Large size of specifications

- Size in terms of line counts

Files	i386	mips
*.md	35766	12930
*.c	28643	12572
*.h	15694	5105

July 2010 Overview: GCC ≡ The Great Compiler Challenge 28/29

The GCC Challenge: Poor Retargetability Mechanism

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Meeting the GCC Challenge

Goal of Understanding	Methodology	Needs Examining		
		Makefiles	Source	MD
Translation sequence of programs	Gray box probing	No	No	No
Build process	Customizing the configuration and building	Yes	No	No
Retargetability issues and machine descriptions	Incremental construction of machine descriptions	No	No	Yes
IR data structures and access mechanisms	Adding passes to massage IRs	No	Yes	Yes
Retargetability mechanism		Yes	Yes	Yes

July 2010 Overview: GCC ≡ The Great Compiler Challenge 29/29

Meeting the GCC Challenge

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay