Demystifying GCC Through Gray Box Probing

Uday Khedker

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

September 2010

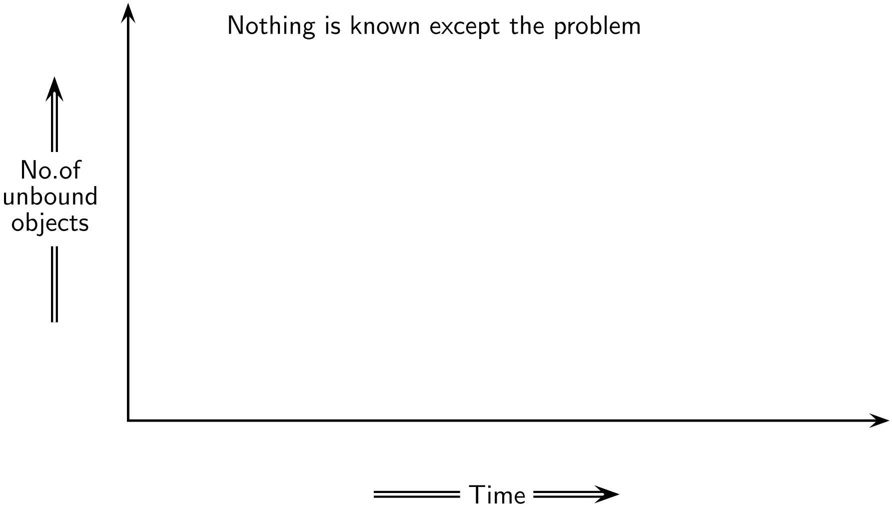
Outline

- Introduction to Compilation
- An Overview of Compilation Process
- An Overview of GCC
- First Level Graybox Probing of GCC
- Graybox Probing of GCC for Machine Independent Optimizations
- Configuration and Building
- Activities of GCC Resource Center

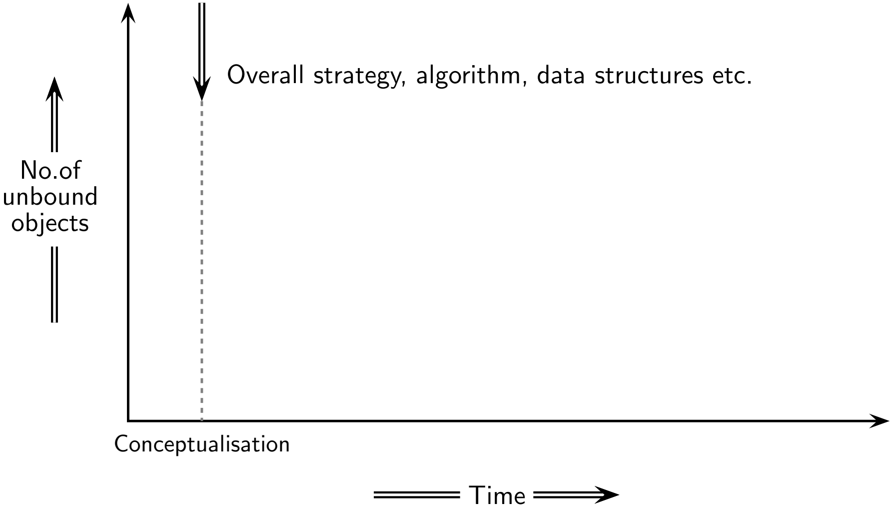
Part 1

Introduction to Compilation

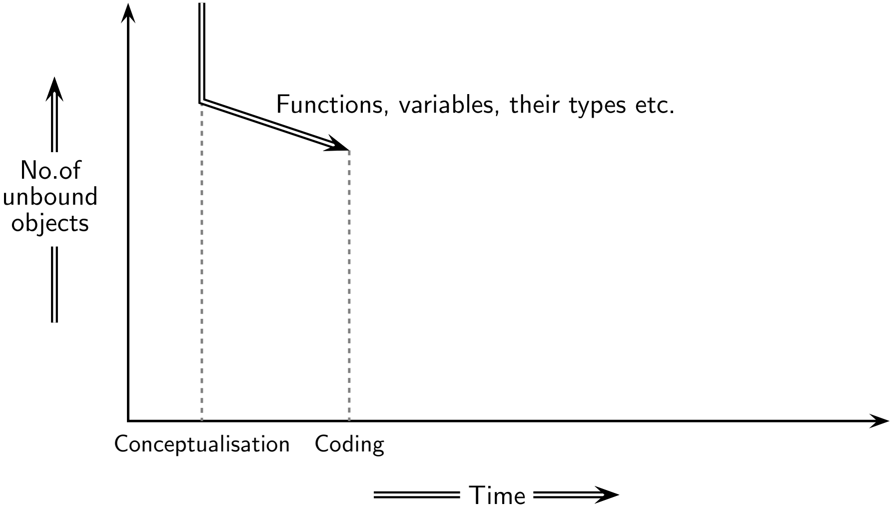
Binding


Nothing is known except the problem
No.of
unbound
objects
Time

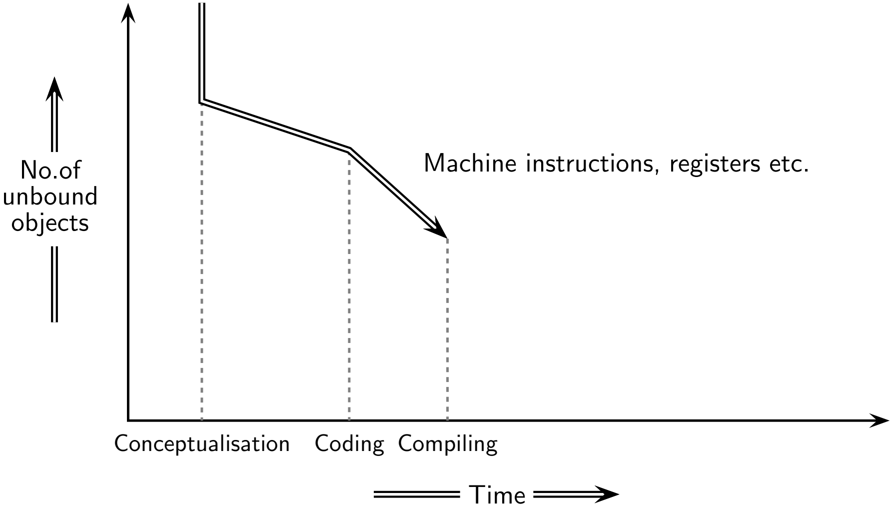
Binding


Overall strategy, algorithm, data structures etc.
No.of unbound objects
Conceptualisation
Time

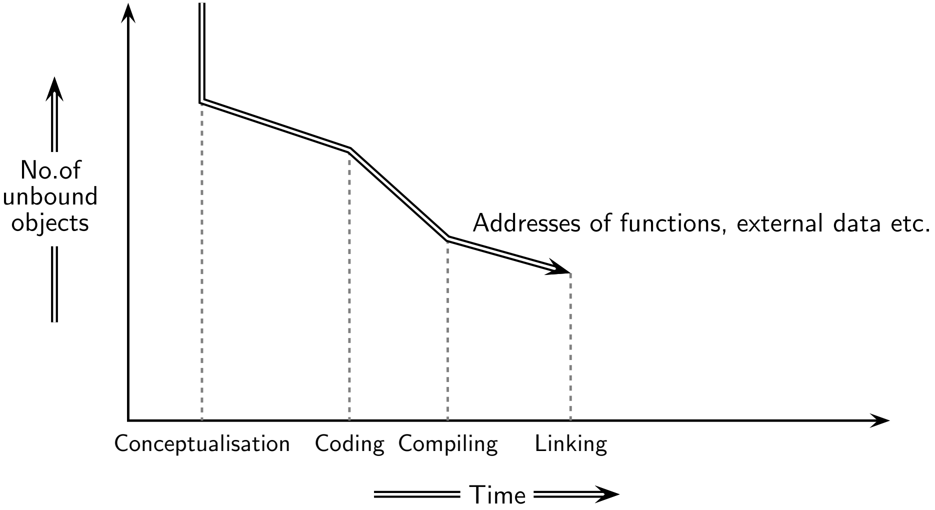
Binding


Functions, variables, their types etc.
No.of unbound objects
Conceptualisation
Coding
Time

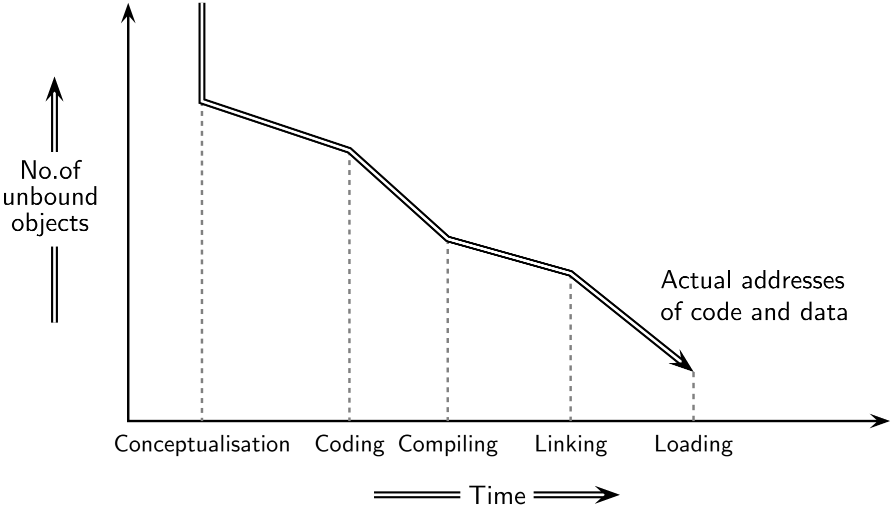
Binding


No.of unbound objects
Machine instructions, registers etc.
Conceptualisation
Coding
Compiling
Time

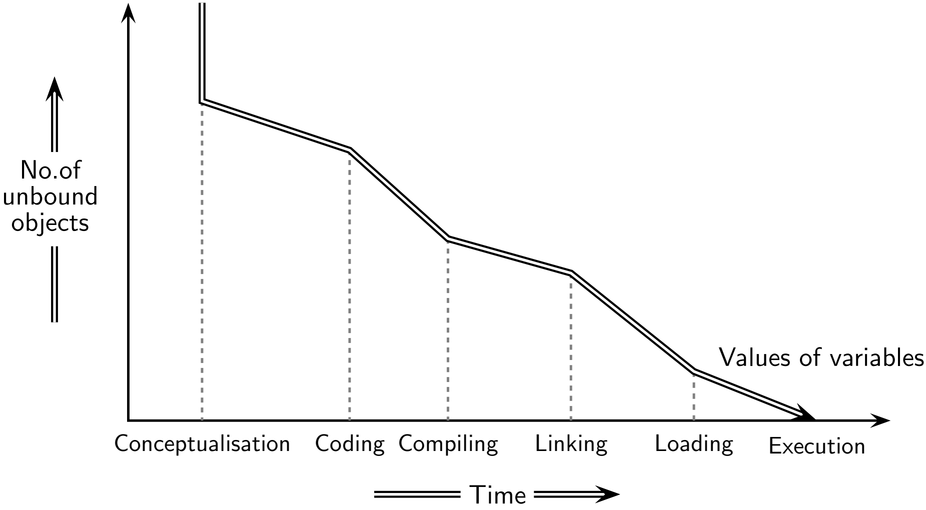
Binding


No.of unbound objects
Addresses of functions, external data etc.
Conceptualisation
Coding
Compiling
Linking
Time

Binding


No.of
unbound
objects
Actual addresses
of code and data
Conceptualisation
Coding
Compiling
Linking
Loading
Time

Binding


No.of unbound objects
Values of variables
Conceptualisation
Coding
Compiling
Linking
Loading
Execution
Time

Implementation Mechanisms

Source Program

↓

Translator

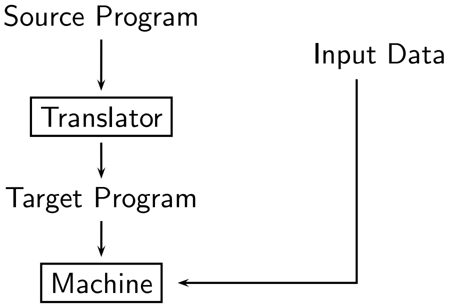
↓

Target Program

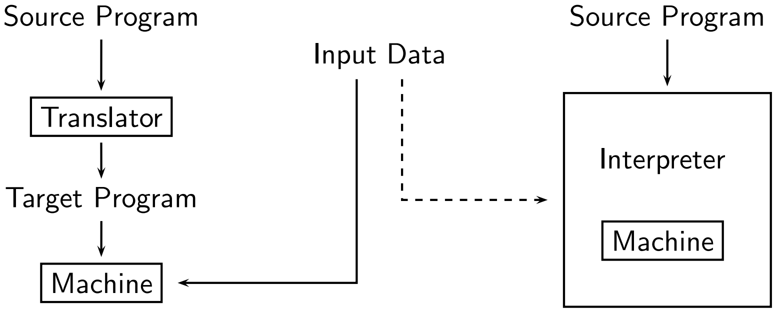
↓

Machine

Implementation Mechanisms



Implementation Mechanisms



Implementation Mechanisms as “Bridges”

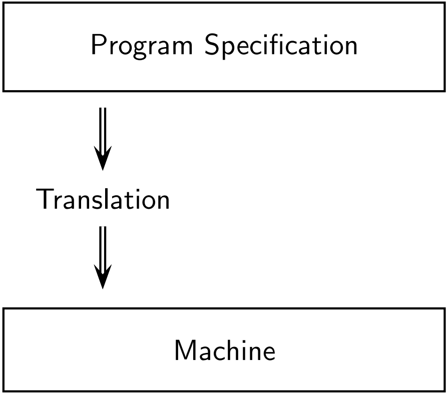
- “Gap” between the “levels” of program specification and execution

Program Specification

Machine

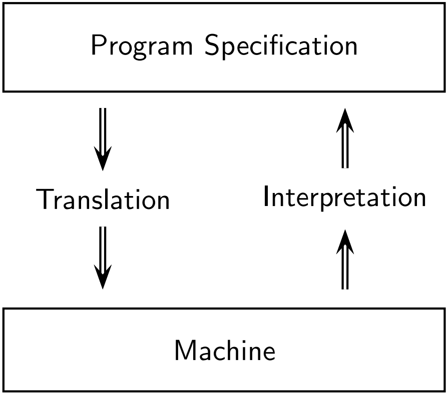
Implementation Mechanisms as "Bridges"

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



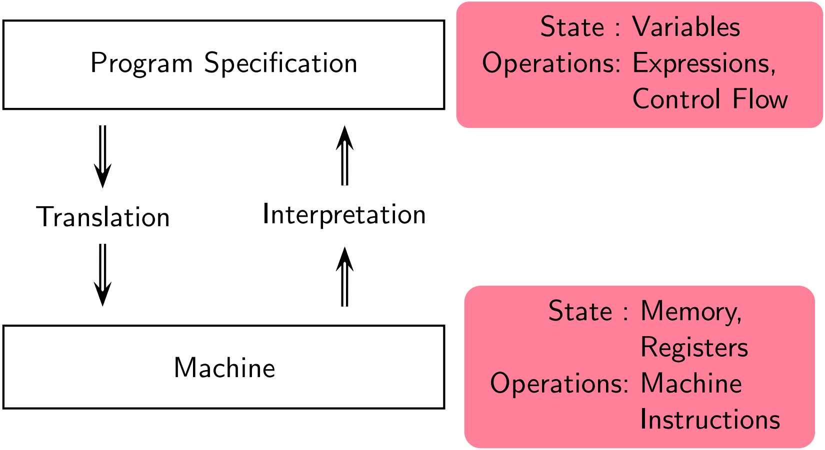
Implementation Mechanisms as "Bridges"

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



Implementation Mechanisms as "Bridges"

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



High and Low Level Abstractions

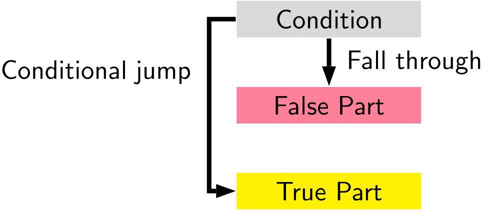
Input C statement

```
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



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

NOT Condition

True Part

False Part

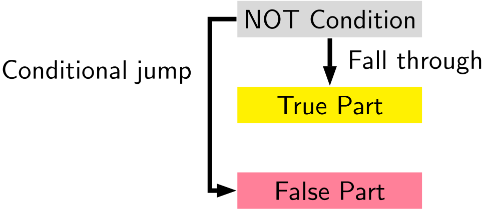
Input C statement

```
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



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
```

Implementation Mechanisms

- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution

- Translation Instructions $\Longrightarrow$ Equivalent Instructions

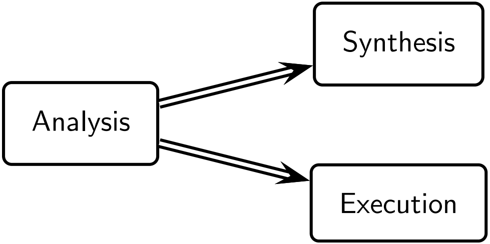
Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution

- Translation Instructions $\Longrightarrow$ Equivalent Instructions

  Interpretation Instructions $\Longrightarrow$ Actions Implied by Instructions

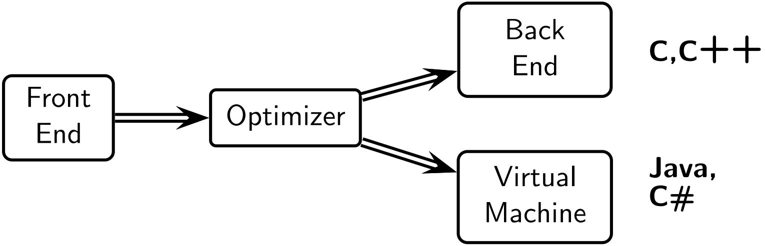
Language Implementation Models



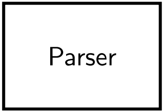
Compilation

Interpretation

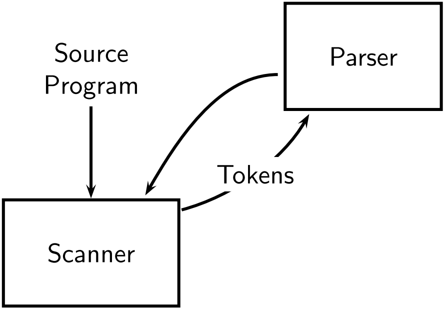
Language Processor Models



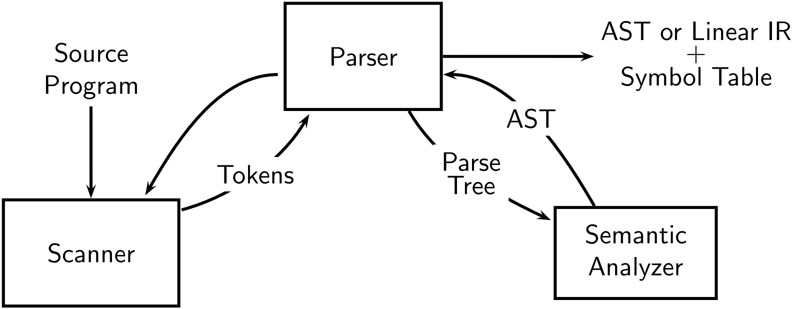
Typical Front Ends



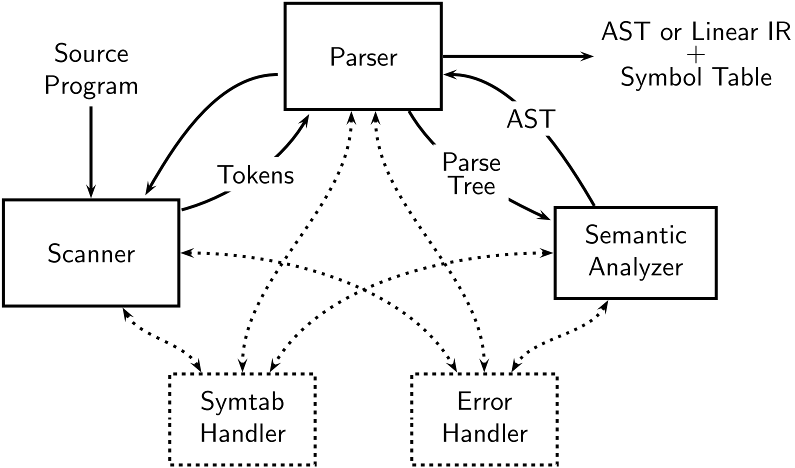
Typical Front Ends



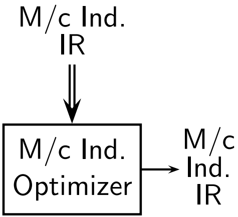
Typical Front Ends



Typical Front Ends

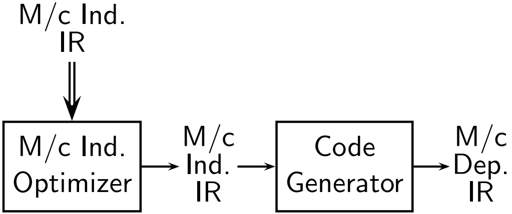


Typical Back Ends



- Compile time evaluations
- Eliminating redundant computations

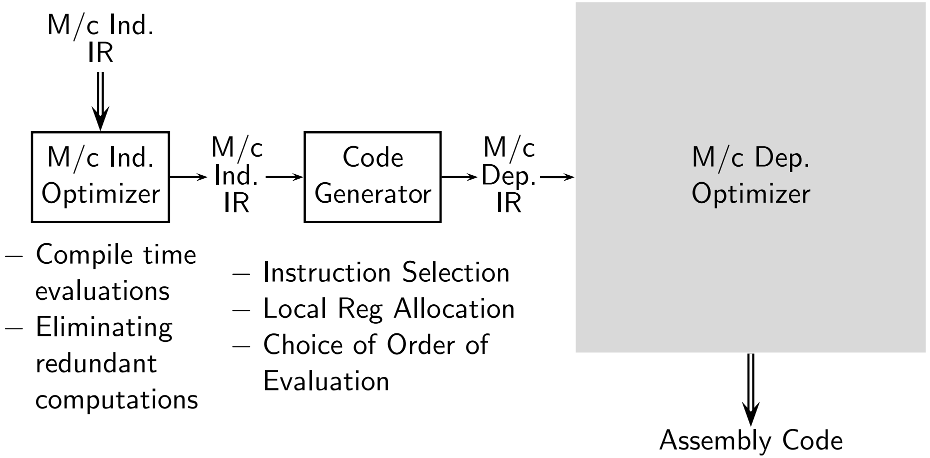
Typical Back Ends



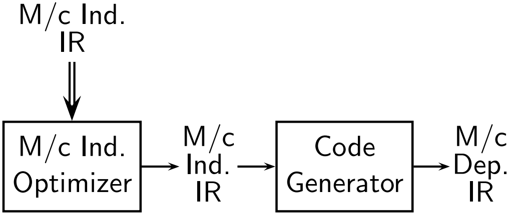
- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Typical Back Ends

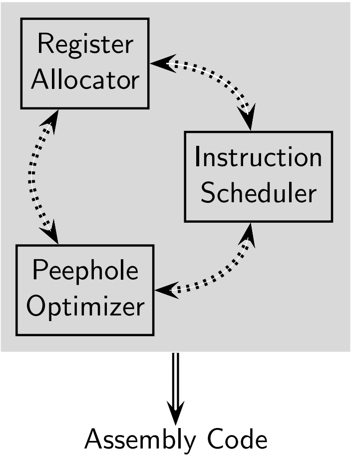


Typical Back Ends



– Compile time evaluations
– Eliminating redundant computations

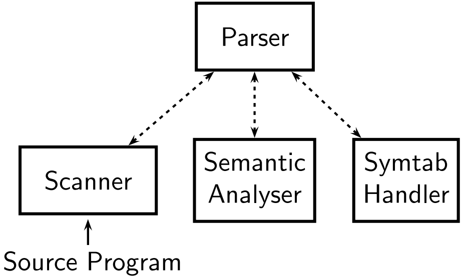
– Instruction Selection
– Local Reg Allocation
– Choice of Order of Evaluation



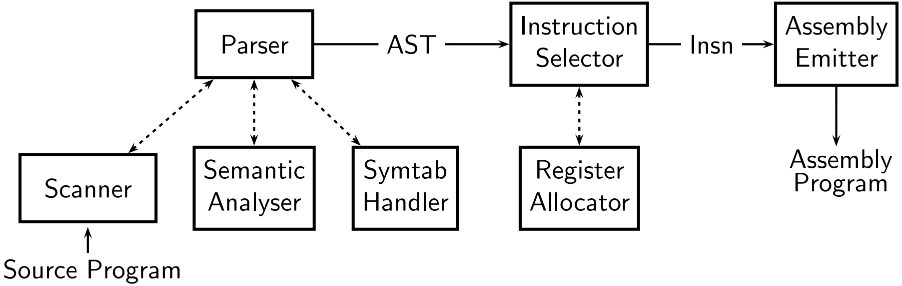
Part 2

An Overview of Compilation Phases

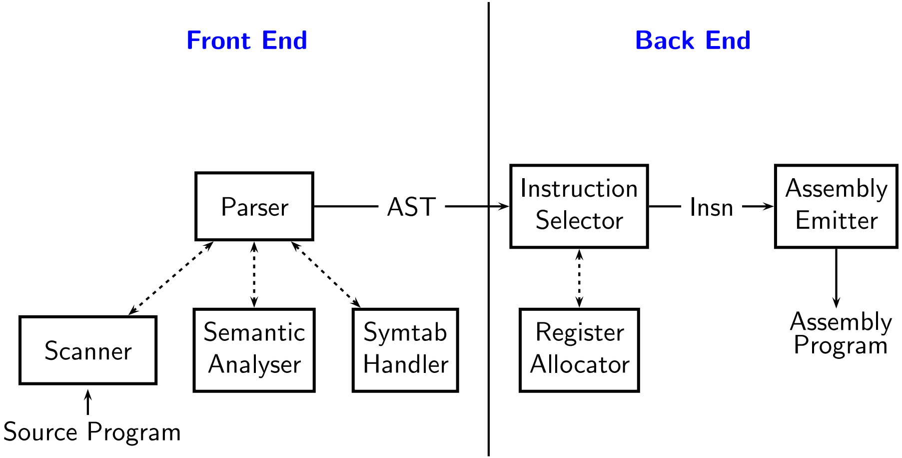
The Structure of a Simple Compiler



The Structure of a Simple Compiler



The Structure of a Simple Compiler



Translation Sequence in Our Compiler: Parsing

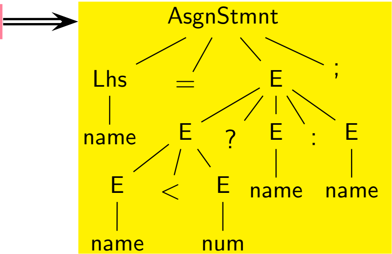
```
a=b<10?b:c;
```

Input

Translation Sequence in Our Compiler: Parsing

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

Input

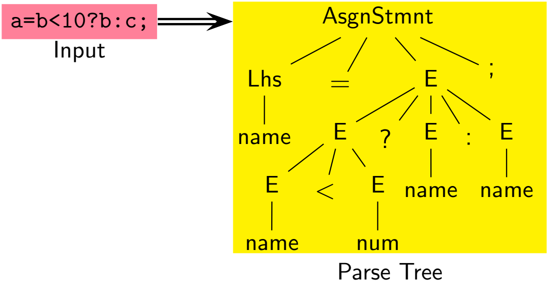


Parse Tree

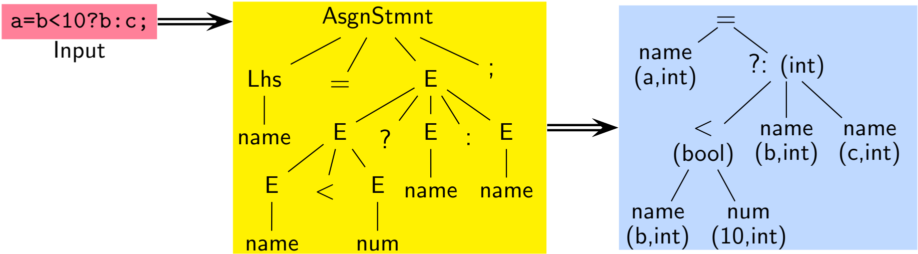
Issues:

- 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: Semantic Analysis



Translation Sequence in Our Compiler: Semantic Analysis



Parse Tree

Abstract Syntax Tree
(with attributes)

Issues:

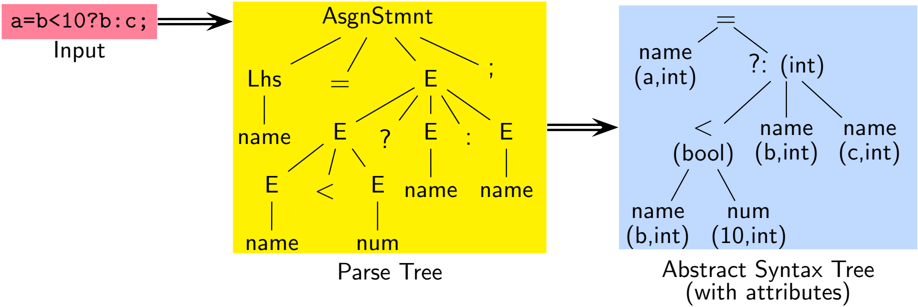
- Symbol tables

  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

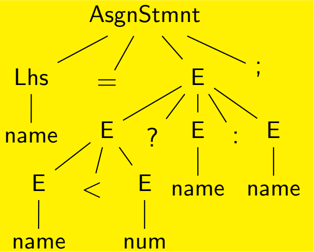
Translation Sequence in Our Compiler: IR Generation



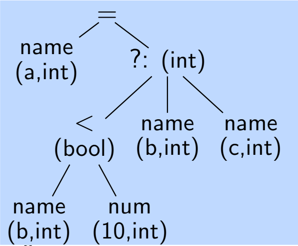
Translation Sequence in Our Compiler: IR Generation

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

Input

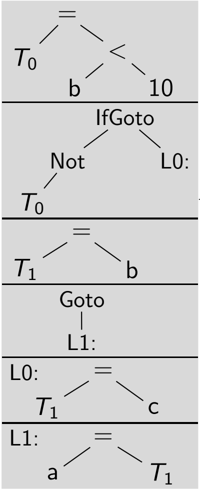


Parse Tree



Abstract Syntax Tree
(with attributes)

Tree List



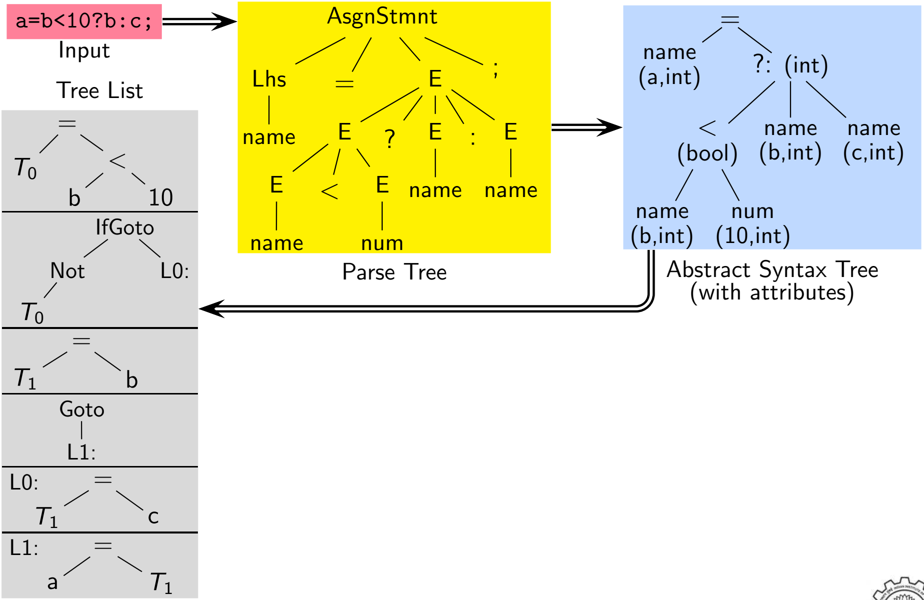
Issues:

- 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

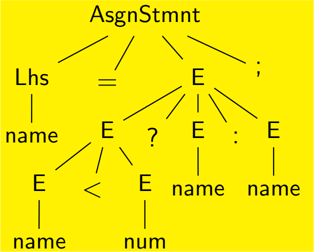
Translation Sequence in Our Compiler: Instruction Selection



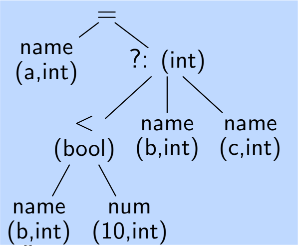
Translation Sequence in Our Compiler: Instruction Selection

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

Input

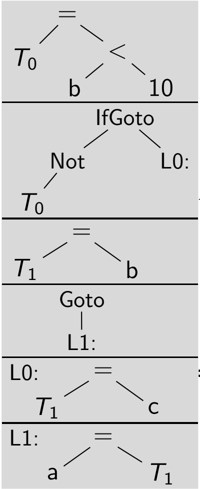


Parse Tree



Abstract Syntax Tree (with attributes)

Tree List



Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow\, !\, T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

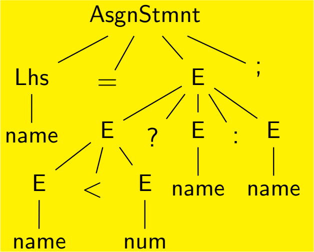
Issues:

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

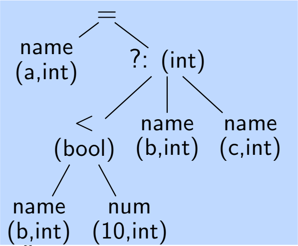
Translation Sequence in Our Compiler: Emitting Instructions

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

Input

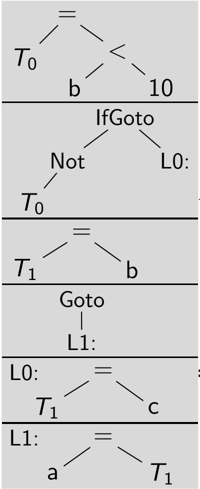


Parse Tree



Abstract Syntax Tree
(with attributes)

Tree List



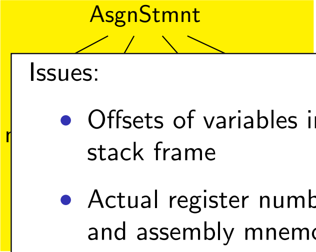
Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow\, !\, T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

Translation Sequence in Our Compiler: Emitting Instructions

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

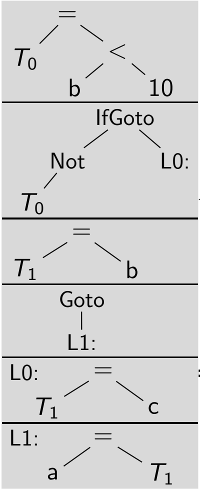
Input

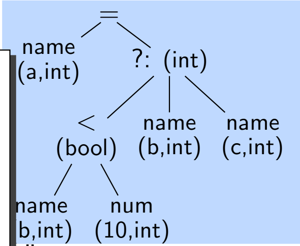


Issues:

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

Tree List





Abstract Syntax Tree (with attributes)

Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow\, ! \, T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

Assembly Code

```
    lw   $t0, 4($fp)
    slti $t0, $t0, 10
    not  $t0, $t0
    bgtz $t0, L0:
    lw   $t0, 4($fp)
    b    L1:
L0: lw   $t0, 8($fp)
L1: sw   0($fp), $t0
```

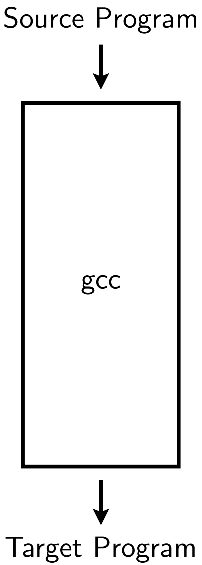
Part 3

GCC ≡ The Great Compiler Challenge

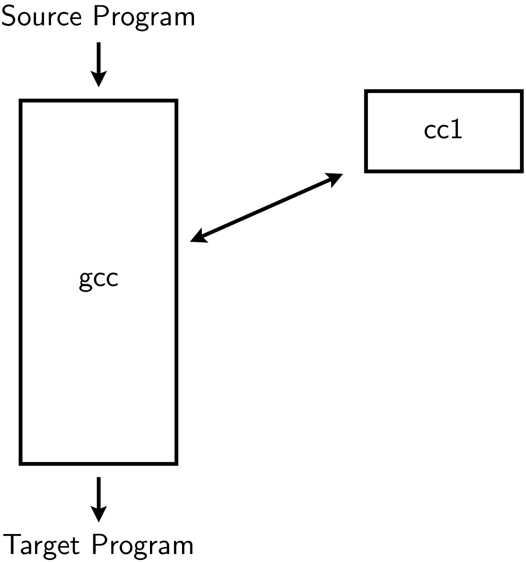
What is GCC?

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

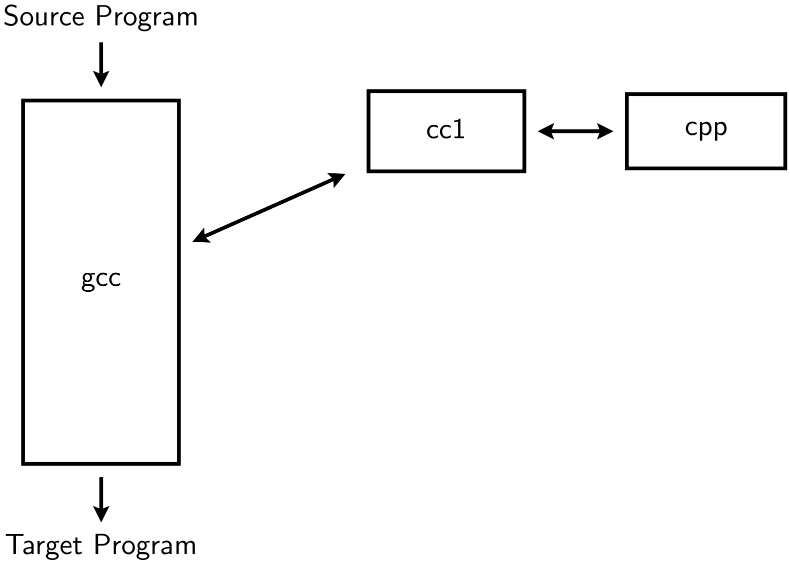
The Gnu Tool Chain



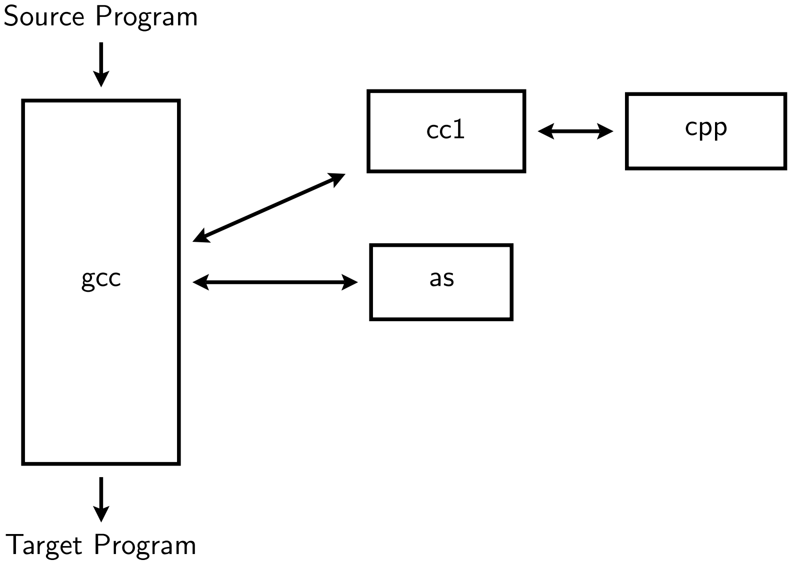
The Gnu Tool Chain



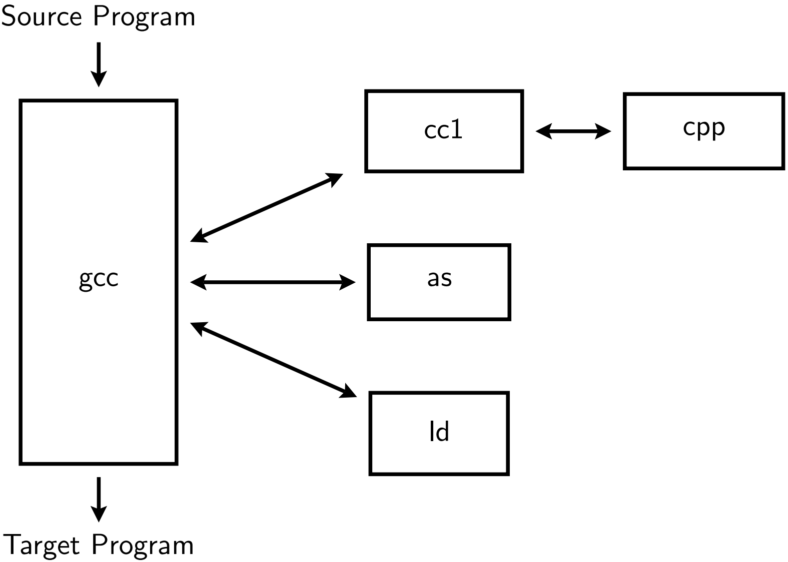
The Gnu Tool Chain



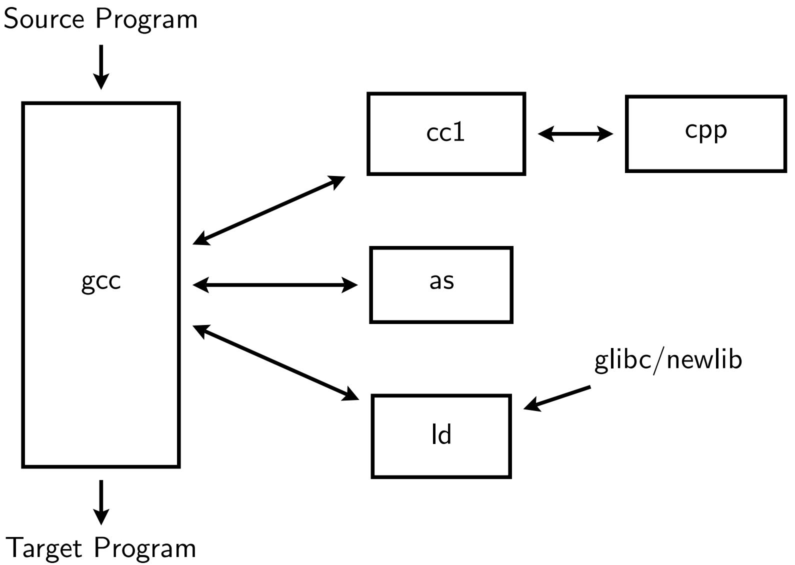
The Gnu Tool Chain



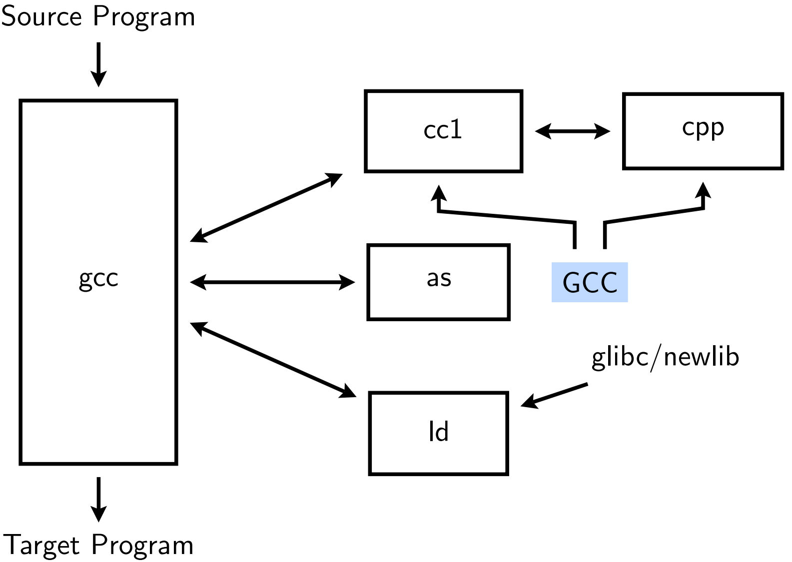
The Gnu Tool Chain



The Gnu Tool Chain



The Gnu Tool Chain



Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:

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:
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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),
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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,
  - Lesser-known target processors:
  - Additional processors independently supported:

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:
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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,
  - Additional processors independently supported:

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:

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,

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,

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,

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,

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,

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,

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,

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,

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),

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,

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,

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

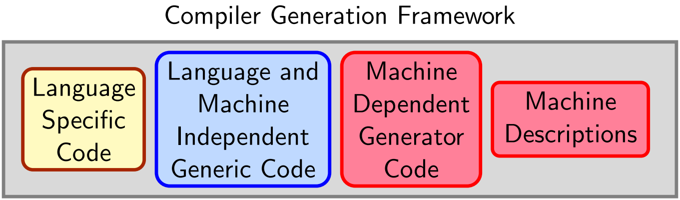
Why is Understanding GCC Difficult?

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

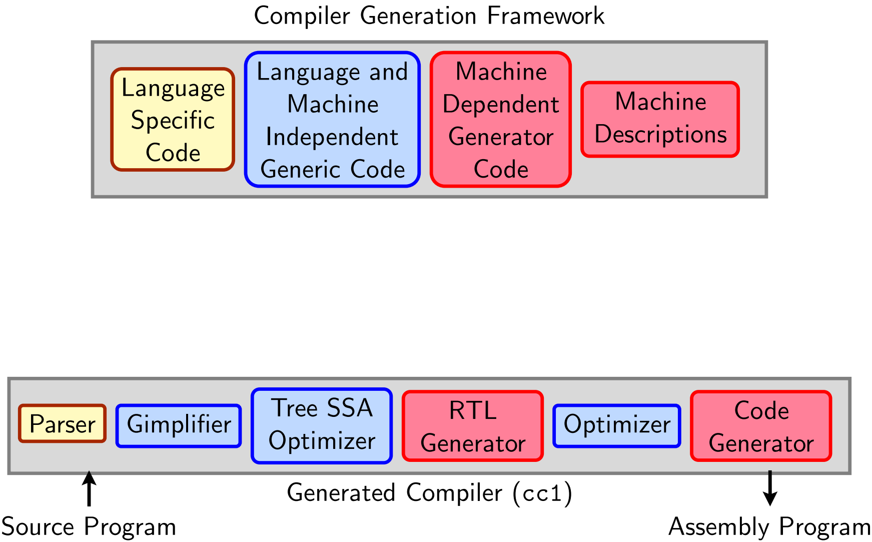
There are two distinct gaps that need to be bridged:

- 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

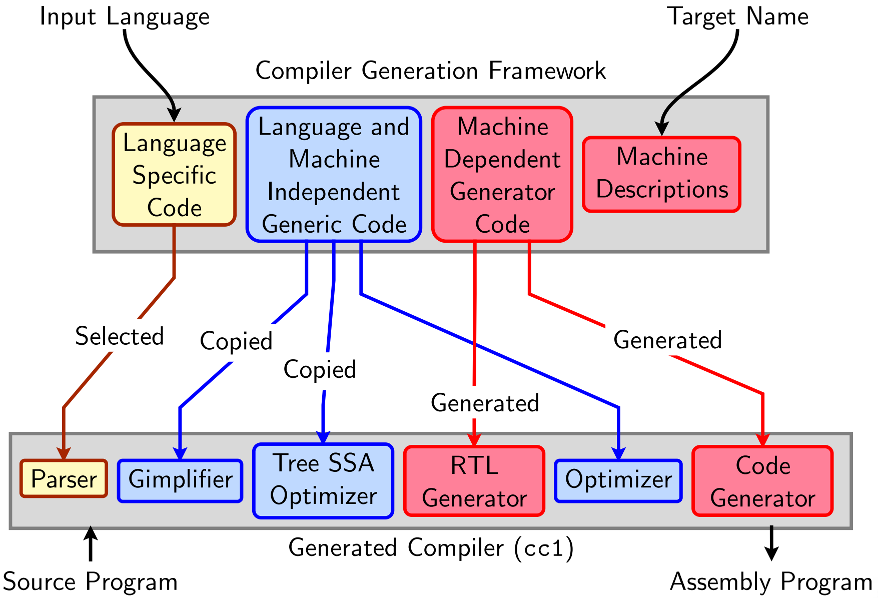
The Architecture of GCC



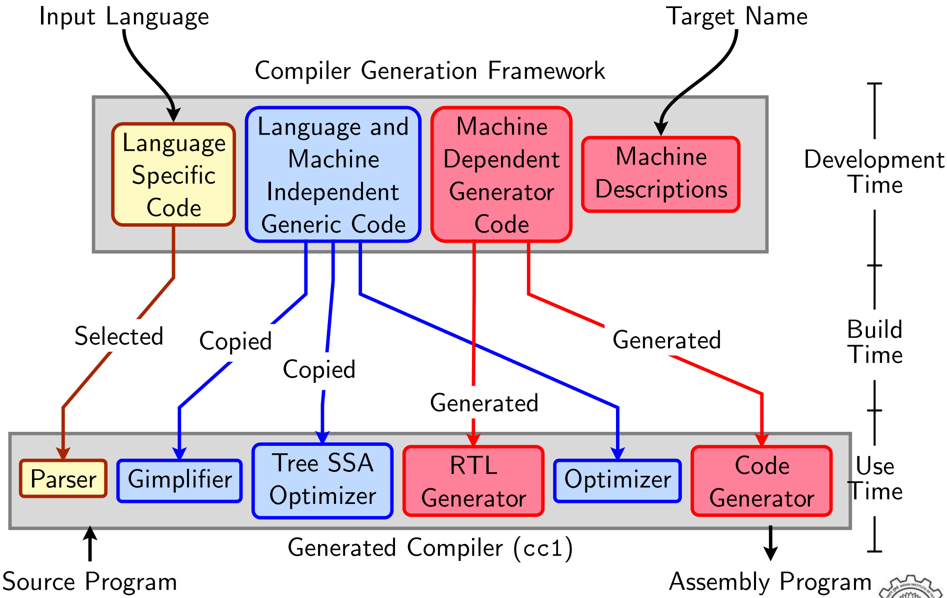
The Architecture of GCC



The Architecture of GCC



The Architecture of GCC



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)

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;
}
```

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

Another Example of The Generation Related Gap

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

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

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;
```

The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism

  Large size of specifications

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

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

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

Part 4

First Level Gray Box Probing of GCC

Outline

- Introduction to Graybox Probing of GCC
- Examining GIMPLE Dumps
  - Translation of data accesses
  - Translation of intraprocedural control flow
  - Translation of interprocedural control flow
- Examining RTL Dumps
- Examining Assembly Dumps

What is Gray Box Probing of GCC?

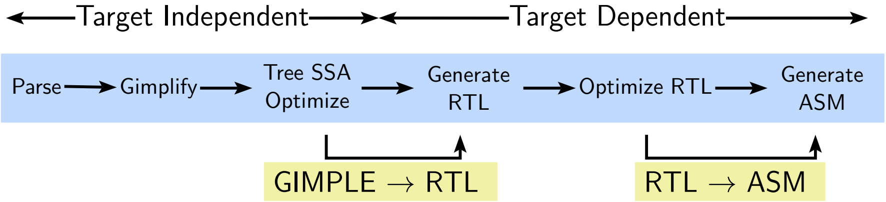
- Black Box probing:
  Examining only the input and output relationship of a system
- White Box probing:
  Examining internals of a system for a given set of inputs
- Gray Box probing:
  Examining input and output of various components/modules
  - Overview of translation sequence in GCC
  - Overview of intermediate representations
  - Intermediate representations of programs across important phases

First Level Gray Box Probing of GCC

- Restricted to the most important translations in GCC

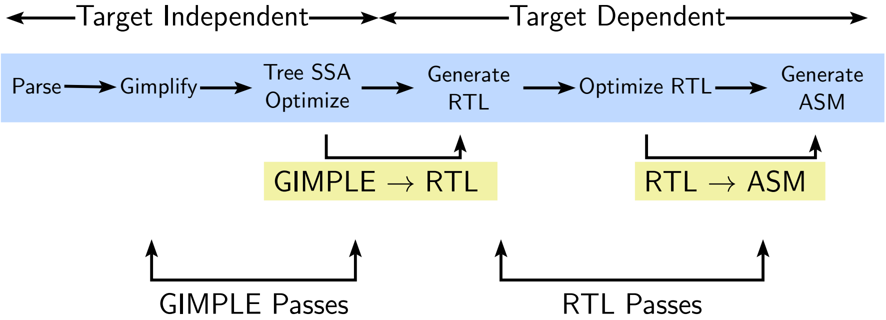
Basic Transformations in GCC

Tranformation from a language to a *different* language



Basic Transformations in GCC

Tranformation from a language to a *different* language



Transformation Passes in GCC 4.5.0

- A total of 203 unique pass names initialized in
  `${SOURCE}/gcc/passes.c`
  Total number of passes is 239.
  - Some passes are called multiple times in different contexts
    Conditional constant propagation and dead code elimination are called thrice
  - Some passes are enabled for specific architectures
  - Some passes have many variations (eg. special cases for loops)
    Common subexpression elimination, dead code elimination
- The pass sequence can be divided broadly in two parts
  - Passes on GIMPLE
  - Passes on RTL
- Some passes are organizational passes to group related passes

Passes On GIMPLE in GCC 4.5.0

Pass Group	Examples	Number of passes
Lowering	GIMPLE IR, CFG Construction	12
Interprocedural Optimizations	Conditional Constant Propagation, Inlining, SSA Construction, LTO	49
Intraprocedural Optimizations	Constant Propagation, Dead Code Elimination, PRE	42
Loop Optimizations	Vectorization, Parallelization	27
Remaining Intraprocedural Optimizations	Value Range Propagation, Rename SSA	23
Generating RTL		01
Total number of passes on GIMPLE		154

Passes On RTL in GCC 4.5.0

Pass Group	Examples	Number of passes
Intraprocedural Optimizations	CSE, Jump Optimization	21
Loop Optimizations	Loop Invariant Movement, Peeling, Unswitching	7
Machine Dependent Optimizations	Register Allocation, Instruction Scheduling, Peephole Optimizations	54
Assembly Emission and Finishing		03
Total number of passes on RTL		85

Finding Out List of Optimizations

- A complete list of optimizations with a brief description

```
gcc -c --help=optimizers
```

- Optimizations enabled at level 2 (other levels are 0, 1, 3, and s)

```
gcc -c -O2 --help=optimizers -Q
```

Producing the Output of GCC Passes

- Use the option `-fdump-<ir>-<passname>`

  `<ir>` could be

  - `tree`: Intraprocedural passes on GIMPLE
  - `ipa`: Interprocedural passes on GIMPLE
  - `rtl`: Intraprocedural passes on RTL

- Use `all` in place of `<pass>` to see all dumps
  Example: `gcc -fdump-tree-all -fdump-rtl-all test.c`

- Dumping more details:
  Suffix `raw` for tree passes and `details` or `slim` for RTL passes
  Individual passes may have more verbosity options (e.g. `-fsched-verbose=5`)

- Use `-S` to stop the compilation with assembly generation

- Use `--verbose-asm` to see more detailed assembly dump

Dumps for Our Code Fragments

GIMPLE dumps (t)

- 001t.tu
- 003t.original
- 004t.gimple
- 006t.vcg
- 008t.omplower
- 009t.lower
- 011t.eh
- 012t.cfg
- 013t.veclower
- 014t.inline_param1
- 021t.cleanup_cfg
- 023t.ssa
- 024t.einline2
- 040t.release_ssa
- 041t.inline_param3
- 135t.cplxlower0
- 140t.optimized
- 219t.statistics

ipa dumps (i)

- 000i.cgraph
- 015i.visibility
- 019i.early_local_cleanups
- 044i.whole-program
- 046i.inline

rtl dumps (r)

- 141r.expand
- 142r.sibling
- 144r.initvals
- 145r.unshare
- 146r.vregs
- 147r.into_cfglayout
- 148r.jump
- 160r.reginfo
- 180r.outof_cfglayout
- 181r.split1
- 183r.dfinit
- 184r.mode_sw
- 185r.asmcons
- 188r.ira
- 191r.split2
- 193r.pro_and_epilogue
- 206r.stack
- 207r.alignments
- 210r.mach
- 211r.barriers
- 215r.shorten
- 216r.nothrow
- 217r.final
- 218r.dfinish

Total Number of Dumps

Optimization Level	Number of Dumps	Goals
Default	47	Fast compilation
O1	134	
O2	156	
O3	165	
Os	154	Optimize for space

Selected Dumps for Our Example Program

GIMPLE dumps (t)

001t.tu
003t.original
004t.gimple
006t.vcg
008t.omplower
009t.lower
011t.eh
012t.cfg
013t.veclower
014t.inline_param1
021t.cleanup_cfg
023t.ssa
024t.einline2
040t.release_ssa
041t.inline_param3
135t.cplxlower0
140t.optimized
219t.statistics

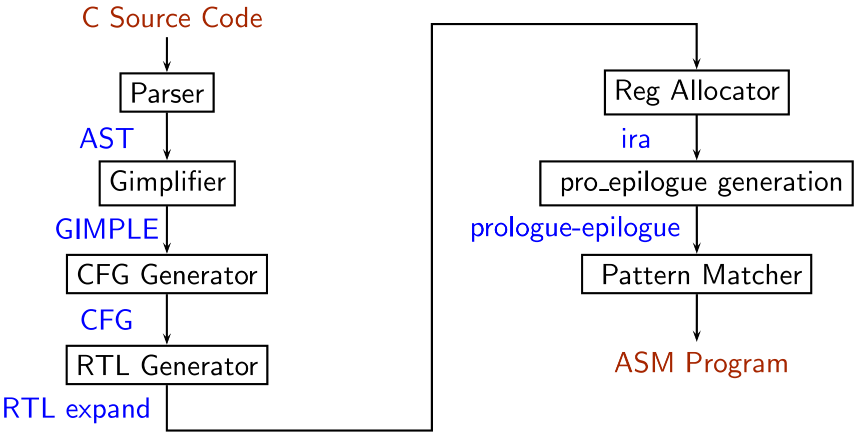
ipa dumps (i)

000i.cgraph
015i.visibility
019i.early_local_cleanups
044i.whole-program
046i.inline

rtl dumps (r)

141r.expand
142r.sibling
144r.initvals
145r.unshare
146r.vregs
147r.into_cfglayout
148r.jump
160r.reginfo
180r.outof_cfglayout
181r.split1
183r.dfinit
184r.mode_sw
185r.asmcons
188r.ira
191r.split2
193r.pro_and_epilogue
206r.stack
207r.alignments
210r.mach
211r.barriers
215r.shorten
216r.nothrow
217r.final
218r.dfinish
assembly output

Passes for First Level Graybox Probing of GCC



Lowering of abstraction!

Gimplifier

- About GIMPLE
  - Three-address representation derived from GENERIC
    Computation represented as a sequence of basic operations
    Temporaries introduced to hold intermediate values
  - Control construct are explicated into conditional jumps
- Examining GIMPLE Dumps
  - Examining translation of data accesses
  - Examining translation of control flow
  - Examining translation of function calls

GIMPLE: Composite Expressions Involving Local and Global Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
x = 10;
y = 5;
D.1954 = x * y;
a.0 = a;
x = D.1954 + a.0;
a.1 = a;
D.1957 = a.1 * x;
y = y - D.1957;
```

GIMPLE: Composite Expressions Involving Local and Global Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
x = 10;
y = 5;
D.1954 = x * y;
a.0 = a;
x = D.1954 + a.0;
a.1 = a;
D.1957 = a.1 * x;
y = y - D.1957;
```

GIMPLE: Composite Expressions Involving Local and Global Variables

test.c

```
int a;

int main()
{
   int x = 10;
   int y = 5;

   x = a + x * y;
   y = y - a * x;

}
```

test.c.004t.gimple

```
   x = 10;
   y = 5;
   D.1954 = x * y;
   a.0 = a;
   x = D.1954 + a.0;
   a.1 = a;
   D.1957 = a.1 * x;
   y = y - D.1957;
```

GIMPLE: Composite Expressions Involving Local and Global Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
x = 10;
y = 5;
D.1954 = x * y;
a.0 = a;
x = D.1954 + a.0;
a.1 = a;
D.1957 = a.1 * x;
y = y - D.1957;
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
a[2] = 10;
D.1952 = a[2];
a[1] = D.1952;
D.1953 = a[1];
D.1954 = a[2];
x = D.1953 + D.1954;
D.1955 = x + 1;
D.1956 = a[1];
D.1957 = D.1955 * D.1956;
a[0] = D.1957;
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
a[2] = 10;
D.1952 = a[2];
a[1] = D.1952;
D.1953 = a[1];
D.1954 = a[2];
x = D.1953 + D.1954;
D.1955 = x + 1;
D.1956 = a[1];
D.1957 = D.1955 * D.1956;
a[0] = D.1957;
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
a[2] = 10;
D.1952 = a[2];
a[1] = D.1952;
D.1953 = a[1];
D.1954 = a[2];
x = D.1953 + D.1954;
D.1955 = x + 1;
D.1956 = a[1];
D.1957 = D.1955 * D.1956;
a[0] = D.1957;
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
a[2] = 10;
D.1952 = a[2];
a[1] = D.1952;
D.1953 = a[1];
D.1954 = a[2];
x = D.1953 + D.1954;
D.1955 = x + 1;
D.1956 = a[1];
D.1957 = D.1955 * D.1956;
a[0] = D.1957;
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
a[2] = 10;
D.1952 = a[2];
a[1] = D.1952;
D.1953 = a[1];
D.1954 = a[2];
x = D.1953 + D.1954;
D.1955 = x + 1;
D.1956 = a[1];
D.1957 = D.1955 * D.1956;
a[0] = D.1957;
```

GIMPLE: 2-D Array Accesses

test.c

```
int main()
{
  int a[3][3], x, y;
  a[0][0] = 7;
  a[1][1] = 8;
  a[2][2] = 9;
  x = a[0][0] / a[1][1];
  y = a[1][1] % a[2][2];
}
```

test.c.004t.gimple

```
a[0][0] = 7;
a[1][1] = 8;
a[2][2] = 9;
D.1953 = a[0][0];
D.1954 = a[1][1];
x = D.1953 / D.1954;
D.1955 = a[1][1];
D.1956 = a[2][2];
y = D.1955 % D.1956;
```

GIMPLE: Use of Pointers

test.c

```
int main()
{
  int **a,*b,c;
  b = &c;
  a = &b;
  **a = 10;  /* c = 10 */
}
~
```

test.c.004t.gimple

```
main ()
{
  int * D.1953;
  int * * a;
  int * b;
  int c;

  b = &c;
  a = &b;
  D.1953 = *a;
  *D.1953 = 10;
}
```

GIMPLE: Use of Pointers

test.c

```
int main()
{
  int **a,*b,c;
  b = &c;
  a = &b;
  **a = 10;  /* c = 10 */
}
~
```

test.c.004t.gimple

```
main ()
{
  int * D.1953;
  int * * a;
  int * b;
  int c;

  b = &c;
  a = &b;
  D.1953 = *a;
  *D.1953 = 10;
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} addr;

typedef struct student
{ int roll;
  addr *city;
} stud;

int main()
{ stud *s;

  s = malloc(sizeof(stud));
  s->roll = 1;
  s->city=malloc(sizeof(addr));
  s->city->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
 void * D.2052;
 void * D.2053;
 struct addr * D.2054;
 struct addr * D.2055;
 struct stud * s;

 D.2052 = malloc (8);
 s = (struct stud *) D.2052;
 s->roll = 1;
 D.2053 = malloc (4);
 D.2054 = (struct addr *) D.2053;
 s->city = D.2054;
 D.2055 = s->city;
 D.2055->name = &"Mumbai"[0];
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} addr;

typedef struct student
{ int roll;
  addr *city;
} stud;

int main()
{ stud *s;

  s = malloc(sizeof(stud));
  s->roll = 1;
  s->city=malloc(sizeof(addr));
  s->city->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
 void * D.2052;
 void * D.2053;
 struct addr * D.2054;
 struct addr * D.2055;
 struct stud * s;

 D.2052 = malloc (8);
 s = (struct stud *) D.2052;
 s->roll = 1;
 D.2053 = malloc (4);
 D.2054 = (struct addr *) D.2053;
 s->city = D.2054;
 D.2055 = s->city;
 D.2055->name = &"Mumbai"[0];
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} addr;

typedef struct student
{ int roll;
  addr *city;
} stud;

int main()
{ stud *s;

  s = malloc(sizeof(stud));
  s->roll = 1;
  s->city=malloc(sizeof(addr));
  s->city->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
 void * D.2052;
 void * D.2053;
 struct addr * D.2054;
 struct addr * D.2055;
 struct stud * s;

 D.2052 = malloc (8);
 s = (struct stud *) D.2052;
 s->roll = 1;
 D.2053 = malloc (4);
 D.2054 = (struct addr *) D.2053;
 s->city = D.2054;
 D.2055 = s->city;
 D.2055->name = &"Mumbai"[0];
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} addr;

typedef struct student
{ int roll;
  addr *city;
} stud;

int main()
{ stud *s;

  s = malloc(sizeof(stud));
  s->roll = 1;
  s->city=malloc(sizeof(addr));
  s->city->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
 void * D.2052;
 void * D.2053;
 struct addr * D.2054;
 struct addr * D.2055;
 struct stud * s;

 D.2052 = malloc (8);
 s = (struct stud *) D.2052;
 s->roll = 1;
 D.2053 = malloc (4);
 D.2054 = (struct addr *) D.2053;
 s->city = D.2054;
 D.2055 = s->city;
 D.2055->name = &"Mumbai"[0];
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];

  p_a = &a[0];
  *p_a = 10;
  D.2048 = p_a + 4;
  *D.2048 = 20;
  D.2049 = p_a + 8;
  *D.2049 = 30;
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];

  p_a = &a[0];
  *p_a = 10;
  D.2048 = p_a + 4;
  *D.2048 = 20;
  D.2049 = p_a + 8;
  *D.2049 = 30;
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];

  p_a = &a[0];
  *p_a = 10;
  D.2048 = p_a + 4;
  *D.2048 = 20;
  D.2049 = p_a + 8;
  *D.2049 = 30;
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];

  p_a = &a[0];
  *p_a = 10;
  D.2048 = p_a + 4;
  *D.2048 = 20;
  D.2049 = p_a + 8;
  *D.2049 = 30;
}
```

GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
         a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
         a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
          a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b
a = D.1199 + c;
<D.1201>:
```

GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
         a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```

GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
         a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```

GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
         a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```

GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
         a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.012t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.012t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.012t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.012t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

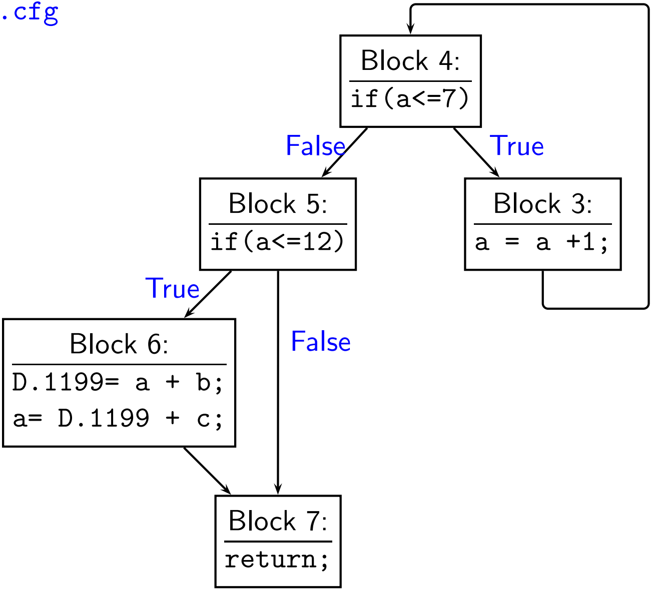
```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.012t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Pictorial View

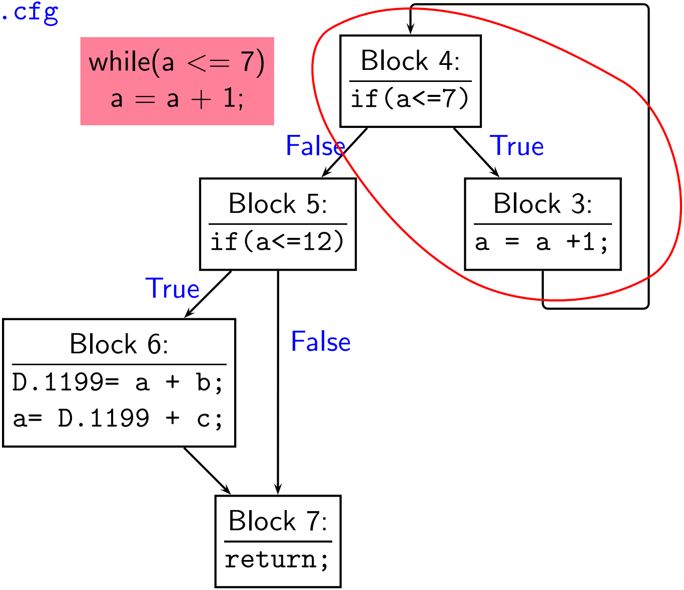
test.c.012t.cfg



Control Flow Graph: Pictorial View

test.c.012t.cfg

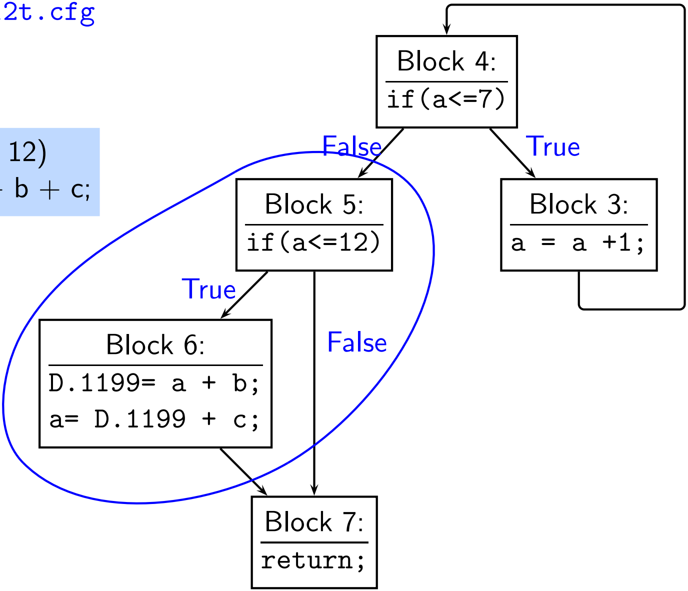
while(a <= 7)
a = a + 1;



Control Flow Graph: Pictorial View

test.c.012t.cfg

if(a <= 12)
a = a + b + c;



GIMPLE: Function Calls and Call Graph

test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3(-1)  @0xb73c7ac8 availabilit
  called by: main/1  (1.00 per call)
  calls:
divide/2(-1)  @0xb73c7a10 availabilit
  called by: main/1  (1.00 per call)
  calls:
main/1(1)  @0xb73c7958 availability:a
  called by:
  calls: printf/3  (1.00 per call)
         multiply/0 (1.00 per call)
         divide/2 (1.00 per call)
multiply/0(0)  @0xb73c78a0 vailabilit
  called by: main/1  (1.00 per call)
  calls:
```

GIMPLE: Function Calls and Call Graph

test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3(-1)  @0xb73c7ac8 availabilit
  called by: main/1   (1.00 per call)
  calls:
divide/2(-1)  @0xb73c7a10 availabilit
  called by: main/1   (1.00 per call)
  calls:
main/1(1)  @0xb73c7958 availability:a
  called by:
  calls: printf/3   (1.00 per call)
         multiply/0 (1.00 per call)
         divide/2 (1.00 per call)
multiply/0(0)  @0xb73c78a0 vailabilit
  called by: main/1   (1.00 per call)
  calls:
```

GIMPLE: Function Calls and Call Graph

test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3(-1)  @0xb73c7ac8 availabilit
  called by: main/1  (1.00 per call)
  calls:
divide/2(-1)  @0xb73c7a10 availabilit
  called by: main/1  (1.00 per call)
  calls:
main/1(1)  @0xb73c7958 availability:a
  called by:
  calls: printf/3  (1.00 per call)
         multiply/0 (1.00 per call)
         divide/2 (1.00 per call)
multiply/0(0)  @0xb73c78a0 vailabilit
  called by: main/1  (1.00 per call)
  calls:
```

GIMPLE: Function Calls and Call Graph

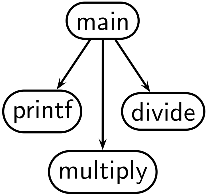
test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3(-1)
  called by: main/1
  calls:
divide/2(-1)
  called by: main/1
  calls:
main/1(1)
  called by:
  calls: printf/3
         multiply/0
         divide/2
multiply/0(0)
  called by: main/1
  calls:
```

call graph



GIMPLE: Function Calls and Call Graph

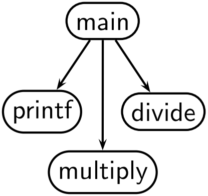
test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3(-1)
  called by: main/1
  calls:
divide/2(-1)
  called by: main/1
  calls:
main/1(1)
  called by:
  calls: printf/3
         multiply/0
         divide/2
multiply/0(0)
  called by: main/1
  calls:
```

call graph

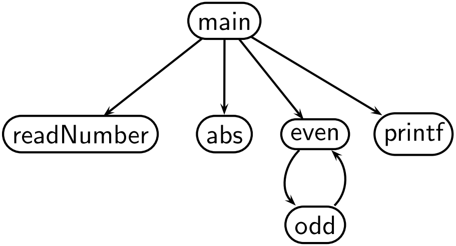


GIMPLE: Call Graphs for Recursive Functions

test.c

```
int even(int n)
{  if (n == 0) return 1;
   else return (!odd(n-1));
}

int odd(int n)
{  if (n == 1) return 1;
   else return (!even(n-1));
}

main()
{ int n;

  n = abs(readNumber());
  if (even(n))
    printf ("n is even\n");
  else printf ("n is odd\n");
}
```

call graph



Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10$, $y = 5$

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	2
y	3
$(y+x)$	
$(y+x)+y$	

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	3
y	3
$(y + x)$	
$(y + x) + y$	

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	3
y	3
$(y+x)$	6
$(y+x)+y$	

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	3
y	4
$(y+x)$	6
$(y+x)+y$	

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	3
y	5
$(y+x)$	6
$(y+x)+y$	

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

x	3
y	5
$(y+x)$	6
$(y+x)+y$	11

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10$, $y = 5$

x	3
y	5
$(y+x)$	6
$(y+x)+y$	11

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10$, $y = 5$

```
x = 2;
y = 3;
x = x + 1; /* 3 */
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y = 5

```
x = 2;
y = 3;
x = x + 1; /* 3 */
D.1572 = y + x;  /* 6 */
y = y + 1;
x = D.1572 + y;
y = y + 1;
```

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

```
x = 2;
y = 3;
x = x + 1; /* 3 */
D.1572 = y + x;  /* 6 */
y = y + 1;  /* 4 */
x = D.1572 + y;
y = y + 1;
```

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y =5

```
x = 2;
y = 3;
x = x + 1; /* 3 */
D.1572 = y + x;  /* 6 */
y = y + 1;  /* 4 */
x = D.1572 + y;  /* 10 */
y = y + 1;
```

Inspect GIMPLE When in Doubt

```
int x=2,y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10 , y = 5

```
x = 2;
y = 3;
x = x + 1; /* 3 */
D.1572 = y + x;  /* 6 */
y = y + 1;  /* 4 */
x = D.1572 + y;  /* 10 */
y = y + 1;  /* 5 */
```

RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

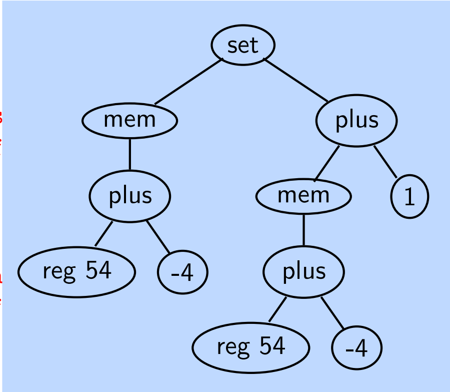
```
(insn 12 11 13 4 t.c:24 (parallel [
    (set (mem/c/i:SI
            (plus:SI
               (reg/f:SI 54 virtual-stack-vars)
               (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
         (plus:SI
            (mem/c/i:SI
               (plus:SI
                  (reg/f:SI 54 virtual-stack-vars)
                  (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
            (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
   ]) -1 (nil))
```

RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-s
              (const_int -4 [0xfffff
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtua
                 (const_int -4 [0xff
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

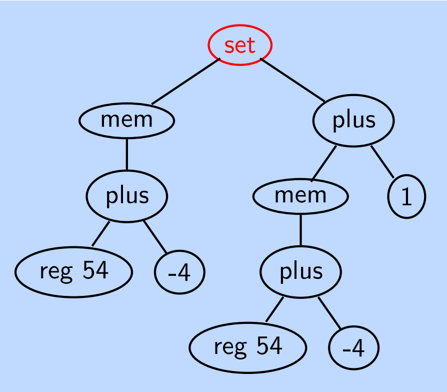


RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-s
              (const_int -4 [0xfffff
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtua
                 (const_int -4 [0xff
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```



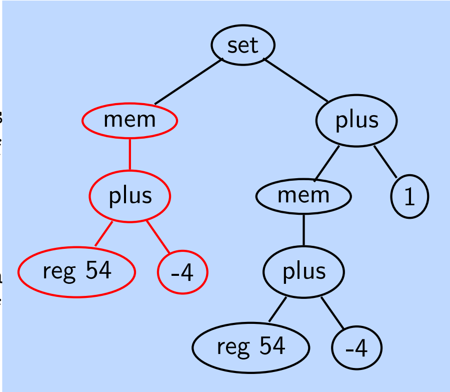
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

a is a local variable allocated on stack

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-s
              (const_int -4 [0xfffff
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtua
                 (const_int -4 [0xff
           (const_int 1 [0x1]))))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```



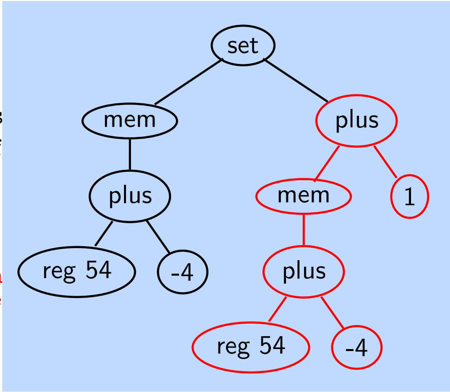
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

a is a local variable allocated on stack

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-s
              (const_int -4 [0xfffff
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtua
                 (const_int -4 [0xff
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```



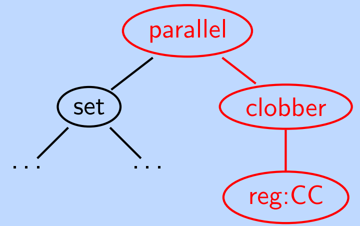
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: `test.c.141r.expand`

side-effect of plus may modify condition code register non-deterministically

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-s
              (const_int -4 [0xffff
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtua
                 (const_int -4 [0xff
           (const_int 1 [0x1]))))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```



RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: test.c.141r.e

Output with slim suffix

```
{[r54:SI-0x4]=[r54:SI-0x4]+0x1;
   clobber flags:CC;
}
```

```
(insn 12 11 13 4 t.c:24 (p
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
          (plus:SI
             (reg/f:SI 54 virtual-stack-vars)
             (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
          (mem/c/i:SI
             (plus:SI
                (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
          (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Current Instruction

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Previous Instruction

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Next Instruction

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Basic Block

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
            (plus:SI
               (reg/f:SI 54 virtual-stack-vars)
               (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
         (plus:SI
            (mem/c/i:SI
               (plus:SI
                  (reg/f:SI 54 virtual-stack-vars)
                  (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
            (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

File name: Line number

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

memory reference that does not trap

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
   (set (mem/c/i:SI
           (plus:SI
              (reg/f:SI 54 virtual-stack-vars)
              (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
           (mem/c/i:SI
              (plus:SI
                 (reg/f:SI 54 virtual-stack-vars)
                 (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

scalar that is not a part of an aggregate

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
    (set (mem/c/i:SI
             (plus:SI
                (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
          (plus:SI
             (mem/c/i:SI
                (plus:SI
                   (reg/f:SI 54 virtual-stack-vars)
                   (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
             (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

register that holds a pointer

Additional Information in RTL

```
(insn 12 11 13 4 t.c:24 (parallel [
    (set (mem/c/i:SI
            (plus:SI
                (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
        (plus:SI
            (mem/c/i:SI
                (plus:SI
                    (reg/f:SI 54 virtual-stack-vars)
                    (const_int -4 [0xfffffffc])) [0 a+0 S4 A32])
            (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

single integer

RTL for i386: Arithmetic Operations (2)

Translation of $a = a + 1$ when a is a global variable

Dump file: `test.c.141r.expand`

```
(insn 11 10 12 4 t.c:26 (set
   (reg:SI 64 [ a.0 ])
   (mem/c/i:SI
      (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0 S4 A32])

(insn 12 11 13 4 t.c:26 (parallel [
   (set (reg:SI 63 [ a.1 ])
        (plus:SI
           (reg:SI 64 [ a.0 ])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))

(insn 13 12 14 4 t.c:26 (set
   (mem/c/i:SI (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0
   (reg:SI 63 [ a.1 ])) -1 (nil))
```

RTL for i386: Arithmetic Operations (2)

Translation of $a = a + 1$ when a is a global variable

Dump file: `test.c.141r.expand`

```
(insn 11 10 12 4 t.c:26 (set
   (reg:SI 64 [ a.0 ])
   (mem/c/i:SI
      (symbol_ref:SI ("a")  <var_decl 0                              2]))

(insn 12 11 13 4 t.c:26 (parallel [
   (set (reg:SI 63 [ a.1 ])
        (plus:SI
           (reg:SI 64 [ a.0 ])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))

(insn 13 12 14 4 t.c:26 (set
   (mem/c/i:SI (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0
   (reg:SI 63 [ a.1 ])) -1 (nil))
```

Load a into reg64

RTL for i386: Arithmetic Operations (2)

Translation of $a = a + 1$ when a is a global variable

Dump file: `test.c.141r.expand`

```
(insn 11 10 12 4 t.c:26 (set
   (reg:SI 64 [ a.0 ])
   (mem/c/i:SI
      (symbol_ref:SI ("a")  <var_decl 0
```

```
(insn 12 11 13 4 t.c:26 (parallel [
   (set (reg:SI 63 [ a.1 ])
        (plus:SI
           (reg:SI 64 [ a.0 ])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))
```

```
(insn 13 12 14 4 t.c:26 (set
   (mem/c/i:SI (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0
   (reg:SI 63 [ a.1 ])) -1 (nil))
```

Load a into reg64

reg63 = reg64 + 1

RTL for i386: Arithmetic Operations (2)

Translation of $a = a + 1$ when a is a global variable

Dump file: `test.c.141r.expand`

```
(insn 11 10 12 4 t.c:26 (set
   (reg:SI 64 [ a.0 ])
   (mem/c/i:SI
      (symbol_ref:SI ("a")  <var_decl 0                        2])

(insn 12 11 13 4 t.c:26 (parallel [
   (set (reg:SI 63 [ a.1 ])
        (plus:SI
           (reg:SI 64 [ a.0 ])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))

(insn 13 12 14 4 t.c:26 (set
   (mem/c/i:SI (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0
   (reg:SI 63 [ a.1 ])) -1 (nil))
```

Load a into reg64

reg63 = reg64 + 1

store reg63 into a

RTL for i386: Arithmetic Operations (2)

Translation of $a = a + 1$ when a is a global variable

Dump file: `test.c.141r.expand`

```
(insn 11 10 12 4 t.c:26 (set
   (reg:SI 64 [ a.0 ])
   (mem/c/i:SI
      (symbol_ref:SI ("a")  <var_decl 0                           2])

(insn 12 11 13 4 t.c:26 (parallel [
   (set (reg:SI 63 [ a.1 ])
        (plus:SI
           (reg:SI 64 [ a.0 ])
           (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))

(insn 13 12 14 4 t.c:26 (set
   (mem/c/i:SI (symbol_ref:SI ("a")  <var_decl 0xb7d8d000 a>) [0 a+0
   (reg:SI 63 [ a.1 ])) -1 (nil))
```

Load a into reg64

reg63 = reg64 + 1

store reg63 into a

Output with `slim` suffix

```
r64:SI=[‘a’]
{r63:SI=r64:SI+0x1;
   clobber flags:CC;
}
[‘a’]=r63:SI
```

RTL for i386: Arithmetic Operations (3)

Translation of $a = a + 1$ when a is a formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel [
   (set
      (mem/c/i:SI
         (reg/f:SI 53 virtual-incoming-args) [0 a+0 S4 A32])
      (plus:SI
         (mem/c/i:SI
            (reg/f:SI 53 virtual-incoming-args) [0 a+0 S4 A32])
               (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

RTL for i386: Arithmetic Operations (3)

Translation of $a = a + 1$ when a is a formal parameter

Dump file: test.c.141r.expand

```
(insn 10 9 11 4 t1.c:25 (parallel [
   (set
      (mem/c/i:SI
         (reg/f:SI 53 virtual-incoming-
      (plus:SI
         (mem/c/i:SI
            (reg/f:SI 53 virtual-incomi
               (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Access through argument pointer register instead of frame pointer register

RTL for i386: Arithmetic Operations (3)

Translation of $a = a + 1$ when a is a formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel [
   (set
      (mem/c/i:SI
         (reg/f:SI 53 virtual-incoming-
      (plus:SI
         (mem/c/i:SI
            (reg/f:SI 53 virtual-incomi
               (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Access through argument pointer register instead of frame pointer register

No offset required?

RTL for i386: Arithmetic Operations (3)

Translation of $a = a + 1$ when a is a formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel [
   (set
      (mem/c/i:SI
         (reg/f:SI 53 virtual-incoming-
      (plus:SI
         (mem/c/i:SI
            (reg/f:SI 53 virtual-incomi
               (const_int 1 [0x1])))
    (clobber (reg:CC 17 flags))
  ]) -1 (nil))
```

Access through argument pointer register instead of frame pointer register

No offset required?

Output with `slim` suffix

```
{[r53:SI]=[r53:SI]+0x1;
clobber flags:CC;
}
```

RTL for i386: Arithmetic Operation (4)

Translation of $a = a + 1$ when a is the second formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel [
   (set
      (mem/c/i:SI
         (plus:SI
            (reg/f:SI 53 virtual-incoming-args)
            (const_int 4 [0x4])) [0 a+0 S4 A32])
      (plus:SI
         (mem/c/i:SI
            (plus:SI
               (reg/f:SI 53 virtual-incoming-args)
               (const_int 4 [0x4])) [0 a+0 S4 A32])
         (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))
```

RTL for i386: Arithmetic Operation (4)

Translation of $a = a + 1$ when a is the second formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel
   (set
      (mem/c/i:SI
         (plus:SI
            (reg/f:SI 53 virtual-
            (const_int 4 [0x4]))
      (plus:SI
         (mem/c/i:SI
            (plus:SI
               (reg/f:SI 53 virtu
               (const_int 4 [0x4]
         (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))
```

Offset 4 added to the argument pointer register

RTL for i386: Arithmetic Operation (4)

Translation of $a = a + 1$ when a is the second formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel
   (set
      (mem/c/i:SI
         (plus:SI
            (reg/f:SI 53 virtual-
            (const_int 4 [0x4]))
      (plus:SI
         (mem/c/i:SI
            (plus:SI
               (reg/f:SI 53 virtu
               (const_int 4 [0x4]
         (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))
```

Offset 4 added to the argument pointer register

When a is the first parameter, its offset is 0!

RTL for i386: Arithmetic Operation (4)

Translation of $a = a + 1$ when a is the second formal parameter

Dump file: `test.c.141r.expand`

```
(insn 10 9 11 4 t1.c:25 (parallel
   (set
      (mem/c/i:SI
         (plus:SI
            (reg/f:SI 53 virtual-
            (const_int 4 [0x4]))
      (plus:SI
         (mem/c/i:SI
            (plus:SI
               (reg/f:SI 53 virtu
               (const_int 4 [0x4]
         (const_int 1 [0x1])))
   (clobber (reg:CC 17 flags))
 ]) -1 (nil))
```

Offset 4 added to the argument pointer register

When a is the first parameter, its offset is 0!

Output with `slim` suffix

```
{[r53:SI+0x4]=[r53:SI+0x4]+0x1;
 clobber flags:CC;
}
```

RTL for spim: Arithmetic Operations

Translation of $a = a + 1$ when a is a local variable

Dump file: `test.c.141r.expand`

r39=stack($fp - 4)
r40=r39+1
stack($fp - 4)=r40

```
(insn 7 6 8 4 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 4 test.c:6 (set (reg:SI 40)
      (plus:SI (reg:SI 39)
           (const_int 1 [...]))) -1 (nil))
(insn 9 8 10 4 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: Arithmetic Operations

Translation of $a = a + 1$ when a is a local variable

Dump file: `test.c.141r.expand`

r39=stack($fp - 4)
r40=r39+1
stack($fp - 4)=r40

```
(insn 7 6 8 4 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 4 test.c:6 (set (reg:SI 40)
      (plus:SI (reg:SI 39)
           (const_int 1 [...]))) -1 (nil))
(insn 9 8 10 4 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: Arithmetic Operations

Translation of $a = a + 1$ when a is a local variable

Dump file: `test.c.141r.expand`

r39=stack($fp - 4)
r40=r39+1
stack($fp - 4)=r40

```
(insn 7 6 8 4 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 4 test.c:6 (set (reg:SI 40)
      (plus:SI (reg:SI 39)
           (const_int 1 [...]))) -1 (nil))
(insn 9 8 10 4 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: Arithmetic Operations

Translation of $a = a + 1$ when a is a local variable

Dump file: `test.c.141r.expand`

r39=stack($fp - 4)
r40=r39+1
stack($fp - 4)=r40

```
(insn 7 6 8 4 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 4 test.c:6 (set (reg:SI 40)
      (plus:SI (reg:SI 39)
           (const_int 1 [...]))) -1 (nil))
(insn 9 8 10 4 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 i386: Control Flow

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: Control Flow

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: Control Flow

Translation of `if (a > b) { /* something */ }`

Dump file: `test.c.141r.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 [0xfffffff8])) [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 [0xfffffffc])) [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: Control Flow

Translation of `if (a > b) { /* something */ }`

Dump file: `test.c.141r.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 [0xfffffff8])) [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 [0xfffffffc])) [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: Control Flow

Translation of `if (a > b) { /* something */ }`

Dump file: `test.c.141r.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 [0xfffffff8])) [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 [0xfffffffc])) [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))
```

Observing Register Allocation for i386

test.c

```
int main()
{
   int a=2, b=3;
   if(a<=12)
     a = a * b;
}
```

test.c.185r.asmcons

(observable dump before register allocation)

```
(insn 10 9 11 3 test.c:5 (set (reg:SI 59)
  (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
     (const_int -4 [0xfffffffc])) [0 a+0 S4 A32

(insn 11 10 12 3 test.c:5 (parallel [
  (set (reg:SI 60)
     (mult:SI (reg:SI 59)
        (mem/c/i:SI (plus:SI (reg/f:SI 20 frame
           (const_int -8 [0xfffffff8])) [0 b+0
  (clobber (reg:CC 17 flags))
  ]) 262 *mulsi3_1 (nil))

(insn 12 11 22 3 test.c:5 (set
      (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xfffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi_1 (nil))
```

Observing Register Allocation for i386

test.c

```
int main()
{
   int a=2, b=3;
   if(a<=12)
     a = a * b;
}
```

test.c.185r.asmcons

(observable dump before register allocation)

```
(insn 10 9 11 3 test.c:5 (set (reg:SI 59)
  (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
     (const_int -4 [0xfffffffc])) [0 a+0 S4 A32

(insn 11 10 12 3 test.c:5 (parallel [
  (set (reg:SI 60)
     (mult:SI (reg:SI 59)
        (mem/c/i:SI (plus:SI (reg/f:SI 20 frame
           (const_int -8 [0xfffffff8])) [0 b+0
  (clobber (reg:CC 17 flags))
  ]) 262 *mulsi3_1 (nil))

(insn 12 11 22 3 test.c:5 (set
      (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xfffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi_1 (nil))
```

Observing Register Allocation for i386

test.c

```
int main()
{
   int a=2, b=3;
   if(a<=12)
     a = a * b;
}
```

test.c.185r.asmcons

(observable dump before register allocation)

```
(insn 10 9 11 3 test.c:5 (set (reg:SI 59)
  (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
     (const_int -4 [0xfffffffc])) [0 a+0 S4 A32

(insn 11 10 12 3 test.c:5 (parallel [
  (set (reg:SI 60)
     (mult:SI (reg:SI 59)
        (mem/c/i:SI (plus:SI (reg/f:SI 20 frame
           (const_int -8 [0xfffffff8])) [0 b+0
  (clobber (reg:CC 17 flags))
  ]) 262 *mulsi3_1 (nil))

(insn 12 11 22 3 test.c:5 (set
      (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xfffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi_1 (nil))
```

Observing Register Allocation for i386

test.c

```
int main()
{
   int a=2, b=3;
   if(a<=12)
     a = a * b;
}
```

test.c.185r.asmcons

(observable dump before register allocation)

```
(insn 10 9 11 3 test.c:5 (set (reg:SI 59)
  (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
     (const_int -4 [0xfffffffc])) [0 a+0 S4 A32

(insn 11 10 12 3 test.c:5 (parallel [
  (set (reg:SI 60)
     (mult:SI (reg:SI 59)
        (mem/c/i:SI (plus:SI (reg/f:SI 20 frame
           (const_int -8 [0xfffffff8])) [0 b+0
  (clobber (reg:CC 17 flags))
  ]) 262 *mulsi3_1 (nil))

(insn 12 11 22 3 test.c:5 (set
      (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xfffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi_1 (nil))
```

Observing Register Allocation for i386

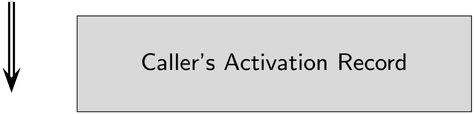
test.c.185r.asmcons

```
(set (reg:SI 59) (mem/c/i:SI
   (plus:SI
     (reg/f:SI 20 frame)
     (const_int -4))))

(set (reg:SI 60)
  (mult:SI
     (reg:SI 59)
     (mem/c/i:SI
       (plus:SI
         (reg/f:SI 20 frame)
         (const_int -8)) )))

(set (mem/c/i:SI (plus:SI
      (reg/f:SI 20 frame)
       (const_int -4)))
   (reg:SI 60))
```

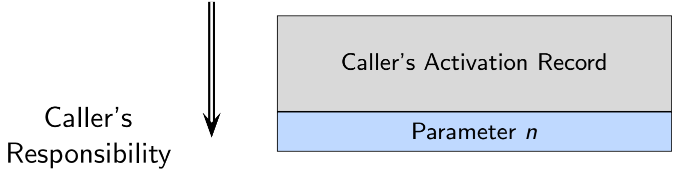
test.c.188r.ira

```
(set (reg:SI 0 ax [59]) (mem/c/i:SI
   (plus:SI
     (reg/f:SI 6 bp)
     (const_int -4))))

(set (reg:SI 0 ax [60])
  (mult:SI
     (reg:SI 0 ax [59])
     (mem/c/i:SI
       (plus:SI
         (reg/f:SI 6 bp)
         (const_int -8)) )))

(set (mem/c/i:SI (plus:SI
      (reg/f:SI 6 bp)
       (const_int -4)))
   (reg:SI 0 ax [60]))
```

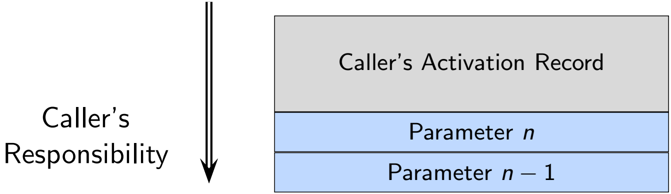
Activation Record Structure in Spim



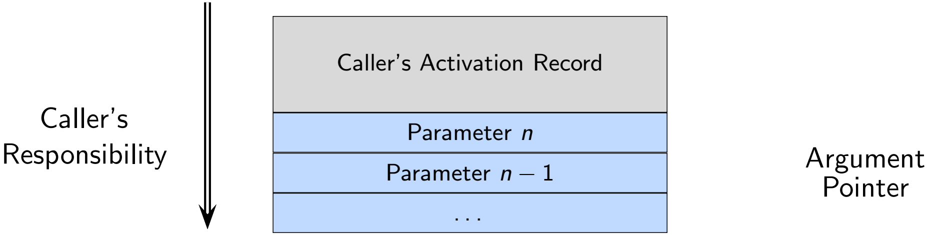
Activation Record Structure in Spim



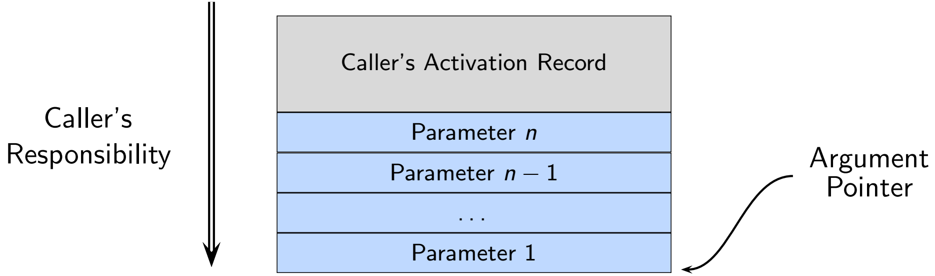
Activation Record Structure in Spim



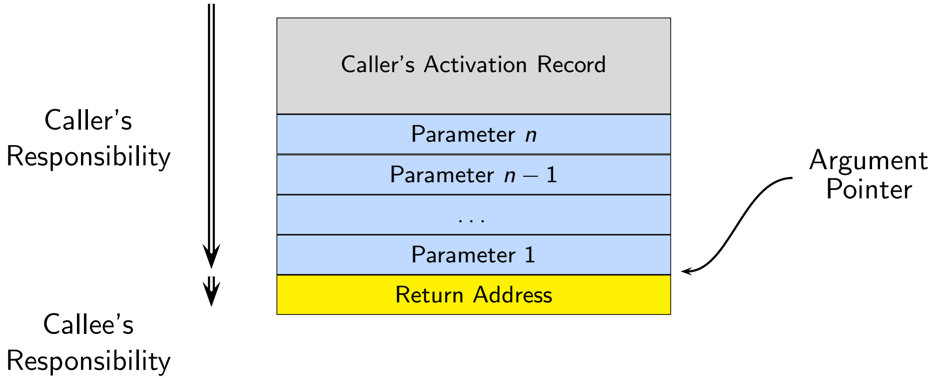
Activation Record Structure in Spim



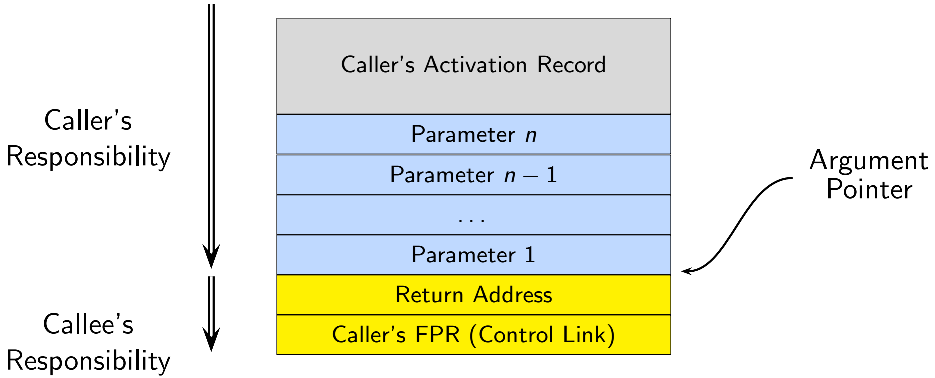
Activation Record Structure in Spim



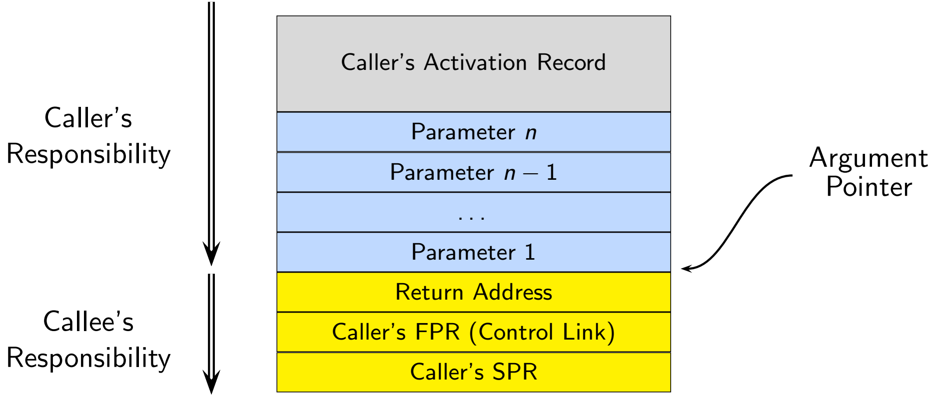
Activation Record Structure in Spim



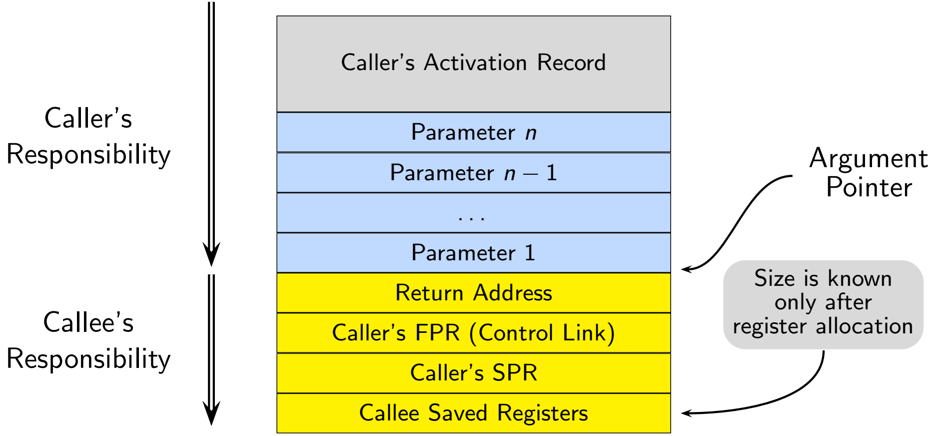
Activation Record Structure in Spim



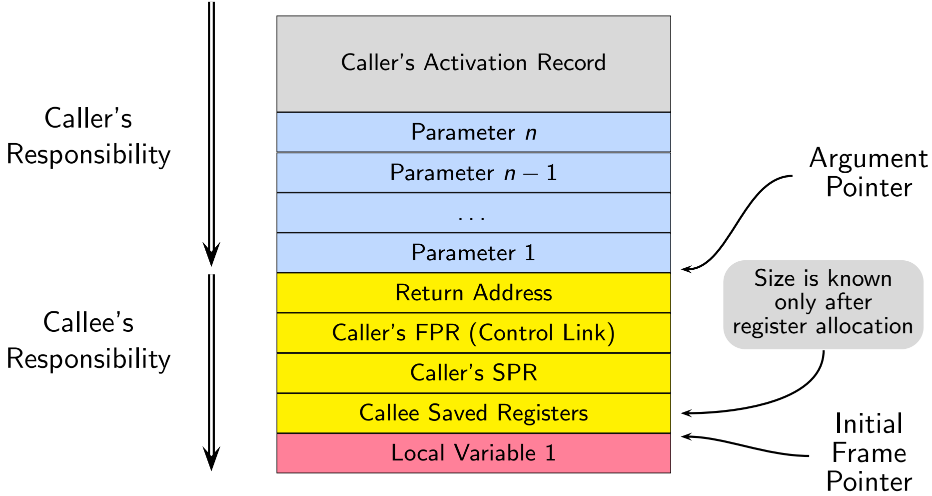
Activation Record Structure in Spim



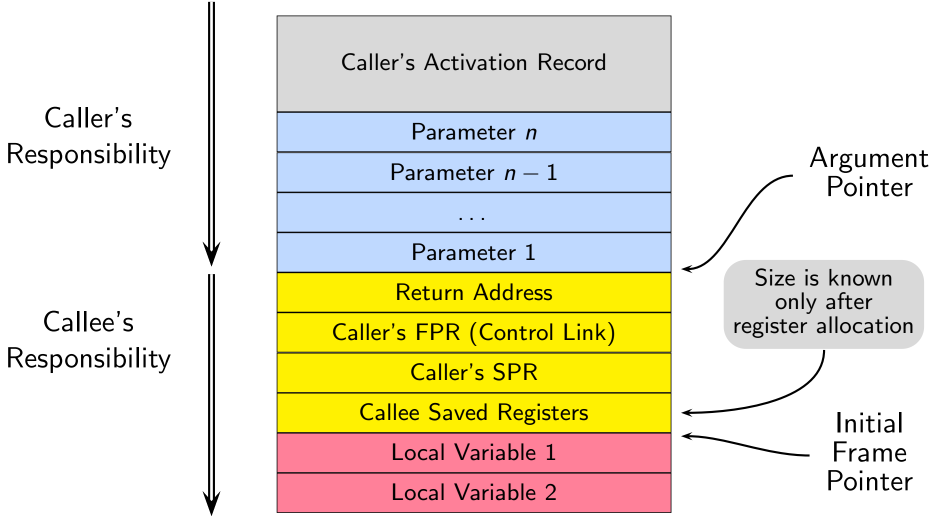
Activation Record Structure in Spim



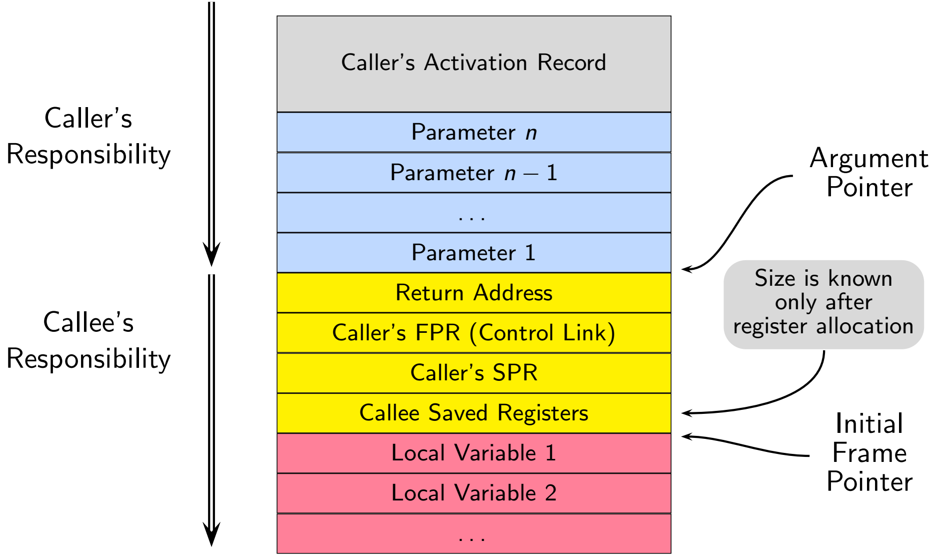
Activation Record Structure in Spim



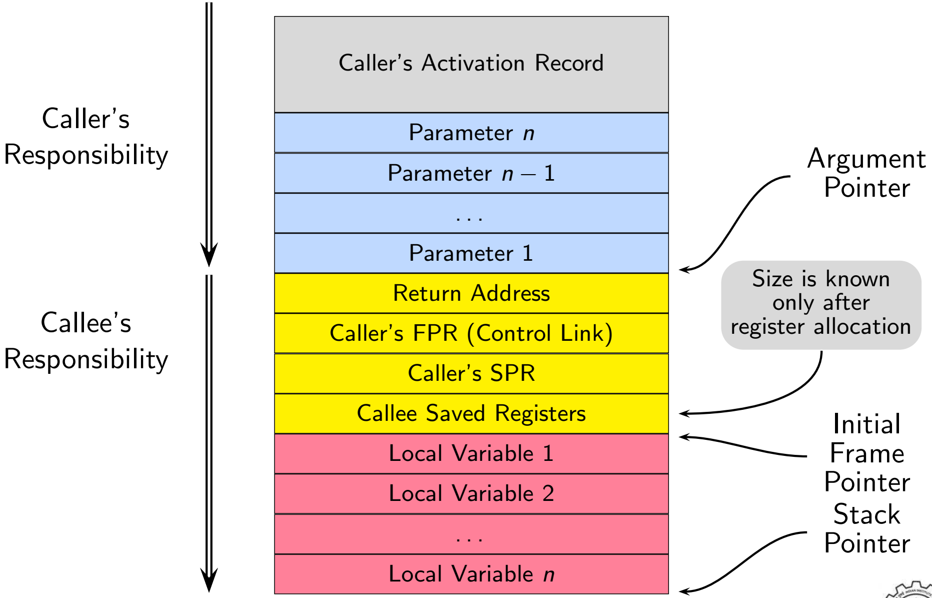
Activation Record Structure in Spim



Activation Record Structure in Spim



Activation Record Structure in Spim



RTL for Function Calls in spim

Calling function	Called function
	• Allocate memory for return value (push)
• Allocate memory for actual parameters on stack	• Store mandatory callee save registers (push)
• Copy actual parameters	• Set frame pointer
• **Call function**	• Allocate local variables (push)
• Get result from stack (pop)	• **Execute code**
• Deallocate memory for activation record (pop)	• Put result in return value space
	• Deallocate local variables (pop)
	• Load callee save registers (pop)
	• Return

Prologue and Epilogue: spim

Dump file: test.c.193r.pro_and_epilogue

```
(insn 17 3 18 2 test.c:2
   (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8])
        (reg:SI 31 $ra)) -1 (nil))
(insn 18 17 19 2 test.c:2
   (set (mem:SI (plus:SI (reg/f:SI 29 $sp)
                     (const_int -4 [...])) [...])
      (reg/f:SI 29 $sp)) -1 (nil))
(insn 19 18 20 2 test.c:2 (set
      (mem:SI (plus:SI (reg/f:SI 29 $sp)
                   (const_int -8 [...])) [...])
      (reg/f:SI 30 $fp)) -1 (nil))
(insn 20 19 21 2 test.c:2 (set (reg/f:SI 30 $fp)
       (reg/f:SI 29 $sp)) -1 (nil))
(insn 21 20 22 2 test.c:2 (set (reg/f:SI 29 $sp)
    (plus:SI (reg/f:SI 30 $fp)
            (const_int -32 [...]))) -1 (nil))
```

Prologue and Epilogue: spim

Dump file: `test.c.193r.pro_and_epilogue`

```
(insn 17 3 18 2 test.c:2
   (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8])
        (reg:SI 31 $ra)) -1 (nil))
(insn 18 17 19 2 test.c:2
   (set (mem:SI (plus:SI (reg/f:SI 29 $sp)
                    (const_int -4 [...])) [...])
      (reg/f:SI 29 $sp)) -1 (nil))
(insn 19 18 20 2 test.c:2 (set
      (mem:SI (plus:SI (reg/f:SI 29 $sp)
                  (const_int -8 [...])) [...])
      (reg/f:SI 30 $fp)) -1 (nil))
(insn 20 19 21 2 test.c:2 (set (reg/f:SI 30 $fp)
       (reg/f:SI 29 $sp)) -1 (nil))
(insn 21 20 22 2 test.c:2 (set (reg/f:SI 29 $sp)
    (plus:SI (reg/f:SI 30 $fp)
            (const_int -32 [...]))) -1 (nil))
```

```
sw $ra, 0($sp)
sw $sp, 4($sp)
sw $fp, 8($sp)
move $fp,$sp
addi $sp,$fp,32
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%ebp)
.L2:
        cmpl  $7, -4(%ebp)
        jle   .L3
        cmpl  $12, -4(%ebp)
        jg    .L6
        movl  -8(%ebp), %edx
        movl  -4(%ebp), %eax
        leal  (%edx,%eax), %eax
        addl  -12(%ebp), %eax
        movl  %eax, -4(%ebp)
.L6:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%ebp)
.L2:
        cmpl  $7, -4(%ebp)
        jle   .L3
        cmpl  $12, -4(%ebp)
        jg    .L6
        movl  -8(%ebp), %edx
        movl  -4(%ebp), %eax
        leal  (%edx,%eax), %eax
        addl  -12(%ebp), %eax
        movl  %eax, -4(%ebp)
.L6:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%ebp)
.L2:
        cmpl  $7, -4(%ebp)
        jle   .L3
        cmpl  $12, -4(%ebp)
        jg    .L6
        movl  -8(%ebp), %edx
        movl  -4(%ebp), %eax
        leal  (%edx,%eax), %eax
        addl  -12(%ebp), %eax
        movl  %eax, -4(%ebp)
.L6:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%ebp)
.L2:
        cmpl  $7, -4(%ebp)
        jle   .L3
        cmpl  $12, -4(%ebp)
        jg    .L6
        movl  -8(%ebp), %edx
        movl  -4(%ebp), %eax
        leal  (%edx,%eax), %eax
        addl  -12(%ebp), %eax
        movl  %eax, -4(%ebp)
.L6:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

Outline

- Example 1
  - Constant Propagation
  - Copy Propagation
  - Dead Code Elimination
  - Loop unrolling
- Example 2
  - Partial Redundancy Elimination
  - Copy Propagation
  - Dead Code Elimination

Example Program 1

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?

Example Program 1

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?
- 12

Example Program 1

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?
- 12
- We use this program to illustrate various shades of the following optimizations:
  Constant propagation, Copy propagation, Loop unrolling, Dead code elimination

Compilation Command

```
$gcc -fdump-tree-all -fdump-rtl-all -O2 ccp.c
```

Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

B2

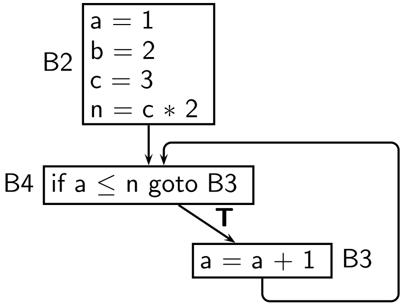
$$\boxed{\begin{array}{l} a = 1 \\ b = 2 \\ c = 3 \\ n = c * 2 \end{array}}$$

Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

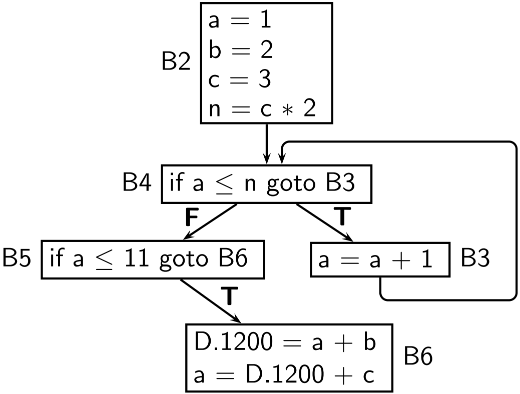


Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

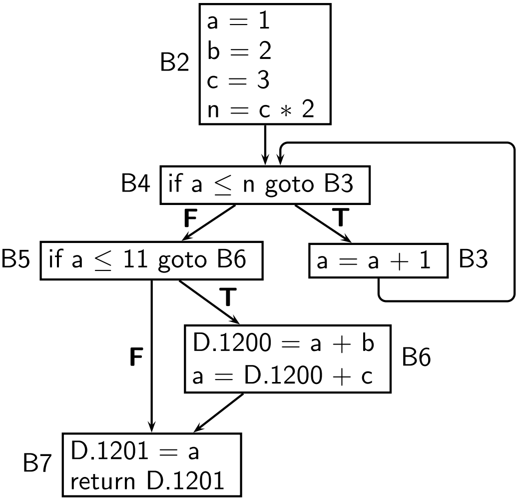


Example Program 1

Program ccp.c

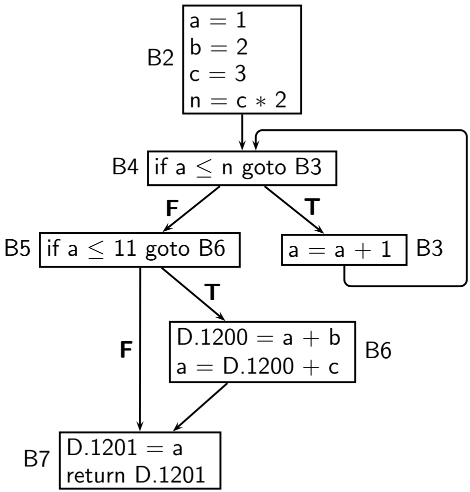
```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph



Control Flow Graph: Pictorial and Textual View

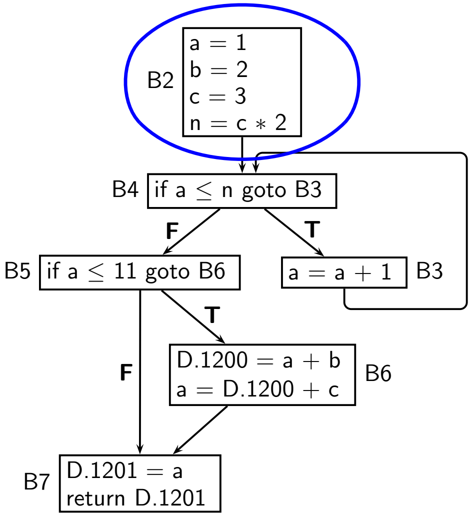
Control flow graph



Dump file ccp.c.012t.cfg

Control Flow Graph: Pictorial and Textual View

Control flow graph

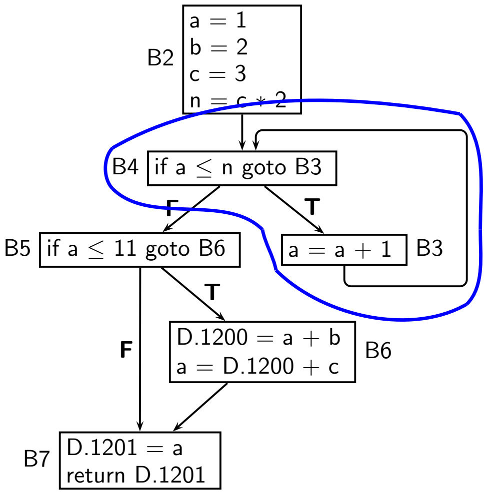


Dump file ccp.c.012t.cfg

```
<bb 2>:
a = 1;
b = 2;
c = 3;
n = c * 2;
goto <bb 4>;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph

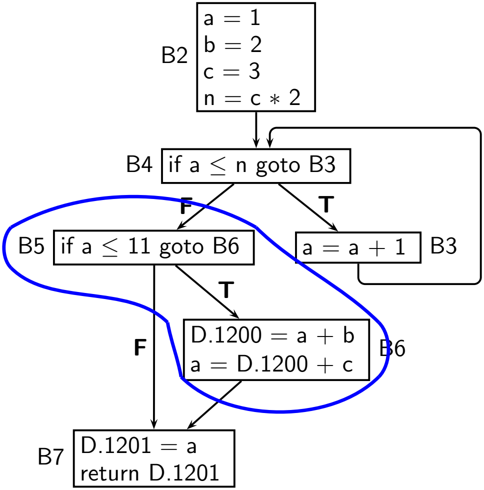


Dump file ccp.c.012t.cfg

```
<bb 3>:
a = a + 1;

<bb 4>:
if (a <= n)
   goto <bb 3>;
else
   goto <bb 5>;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph

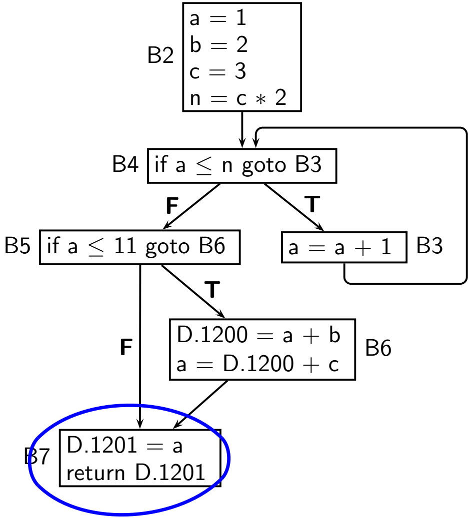


Dump file ccp.c.012t.cfg

```
<bb 5>:
if (a <= 11)
    goto <bb 6>;
else
    goto <bb 7>;

<bb 6>:
D.1200 = a + b;
a = D.1200 + c;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph



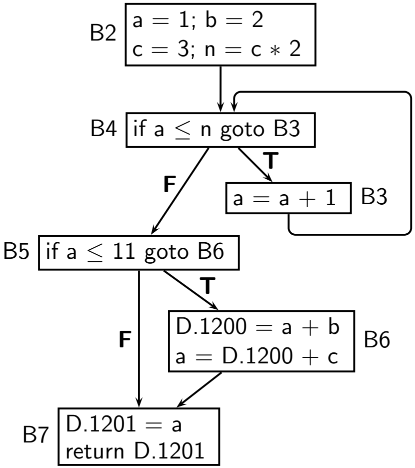
Dump file ccp.c.012t.cfg

```
<bb 7>:
D.1201 = a;
return D.1201;
```

Single Static Assignment (SSA) Form

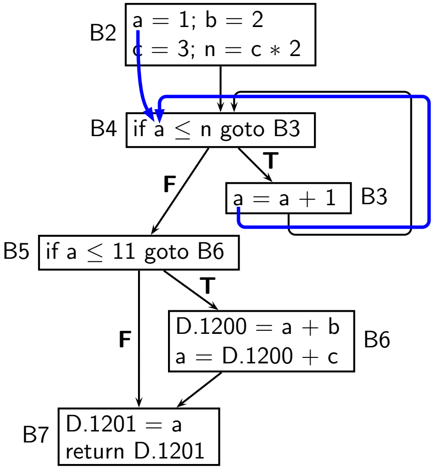
Control flow graph

SSA Form



Single Static Assignment (SSA) Form

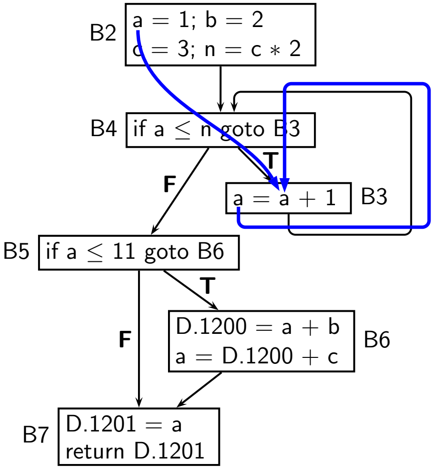
Control flow graph



SSA Form

Single Static Assignment (SSA) Form

Control flow graph

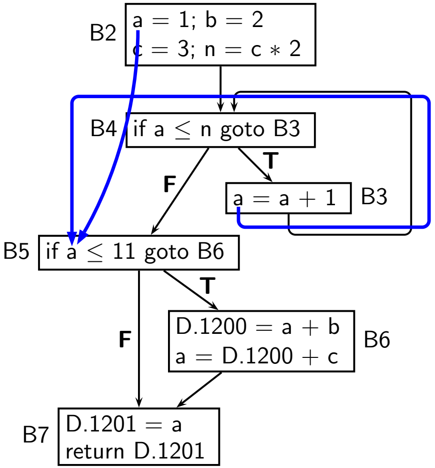


SSA Form

Single Static Assignment (SSA) Form

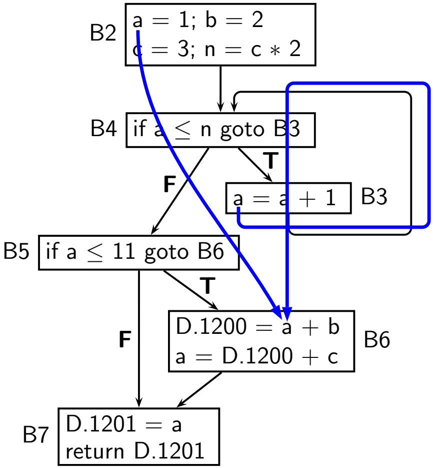
Control flow graph

SSA Form



Single Static Assignment (SSA) Form

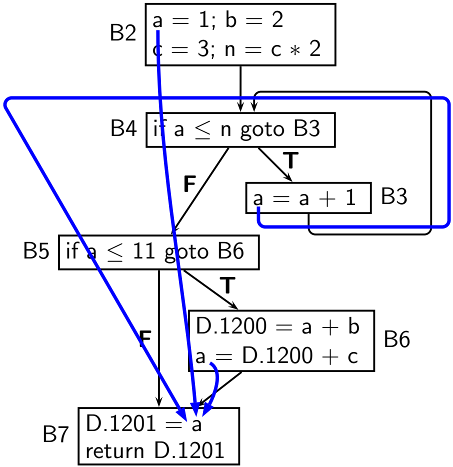
Control flow graph



SSA Form

Single Static Assignment (SSA) Form

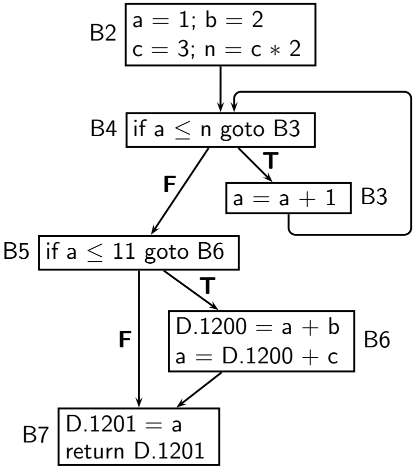
Control flow graph



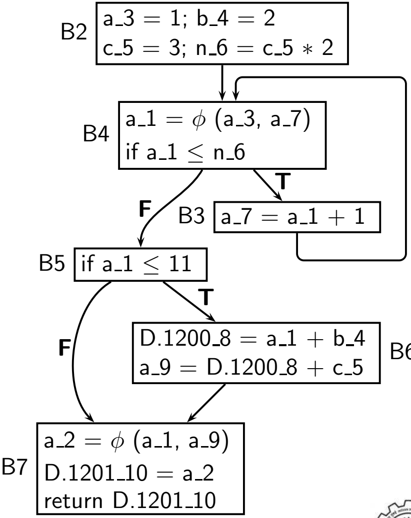
SSA Form

Single Static Assignment (SSA) Form

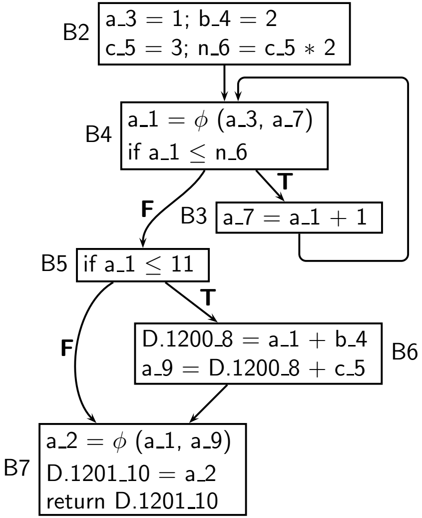
Control flow graph



SSA Form



Properties of SSA Form

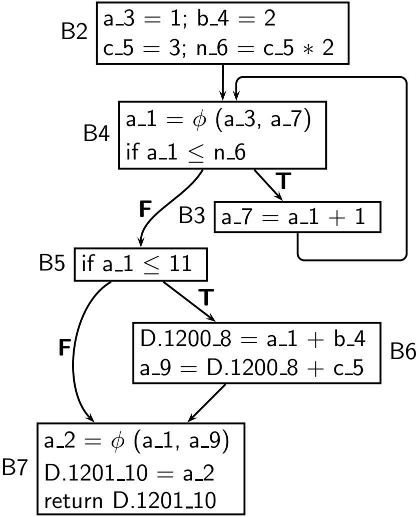


- A ϕ function is a multiplexer or a selection function
- Every use of a variable corresponds to a unique definition of the variable
- For every use, the definition is guaranteed to appear on every path leading to the use

SSA construction algorithm is expected to insert as few ϕ functions as possible to ensure the above properties

SSA Form: Pictorial and Textual View

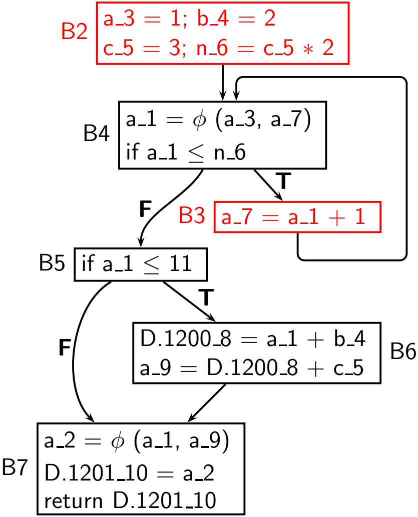
CFG in SSA form



Dump file ccp.c.023t.ssa

SSA Form: Pictorial and Textual View

CFG in SSA form

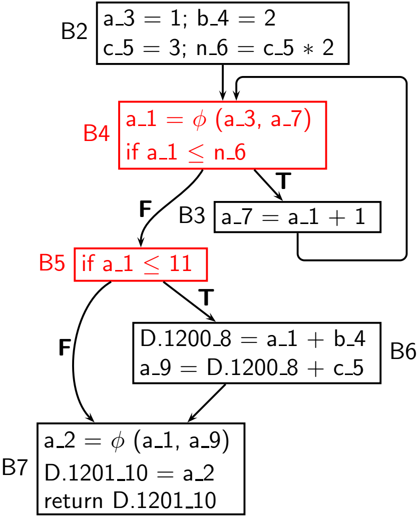


Dump file ccp.c.023t.ssa

```
  <bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;
```

SSA Form: Pictorial and Textual View

CFG in SSA form

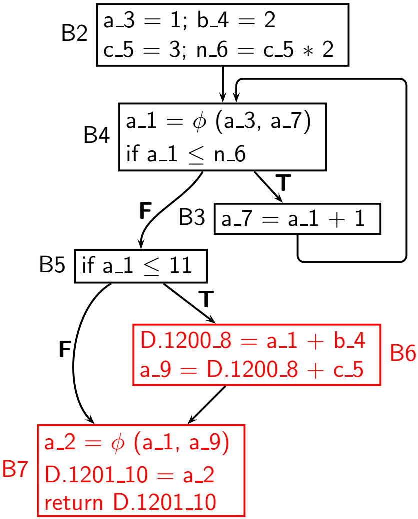


Dump file ccp.c.023t.ssa

```
<bb 4>:
  # a_1 = PHI <a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;

<bb 5>:
  if (a_1 <= 11)
    goto <bb 6>;
  else
    goto <bb 7>;
```

SSA Form: Pictorial and Textual View

CFG in SSA form



Dump file ccp.c.023t.ssa

```
<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```

A Comparison of CFG and SSA Dumps

Dump file ccp.c.012t.cfg

Dump file ccp.c.023t.ssa

A Comparison of CFG and SSA Dumps

Dump file ccp.c.012t.cfg

```
<bb 2>:
  a = 1;
  b = 2;
  c = 3;
  n = c * 2;
  goto <bb 4>;

<bb 3>:
  a = a + 1;
```

Dump file ccp.c.023t.ssa

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;
```

A Comparison of CFG and SSA Dumps

Dump file ccp.c.012t.cfg

```
<bb 4>:
  if (a <= n)
     goto <bb 3>;
  else
     goto <bb 5>;

<bb 5>:
  if (a <= 11)
      goto <bb 6>;
  else
      goto <bb 7>;
```

Dump file ccp.c.023t.ssa

```
<bb 4>:
  # a_1 = PHI <a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;

<bb 5>:
  if (a_1 <= 11)
    goto <bb 6>;
  else
    goto <bb 7>;
```

A Comparison of CFG and SSA Dumps

Dump file ccp.c.012t.cfg

```
<bb 6>:
D.1200 = a + b;
a = D.1200 + c;

<bb 7>:
D.1201 = a;
return D.1201;
```

Dump file ccp.c.023t.ssa

```
<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```

Copy Renamimg

Input dump: ccp.c.023t.ssa

```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```

Output dump: ccp.c.026t.copyrename1

```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  a_10 = a_2;
  return a_10;
```

First Level Constant and Copy Propagation

Input dump: ccp.c.026t.copyrename1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;

<bb 4>:
  # a_1 = PHI < a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;
```

Output dump: ccp.c.027t.cpp1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;

<bb 4>:
  # a_1 = PHI < 1(2), a_7(3)>
  if (a_1 <= 6)
    goto <bb 3>;
  else
    goto <bb 5>;
```

First Level Constant and Copy Propagation

Input dump: ccp.c.026t.copyrename1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

...

<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;
```

Output dump: ccp.c.027t.cpp1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

...

<bb 6>:
  D.1200_8 = a_1 + 2;
  a_9 = D.1200_8 + 3;
```

Second Level Copy Propagation

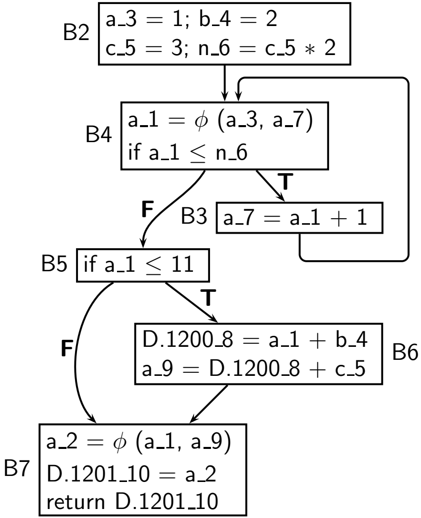
Input dump: ccp.c.029t.ccp1

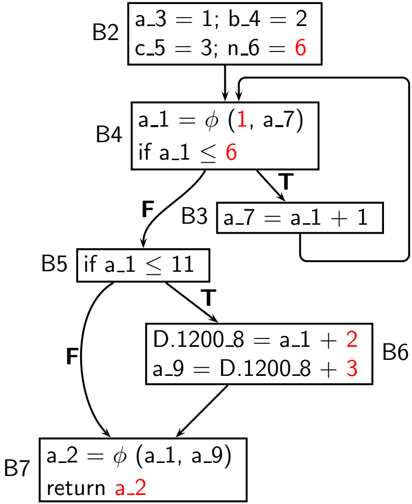
```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  a_10 = a_2;
  return a_10;
```

Output dump: ccp.c.031t.copyprop1

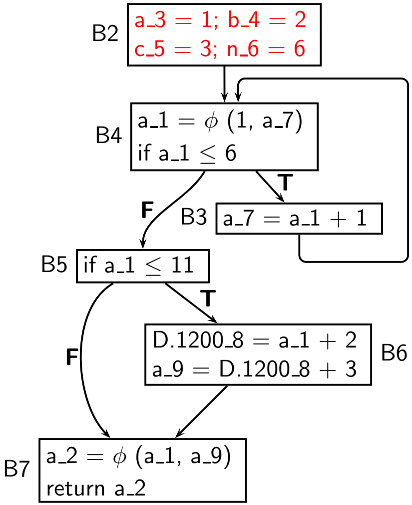
```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  return a_2;
```

The Result of Copy Propagation and Renaming



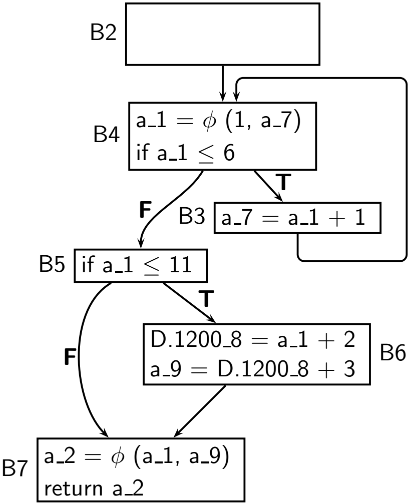


The Result of Copy Propagation and Renaming



- No uses for variables a_3, b_4, c_5, and n_6
- Assignments to these variables can be deleted

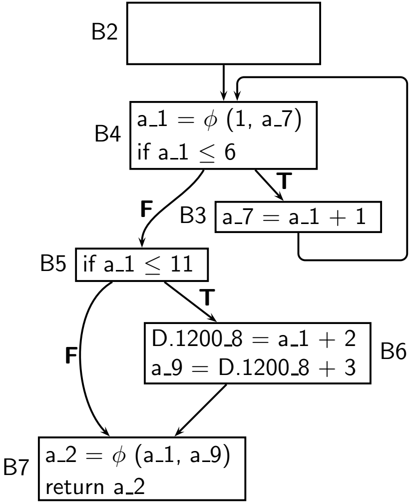
Dead Code Elimination Using Control Dependence



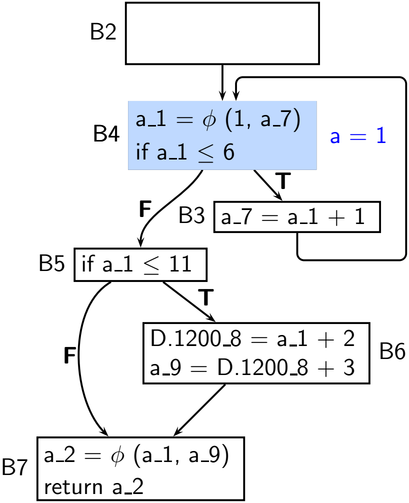
Dump file ccp.c.033t.cddce1

```
<bb 2>:
  goto <bb 4>;
<bb 3>:
  a_7 = a_1 + 1;
<bb 4>:
  # a_1 = PHI <1(2), a_7(3)>
  if (a_1 <= 6) goto <bb 3>;
  else goto <bb 5>;
<bb 5>:
  if (a_1 <= 11) goto <bb 6>;
  else goto <bb 7>;
<bb 6>:
  D.1200_8 = a_1 + 2;
  a_9 = D.1200_8 + 3;
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  return a_2;
```

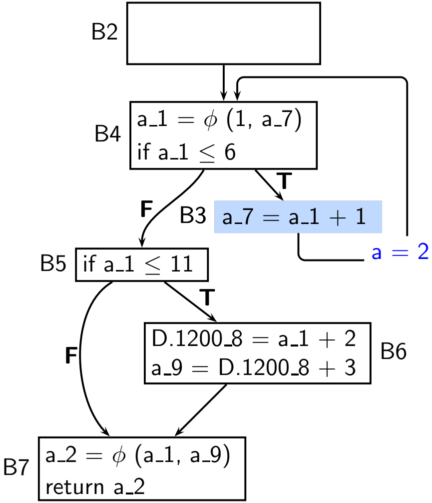
Loop Unrolling



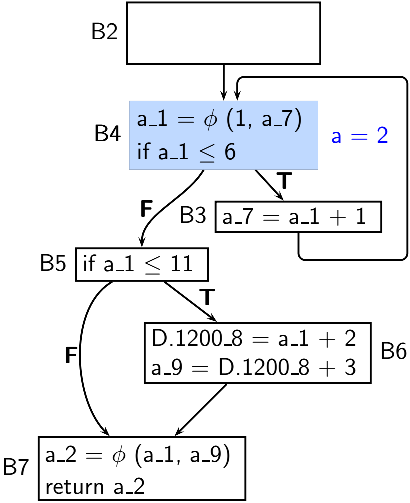
Loop Unrolling



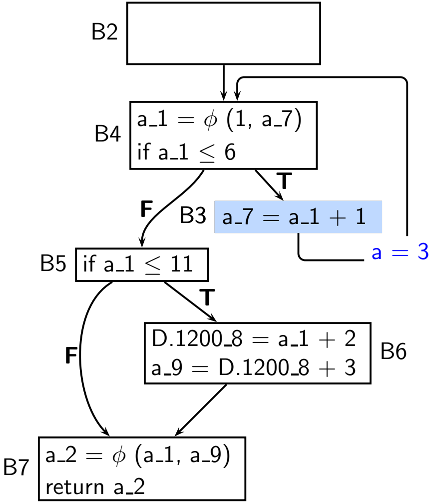
Loop Unrolling



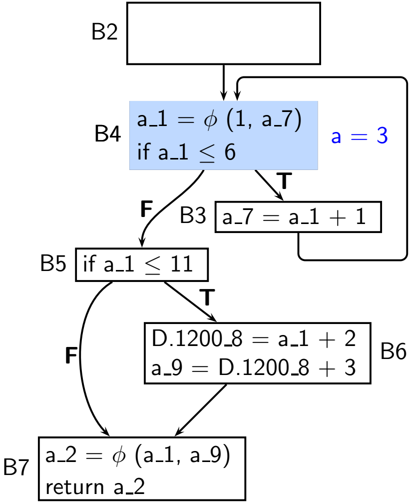
Loop Unrolling



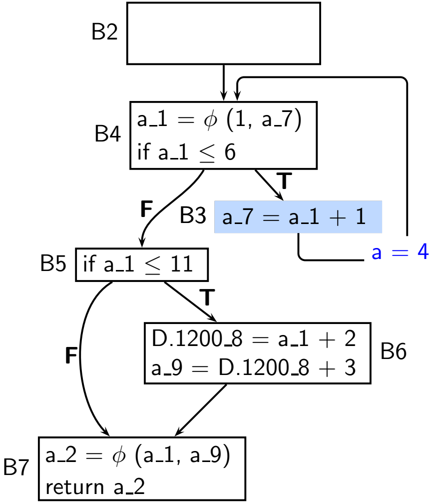
Loop Unrolling



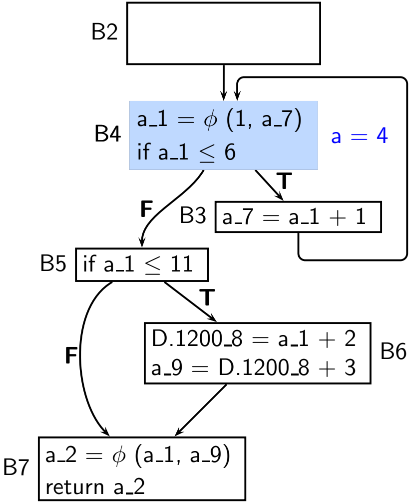
Loop Unrolling



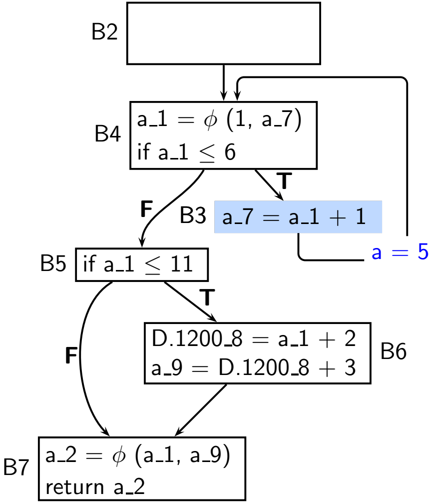
Loop Unrolling



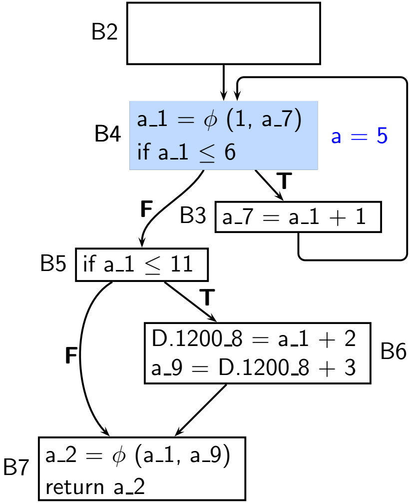
Loop Unrolling



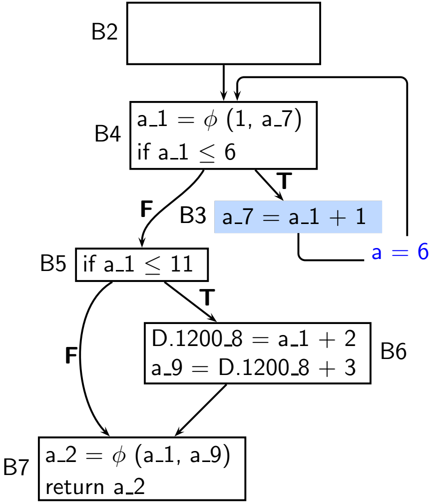
Loop Unrolling



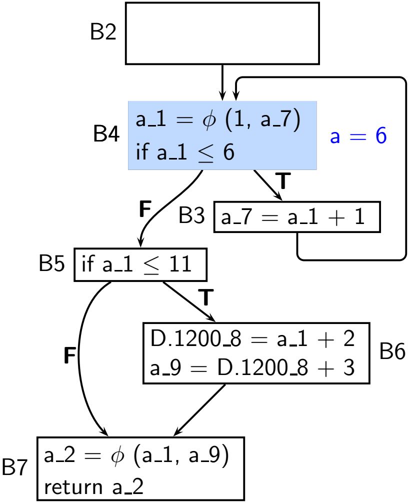
Loop Unrolling



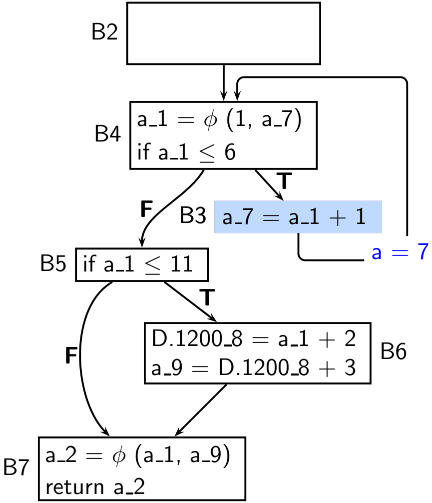
Loop Unrolling



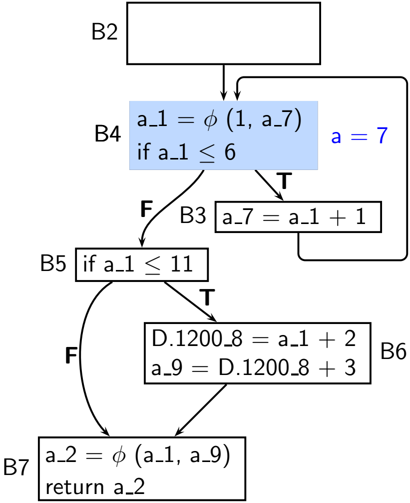
Loop Unrolling



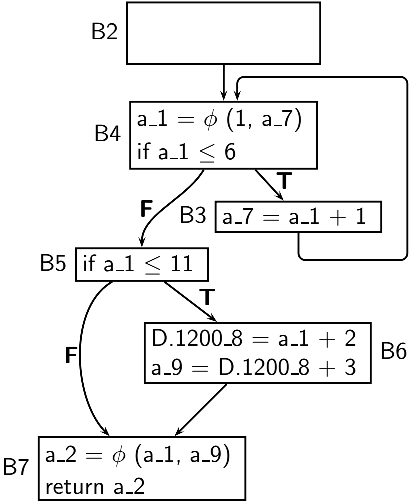
Loop Unrolling

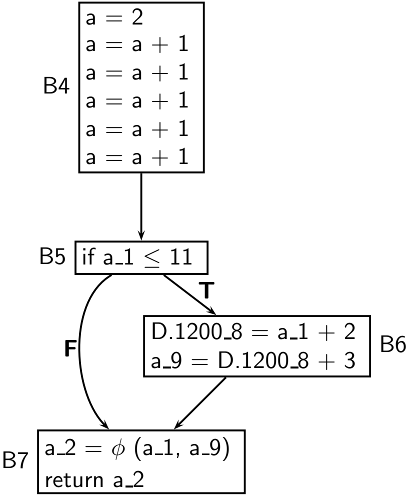


Loop Unrolling



Loop Unrolling

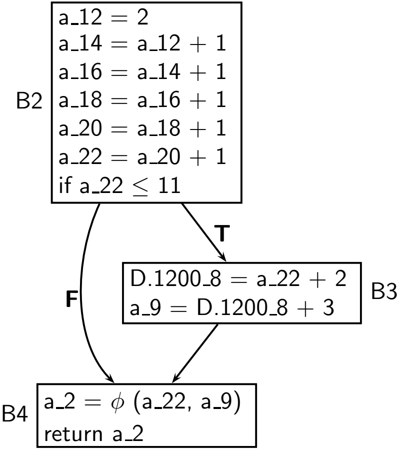




Complete Unrolling of Inner Loops

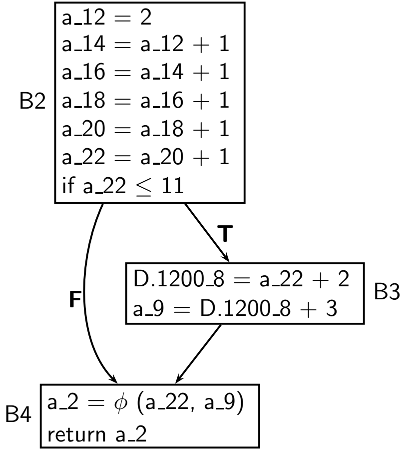
Dump file: ccp.c.058t.cunrolli

```
<bb 2>:
  a_12 = 2;
  a_14 = a_12 + 1;
  a_16 = a_14 + 1;
  a_18 = a_16 + 1;
  a_20 = a_18 + 1;
  a_22 = a_20 + 1;
  if (a_22 <= 11) goto <bb 3>;
  else goto <bb 4>;

<bb 3>:
  D.1959_8 = a_22 + 2;
  a_9 = D.1959_8 + 3;

<bb 4>:
  # a_2 = PHI <a_22(2), a_9(3)>
  return a_2;
```



Another Round of Constant Propagation

Input



Dump file: ccp.c.059t.ccp2

```
<bb 2>:
  a_22 = 7;
  a_9 = 12;
  return 12;
```

Dead Code Elimination Using Copy Propagation

Dump file: ccp.c.059t.ccp2

```
a_22 = 7;
a_9 = 12;
return 12;
```

Dump file: ccp.c.066t.copyprop2

```
<bb 2>:
  return 12;
```

Example Program 2

```
int f(int b, int c, int n)
{ int a;

   do
   {
      a = b+c;
   }
   while (a <= n);

   return a;
}
```

We use this program to illustrate the following optimizations:

Partial Redundancy Elimination, Copy Propagation, Dead Code Elimination

Compilation Command

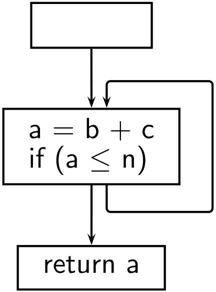
```
$gcc -fdump-tree-all -fdump-rtl-all -O2 -S ccp.c
```

Example Program 2

loop.c

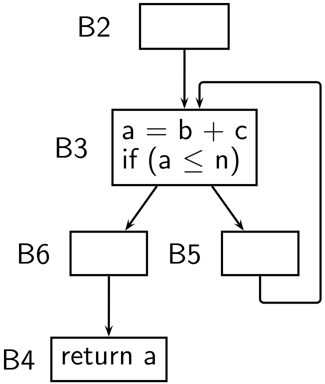
```
int f(int b, int c, int n)
{ int a;

   do
   {
      a = b+c;
   }
   while (a <= n);

   return a;
}
```

Control Flow Graph



Dump of Input to PRE Pass

Control Flow Graph



loop.c.091t.crited

```
<bb 2>:

<bb 3>:
  a_3 = c_2(D) + b_1(D);
  if (a_3 <= n_4(D)) goto <bb 5>;
  else goto <bb 6>;

<bb 5>:
  goto <bb 3>;

<bb 6>:

<bb 4>:
  # a_6 = PHI <a_3(6)>
  return a_6;
```

Input and Output of PRE Pass

loop.c.091t.crited

```
<bb 2>:

<bb 3>:
  a_3 = c_2(D) + b_1(D);
  if (a_3 <= n_4(D))
       goto <bb 5>;
  else goto <bb 6>;

<bb 5>:
  goto <bb 3>;

<bb 6>:
<bb 4>:
  # a_6 = PHI <a_3(6)>
  return a_6;
```

loop.c.092t.pre

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  a_3 = pretmp.2_7;
  if (a_3 <= n_4(D))
       goto <bb 5>;
  else goto <bb 6>;

<bb 5>:
  goto <bb 3>;

<bb 6>:
<bb 4>:
  # a_6 = PHI <a_3(6)>
  return a_6;
```

Copy Propagation after PRE

loop.c.092t.pre

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  a_3 = pretmp.2_7;
  if ( a_3 <= n_4(D))
       goto <bb 5>;
  else goto <bb 6>;

<bb 5>:
  goto <bb 3>;

<bb 6>:
<bb 4>:
  # a_6 = PHI <a_3(6)>
  return a_6;
```

loop.c.096t.copyprop4

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  a_3 = pretmp.2_7;
  if ( n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 5>;

<bb 4>:
  goto <bb 3>;

<bb 5>:
  # a_8 = PHI <pretmp.2_7(3)>
  return a_8;
```

Dead Code Elimination

loop.c.096t.copyprop4

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  a_3 = pretmp.2_7;
  if (n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 5>;

<bb 4>:
  goto <bb 3>;

<bb 5>:
  # a_8 = PHI <pretmp.2_7(3)>
  return a_8;
```

loop.c.097t.dceloop1

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  if (n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 5>;

<bb 4>:
  goto <bb 3>;

<bb 5>:
  # a_8 = PHI <pretmp.2_7(3)>
  return a_8;
```

Redundant ϕ Function Elimination and Copy Propagation

loop.c.097t.dceloop1

```
<bb 2>:
  pretmp.2_7 = b_1(D) + c_2(D);

<bb 3>:
  if (n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 5>;

<bb 4>:
  goto <bb 3>;

<bb 5>:
  # a_8 = PHI <pretmp.2_7(3)>
  return a_8;
```

loop.c.124t.phicprop2

```
<bb 2>:
  pretmp.2_7 = c_2(D) + b_1(D);
  if (n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 3>;

<bb 3>:
  return pretmp.2_7;

<bb 4>:
  goto <bb 4>;
```

Final Assembly Program

loop.c.124t.phicprop2

```
<bb 2>:
  pretmp.2_7 = c_2(D) + b_1(D);
  if (n_4(D) >= pretmp.2_7)
    goto <bb 4>;
  else
    goto <bb 3>;

<bb 3>:
  return pretmp.2_7;

<bb 4>:
  goto <bb 4>;
```

loop.s

```
        pushl   %ebp
        movl    %esp, %ebp
        movl    12(%ebp), %eax
        addl    8(%ebp), %eax
        cmpl    %eax, 16(%ebp)
        jge     .L2
        popl    %ebp
        ret
.L2:
.L3:
        jmp     .L3
```

Why infinite loop?

Infinite Loop in Example Program 2

```
int f(int b, int c, int n)
{ int a;

   do
   {
      a = b+c;
   }
   while (a <= n);

   return a;
}
```

The program does not terminate unless a $>$ n

Part 5

Configuration and Building

Configuration and Building: Outline

- Code Organization of GCC
- Configuration and Building
- Native build Vs. cross build
- Testing GCC

GCC Code Organization

Logical parts are:

- Build configuration files
- Front end + generic + generator sources
- Back end specifications
- Emulation libraries
  (eg. `libgcc` to emulate operations not supported on the target)
- Language Libraries (except C)
- Support software (e.g. garbage collector)

GCC Code Organization

Front End Code

- Source language dir: `$(SOURCE_D)/<lang dir>`
- Source language dir contains
  - Parsing code (Hand written)
  - Additional AST/Generic nodes, if any
  - Interface to Generic creation

Except for C – which is the "native" language of the compiler

C front end code in: `$(SOURCE_D)/gcc`

Optimizer Code and Back End Generator Code

- Source language dir: `$(SOURCE_D)/gcc`

Back End Specification

- $(SOURCE_D)/gcc/config/<target dir>/
  Directory containing back end code
- Two main files: <target>.h and <target>.md,
  e.g. for an i386 target, we have
  $(SOURCE_D)/gcc/config/i386/i386.md and
  $(SOURCE_D)/gcc/config/i386/i386.h
- Usually, also <target>.c for additional processing code
  (e.g. $(SOURCE_D)/gcc/config/i386/i386.c)
- Some additional files

Configuration

Preparing the GCC source for local adaptation:

- The platform on which it will be compiled
- The platform on which the generated compiler will execute
- The platform for which the generated compiler will generate code
- The directory in which the source exists
- The directory in which the compiler will be generated
- The directory in which the generated compiler will be installed
- The input languages which will be supported
- The libraries that are required
- etc.

Pre-requisites for Configuring and Building GCC 4.5.0

- ISO C90 Compiler / GCC 2.95 or later
- GNU bash: for running configure etc
- Awk: creating some of the generated source file for GCC
- bzip/gzip/untar etc. For unzipping the downloaded source file
- GNU make version 3.8 (or later)
- GNU Multiple Precision Library (GMP) version 4.3.2
- MPFR Library version 3.0.0 (or later)
  (multiple precision floating point with correct rounding)
- MPC Library version 0.8.2 (or later)
- Parma Polyhedra Library (PPL) version 0.10
- CLooG-PPL (Chunky Loop Generator) version 0.15.9
- jar, or InfoZIP (zip and unzip)
- libelf version 0.8.12 (or later) (for LTO)

Sequence of Build for GCC 4.5.0

- Configuration Options
  - GMP(4.3.2)

    ```
    CPPFLAGS=-fexceptions ./configure --enable-cxx
                                   --prefix=/usr/local
    ```
  - MPFR(3.0.0), MPC(0.8.2) and PPL(0.10.2)

    ```
    ./configure --prefix=/usr/local
    ```
  - CLooG-PPL(0.15.9)

    ```
    ./configure --with-ppl=/usr/local
    ```
- Building all of them (in the order given above)

```
make
make check
sudo make install
sudo ldconfig
```

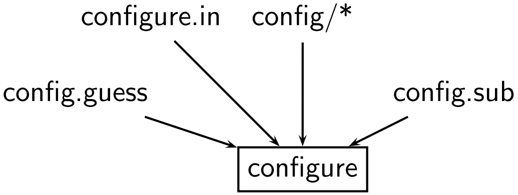
Our Conventions for Directory Names

- GCC source directory : `$(SOURCE_D)`
- GCC build directory : `$(BUILD)`
- GCC install directory : `$(INSTALL)`
- Important
  - `$(SOURCE_D)` $\neq$ `$(BUILD)` $\neq$ `$(INSTALL)`
  - None of the above directories should be contained in any of the above directories

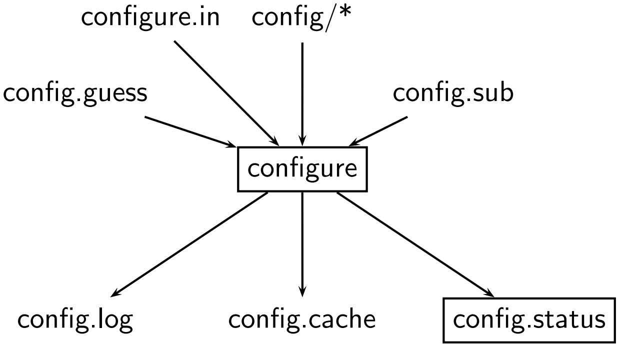
Configuring GCC

$\boxed{\text{configure}}$

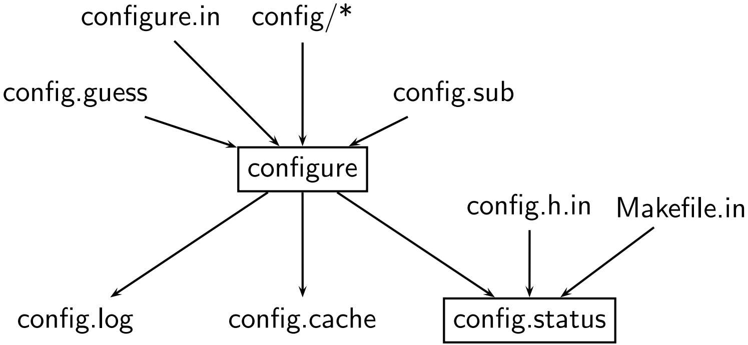
Configuring GCC



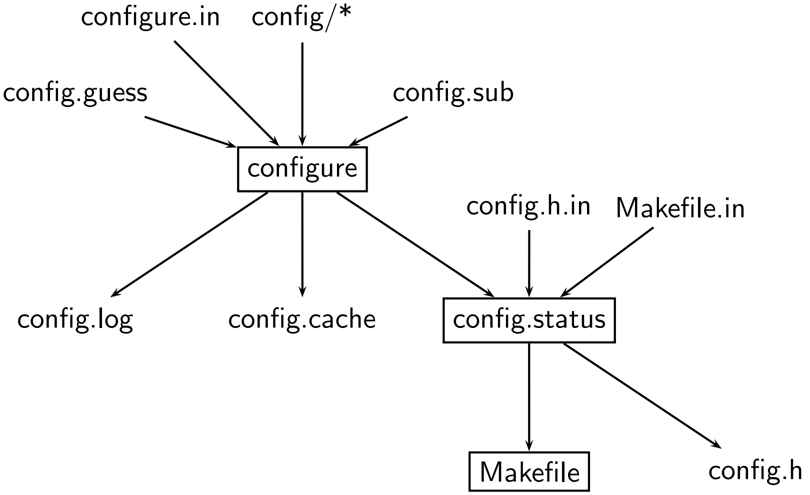
Configuring GCC



Configuring GCC



Configuring GCC



Steps in Configuration and Building

Usual Steps	
- Download and untar the source `cd $(SOURCE_D)` `./configure` `make` `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source • `cd $(SOURCE_D)` • `./configure` • `make` • `make install`	• Download and untar the source • `cd $(BUILD)` • `$(SOURCE_D)/configure` • `make` • `make install`

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $(SOURCE_D)`	• `cd $(BUILD)`
• `./configure`	• `$(SOURCE_D)/configure`
• `make`	• `make`
• `make install`	• `make install`

GCC generates a large part of source code during a build!

Building a Compiler: Terminology

- The sources of a compiler are compiled (i.e. built) on *Build system*, denoted BS.
- The built compiler runs on the *Host system*, denoted HS.
- The compiler compiles code for the *Target system*, denoted TS.

The built compiler itself runs on HS and generates executables that run on TS.

Variants of Compiler Builds

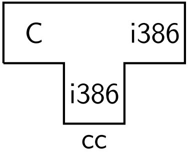
BS = HS = TS	Native Build
BS = HS $\neq$ TS	Cross Build
BS $\neq$ HS $\neq$ TS	Canadian Cross

Example

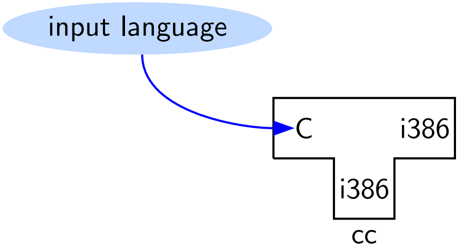
Native i386: built on i386, hosted on i386, produces i386 code.

Sparc cross on i386: built on i386, hosted on i386, produces Sparc code.

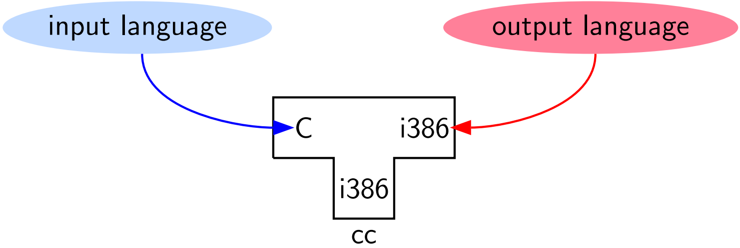
T Notation for a Compiler



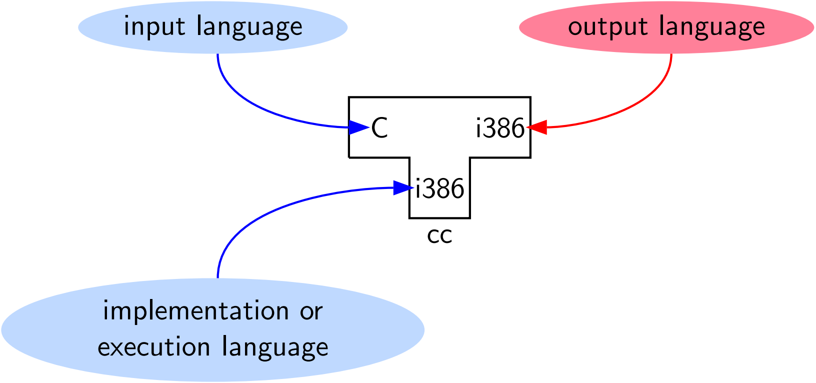
T Notation for a Compiler



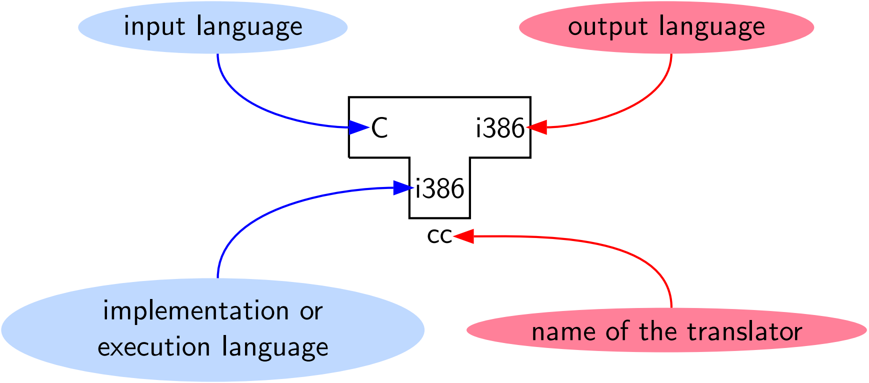
T Notation for a Compiler



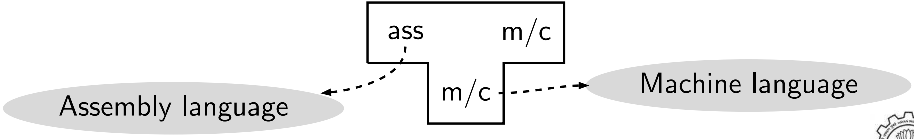
T Notation for a Compiler



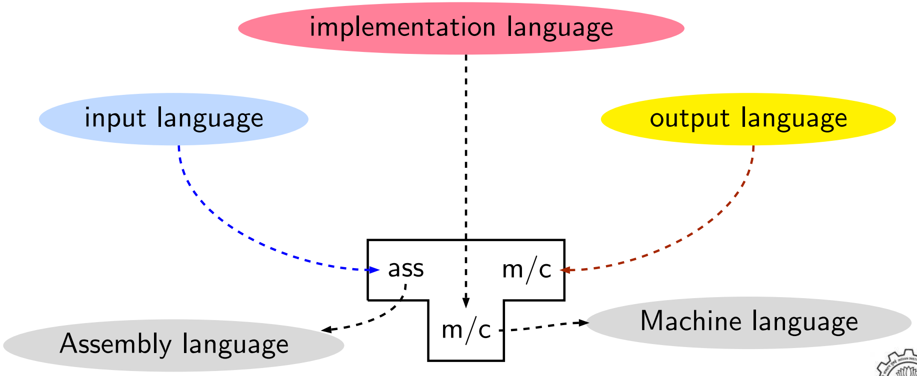
T Notation for a Compiler



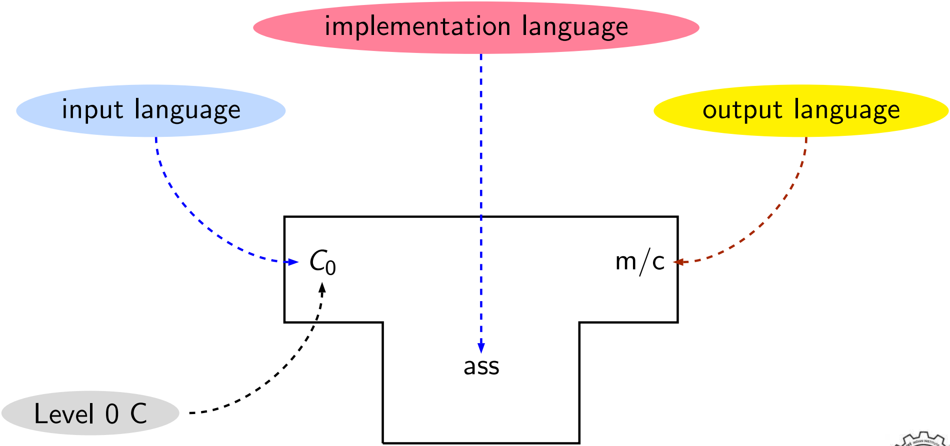
Bootstrapping: The Conventional View



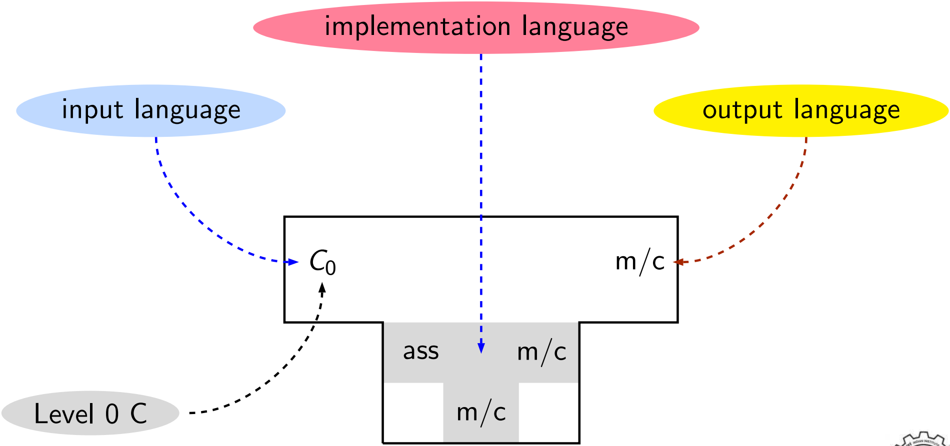
Bootstrapping: The Conventional View



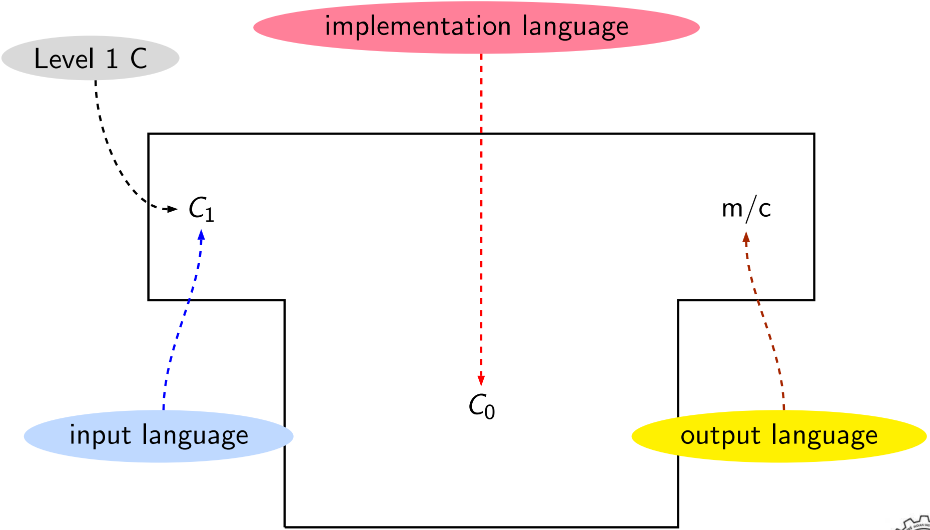
Bootstrapping: The Conventional View



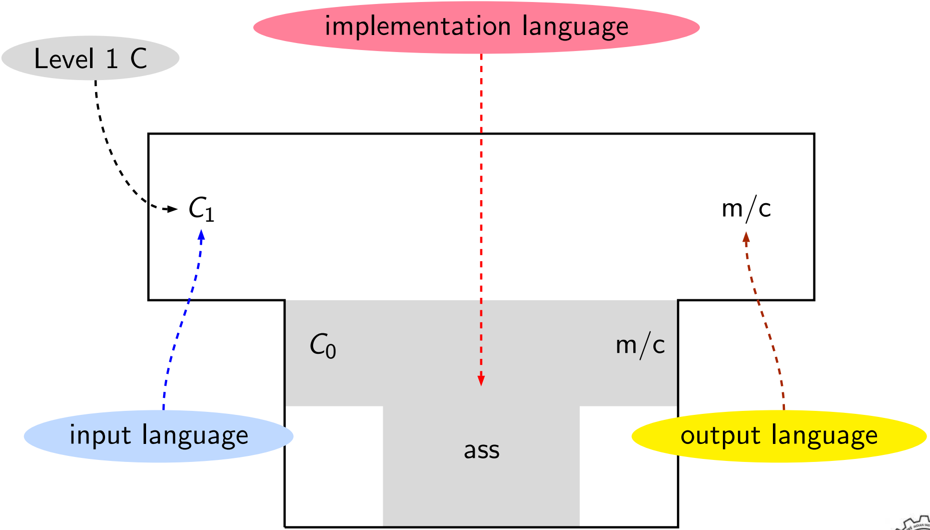
Bootstrapping: The Conventional View



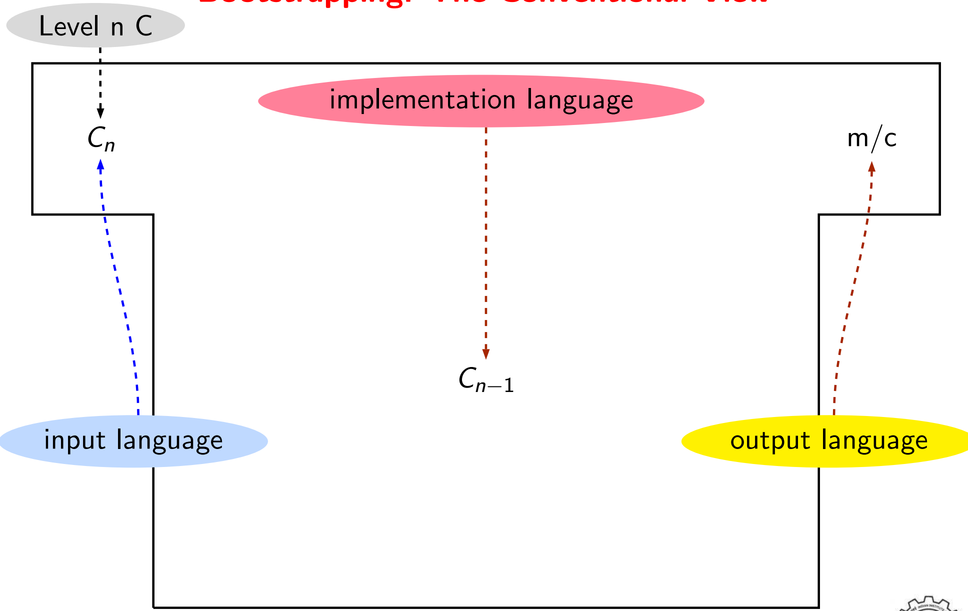
Bootstrapping: The Conventional View



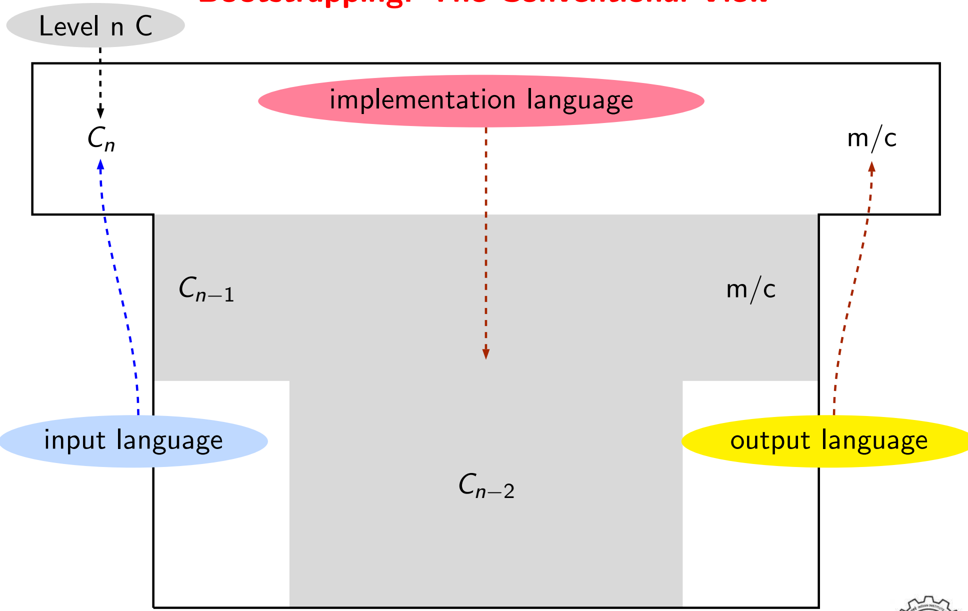
Bootstrapping: The Conventional View



Bootstrapping: The Conventional View



Bootstrapping: The Conventional View



Bootstrapping: GCC View

- Language need not change, but the compiler may change
  Compiler is improved, bugs are fixed and newer versions are released
- To build a new version of a compiler given a built old version:
  - Stage 1: Build the new compiler using the old compiler
  - Stage 2: Build another new compiler using compiler from stage 1
  - Stage 3: Build another new compiler using compiler from stage 2
    Stage 2 and stage 3 builds must result in identical compilers

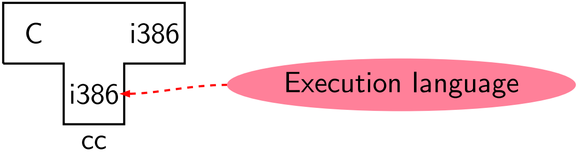
$\Rightarrow$ Building cross compilers stops after Stage 1!

A Native Build on i386

GCC
Source

Requirement: BS = HS = TS = i386

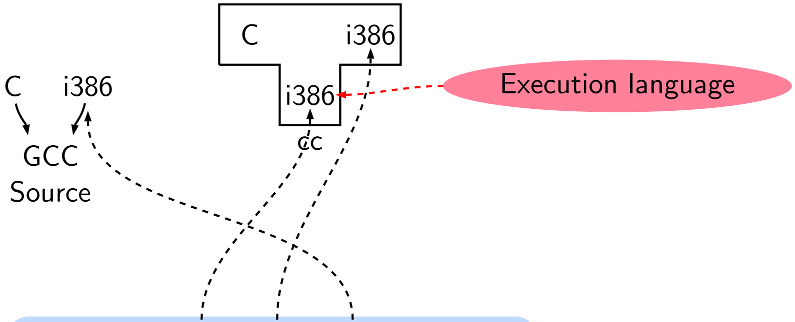
A Native Build on i386



GCC
Source

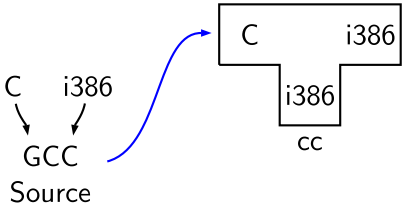
Requirement: BS = HS = TS = i386

A Native Build on i386



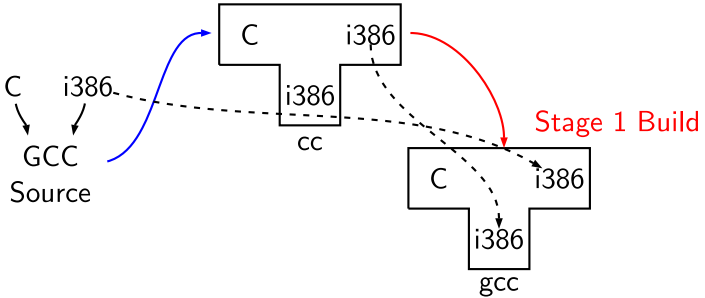
Requirement: BS = HS = TS = i386

A Native Build on i386



Requirement: BS = HS = TS = i386

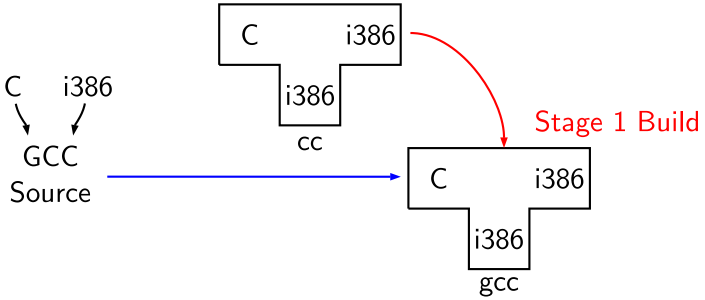
A Native Build on i386



Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc

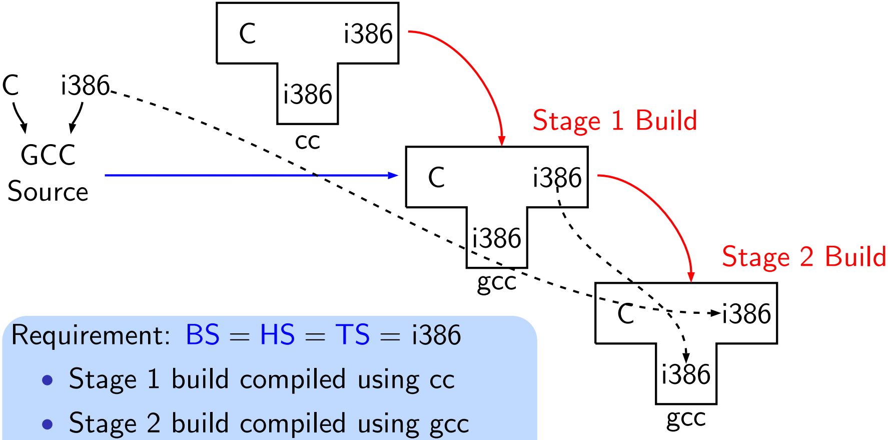
A Native Build on i386



Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc

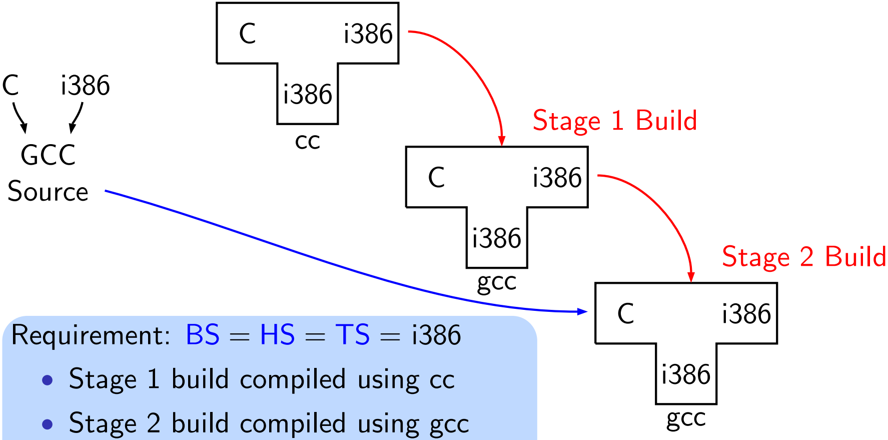
A Native Build on i386



Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc

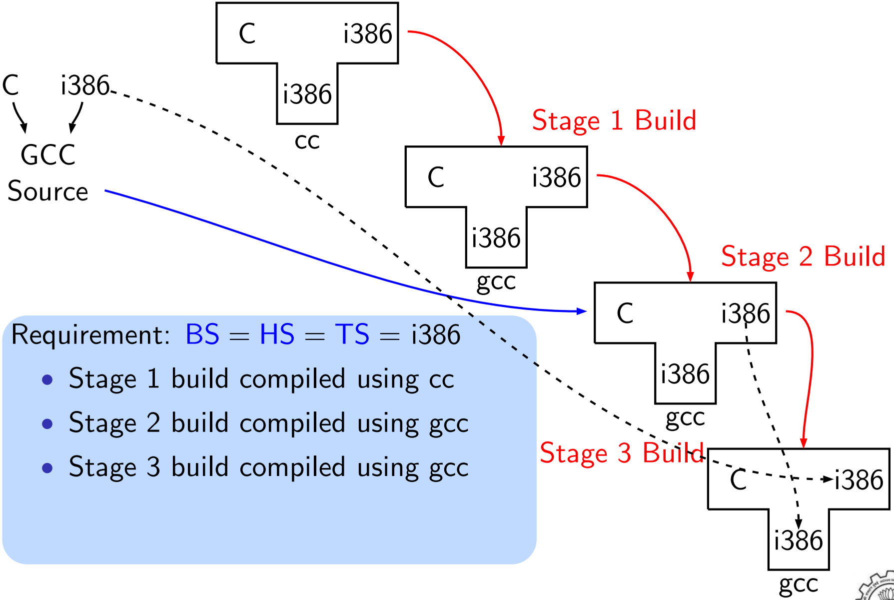
A Native Build on i386



Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc

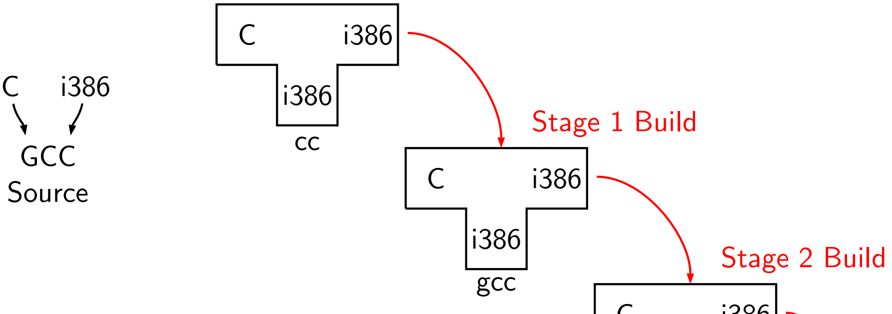
A Native Build on i386

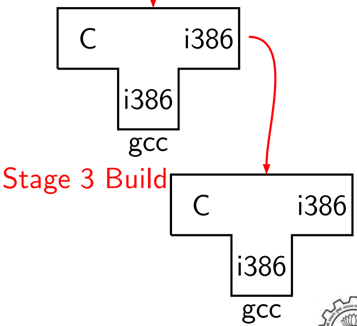


Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
- Stage 3 build compiled using gcc

A Native Build on i386





Requirement: BS = HS = TS = i386

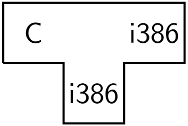
- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
- Stage 3 build compiled using gcc
- Stage 2 and Stage 3 Builds must be identical for a successful native build

A Cross Build on i386

GCC
Source

Requirement: BS = HS = i386, TS = mips

A Cross Build on i386

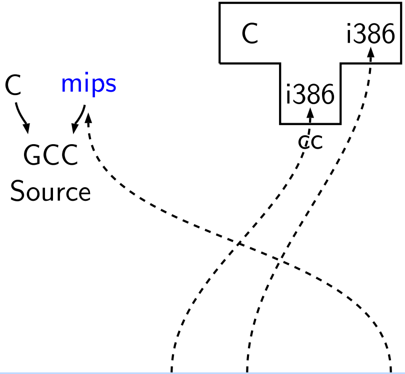


cc

GCC
Source

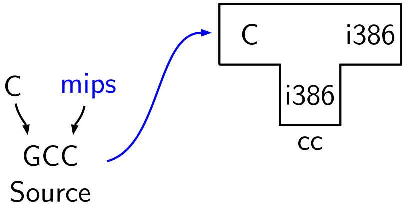
Requirement: BS = HS = i386, TS = mips

A Cross Build on i386



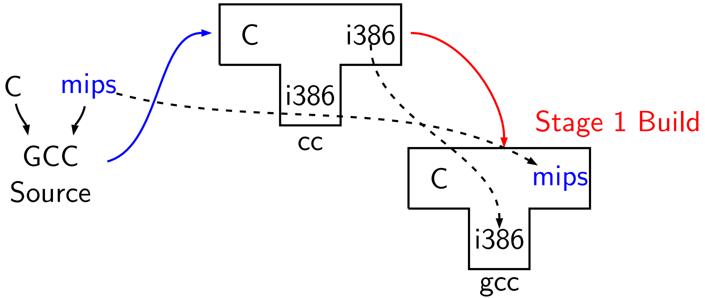
Requirement: BS = HS = i386, TS = mips

A Cross Build on i386



Requirement: BS = HS = i386, TS = mips

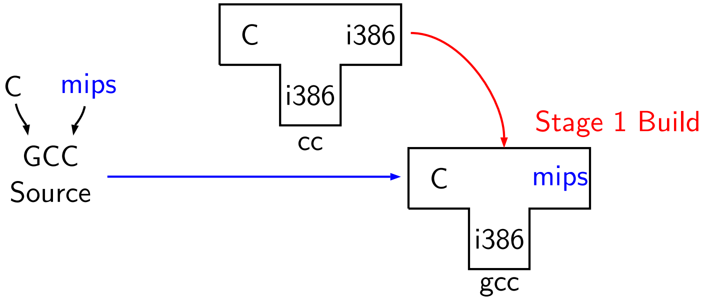
A Cross Build on i386



Requirement: BS = HS = i386, TS = mips

- Stage 1 build compiled using cc

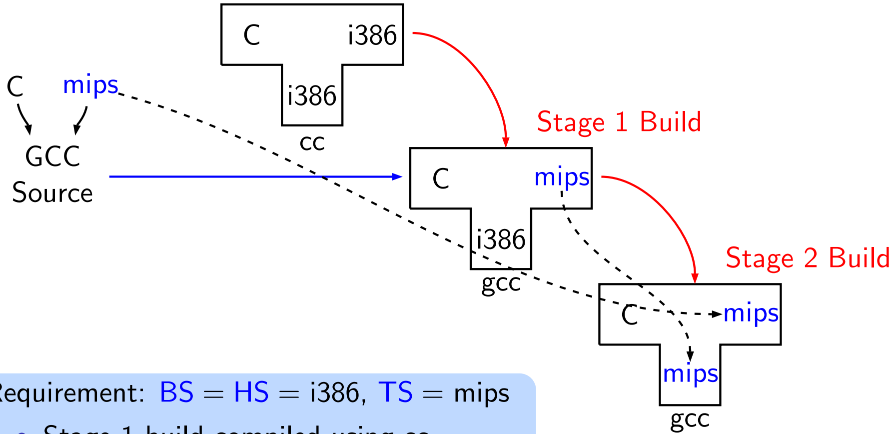
A Cross Build on i386



Requirement: BS = HS = i386, TS = mips

- Stage 1 build compiled using cc

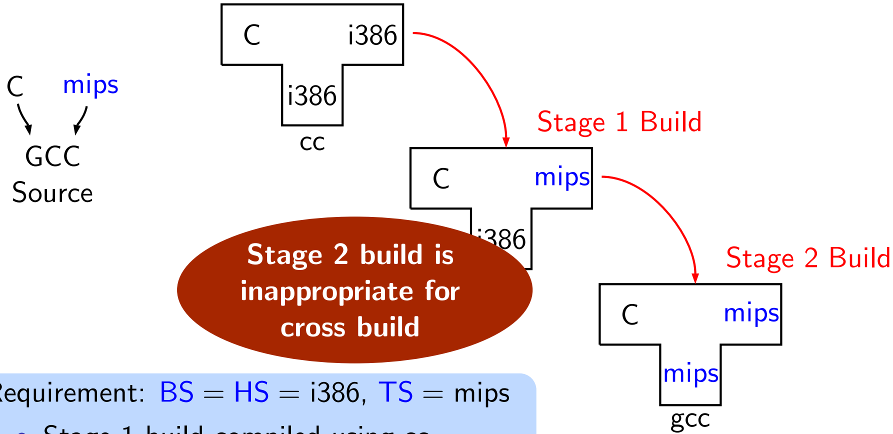
A Cross Build on i386



Requirement: BS = HS = i386, TS = mips

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
  Its HS = mips and not i386!

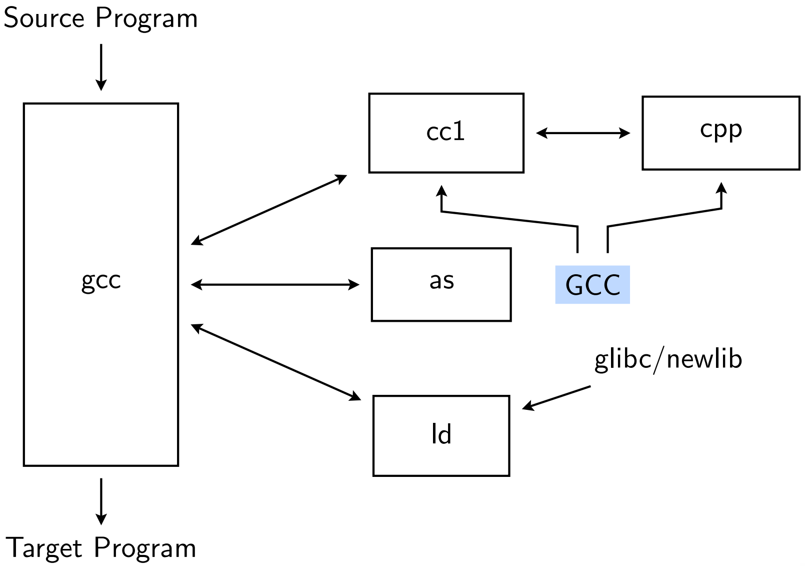
A Cross Build on i386



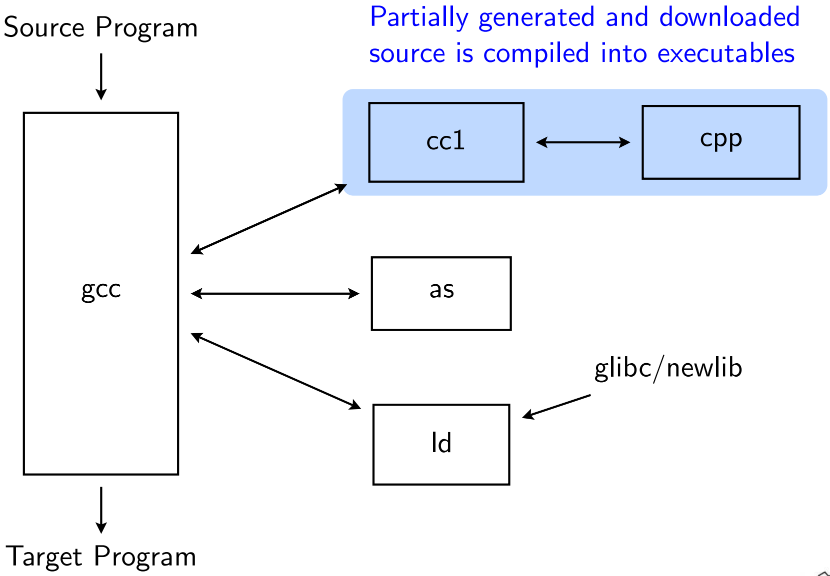
Requirement: BS = HS = i386, TS = mips

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
  Its HS = mips and not i386!

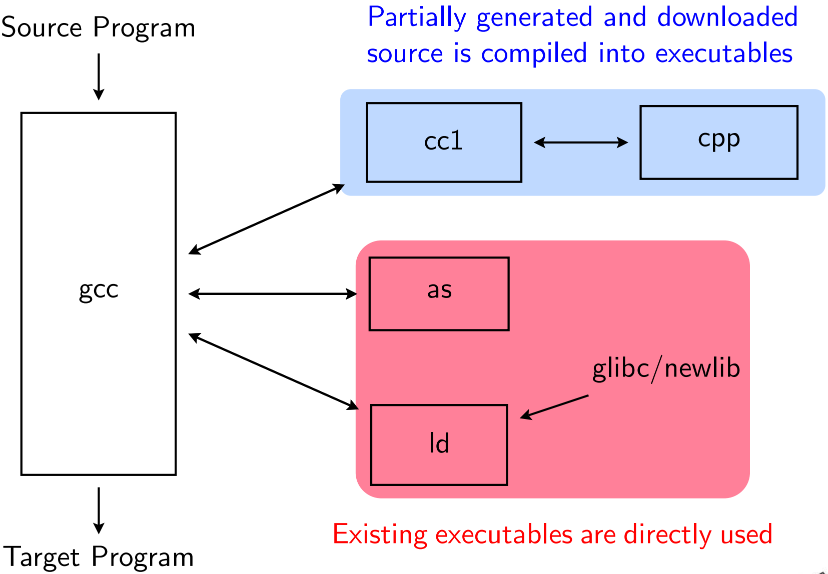
A More Detailed Look at Building



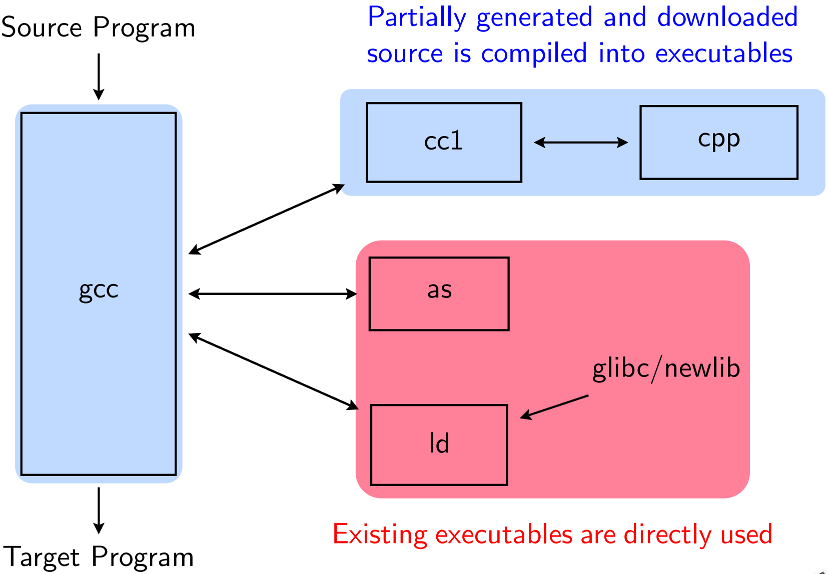
A More Detailed Look at Building



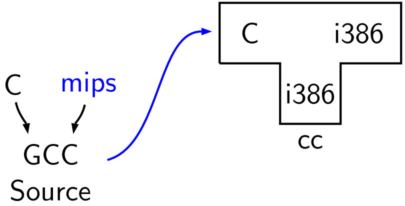
A More Detailed Look at Building



A More Detailed Look at Building



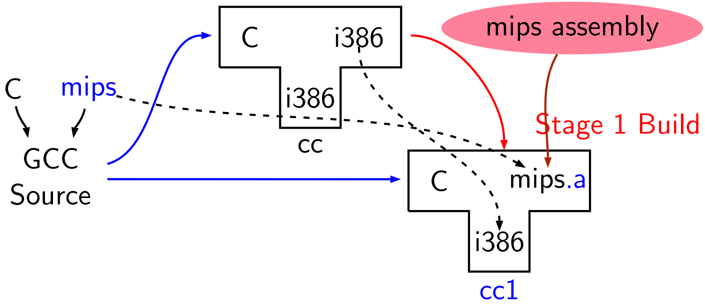
A More Detailed Look at Cross Build



Requirement: BS = HS = i386, TS = mips

we have not built binutils for mips

A More Detailed Look at Cross Build

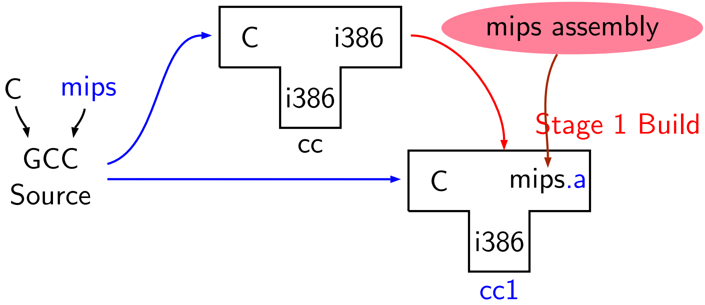


Requirement: BS = HS = i386, TS = mips

- *Stage 1 cannot build gcc but can build only cc1*

we have not built binutils for mips

A More Detailed Look at Cross Build

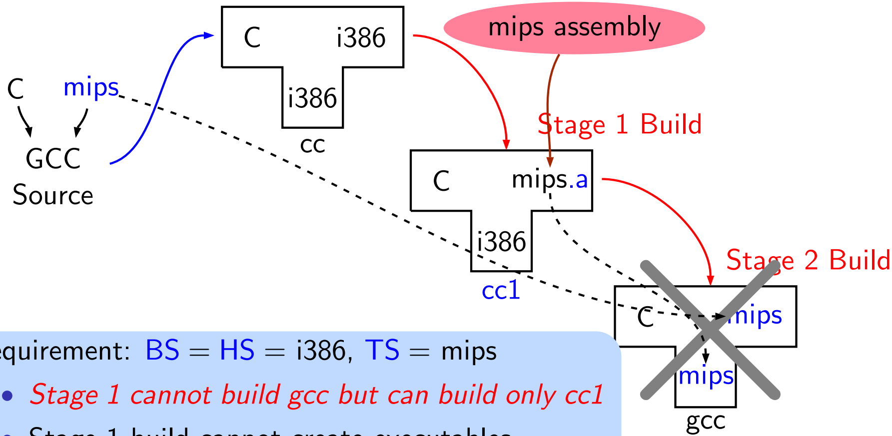


Requirement: BS = HS = i386, TS = mips

- *Stage 1 cannot build gcc but can build only cc1*
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build

we have not built binutils for mips

A More Detailed Look at Cross Build

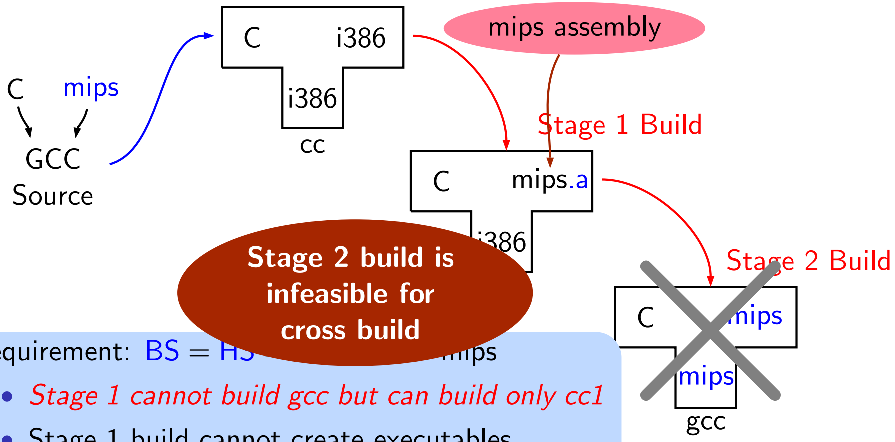


Requirement: BS = HS = i386, TS = mips

- *Stage 1 cannot build gcc but can build only cc1*
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build
- Stage 2 build is not possible

we have
not built binutils
for mips

A More Detailed Look at Cross Build



Stage 2 build is infeasible for cross build

Requirement: BS = HS … mips

- *Stage 1 cannot build gcc but can build only cc1*
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build
- Stage 2 build is not possible

we have not built binutils for mips

Cross Build Revisited

- Option 1: Build binutils in the same source tree as `gcc`
  Copy binutils source in `$(SOURCE_D)`, configure and build stage 1
- Option 2:
  - Compile cross-assembler (`as`), cross-linker (`ld`), cross-archiver (`ar`), and cross-program to build symbol table in archiver (`ranlib`),
  - Copy them in `$(INSTALL)/bin`
  - Build stage GCC
  - Install newlib
  - Reconfigure and build GCC
    Some options differ in the two builds

Commands for Configuring and Building GCC

This is what *we* specify

- `cd $(BUILD)`

Commands for Configuring and Building GCC

This is what *we* specify

- `cd $(BUILD)`
- `$(SOURCE_D)/configure <options>`
  `configure` output: customized `Makefile`

Commands for Configuring and Building GCC

This is what *we* specify

- `cd $(BUILD)`
- `$(SOURCE_D)/configure <options>`
  `configure` output: customized `Makefile`
- `make 2> make.err > make.log`

Commands for Configuring and Building GCC

This is what *we* specify

- `cd $(BUILD)`
- `$(SOURCE_D)/configure <options>`
  `configure` output: customized `Makefile`
- `make 2> make.err > make.log`
- `make install 2> install.err > install.log`

Build for a Given Machine

This is what actually happens!

- Generation
    - Generator sources (`$(SOURCE_D)/gcc/gen*.c`) are read and generator executables are created in `$(BUILD)/gcc/build`
    - MD files are read by the generator executables and back end source code is generated in `$(BUILD)/gcc`
- Compilation
  Other source files are read from `$(SOURCE_D)` and executables created in corresponding subdirectories of `$(BUILD)`
- Installation
  Created executables and libraries are copied in `$(INSTALL)`

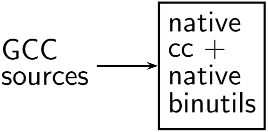
Build for a Given Machine

This is what actually happens!

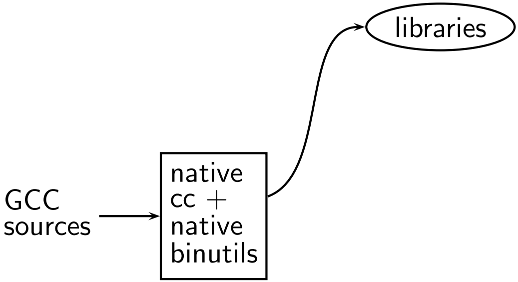
- Generation
  - Generator sources (`$(SOURCE_D)/gcc/gen*.c`) are read and generator executables are created in `$(BUILD)/gcc/build`
  - MD files are read by the generator executables and back end source code is generated in `$(BUILD)/gcc`
- Compilation
  Other source files are read from `$(SOURCE_D)` and executables created in corresponding subdirectories of `$(BUILD)`
- Installation
  Created executables and libraries are copied in `$(INSTALL)`

```
genattr
gencheck
genconditions
genconstants
genflags
genopinit
genpreds
genattrtab
genchecksum
gencondmd
genemit
gengenrtl
genmddeps
genoutput
genrecog
genautomata
gencodes
genconfig
genextract
gengtype
genmodes
genpeep
```

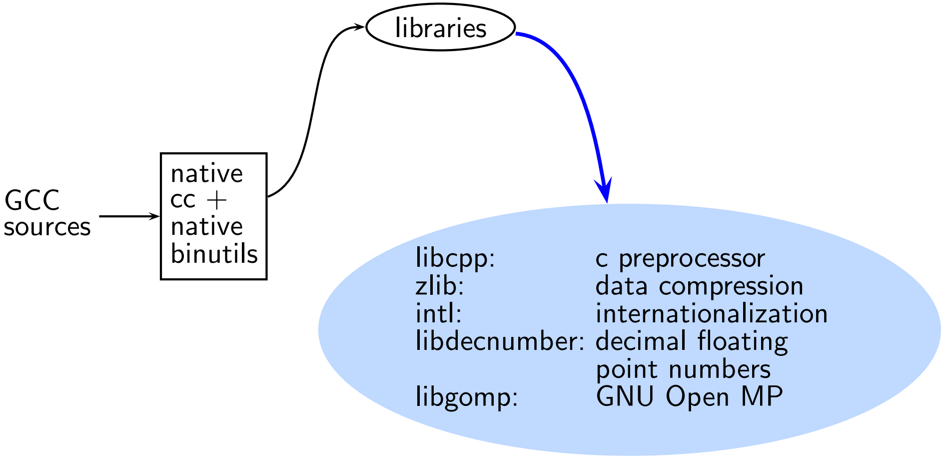
More Details of an Actual Stage 1 Build for C



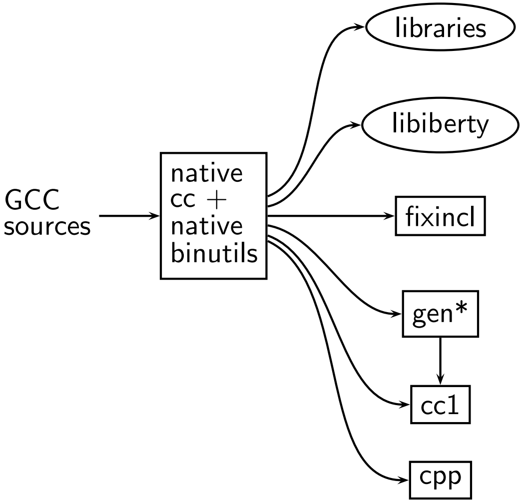
More Details of an Actual Stage 1 Build for C



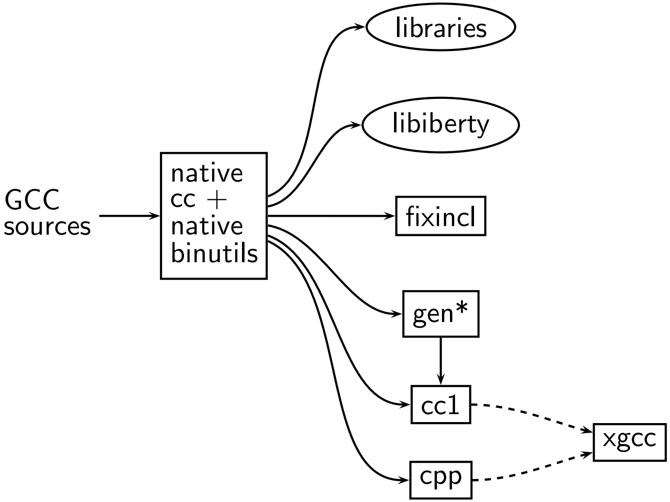
More Details of an Actual Stage 1 Build for C



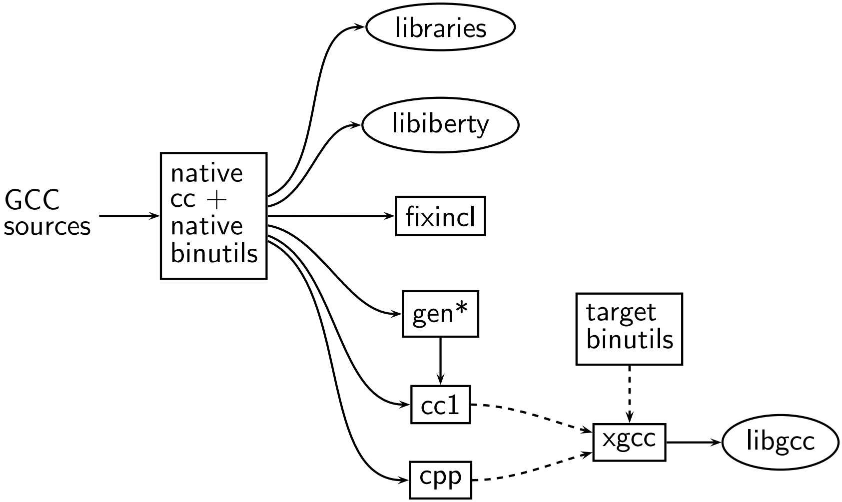
More Details of an Actual Stage 1 Build for C



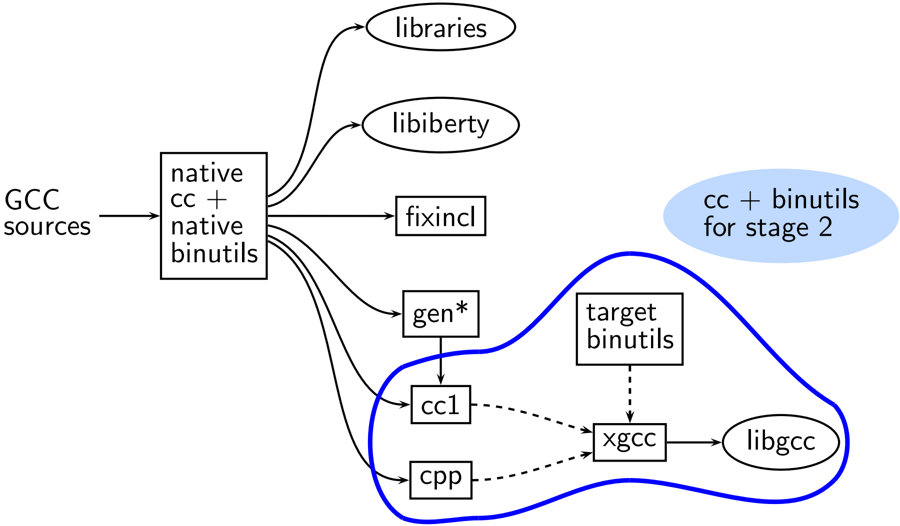
More Details of an Actual Stage 1 Build for C



More Details of an Actual Stage 1 Build for C



More Details of an Actual Stage 1 Build for C



Build Failures due to Machine Descriptions

Incomplete MD specifications	$\Rightarrow$	Unsuccessful build
Incorrect MD specification	$\Rightarrow$	Successful build but run time failures/crashes (either ICE or `SIGSEGV`)

Building cc1 Only

- Add a new target in the Makefile.in

```
cc1:
    make all-gcc TARGET-gcc=cc1$(exeext)
```

- Configure and build with the command make cc1.

Common Configuration Options

`--target`

- Necessary for cross build
- Possible `host-cpu-vendor` strings: Listed in `$(SOURCE_D)/config.sub`

`--enable-languages`

- Comma separated list of language names
- Default names: `c`, `c++`, `fortran`, `java`, `objc`
- Additional names possible: `ada`, `obj-c++`, `treelang`

`--prefix=$(INSTALL)`

`--program-prefix`

- Prefix string for executable names

`--disable-bootstrap`

- Build stage 1 only

Configuring and Building GCC – Summary

- Choose the source language: C (`--enable-languages=c`)
- Choose installation directory: (`--prefix=<absolute path>`)
- Choose the target for non native builds: (`--target=sparc-sunos-sun`)
- Run: `configure` with above choices
- Run: `make` to
  - generate target specific part of the compiler
  - build the entire compiler
- Run: `make install` to install the compiler

Tip

Redirect <u>all</u> the outputs:

```
$ make > make.log 2> make.err
```

Part 6

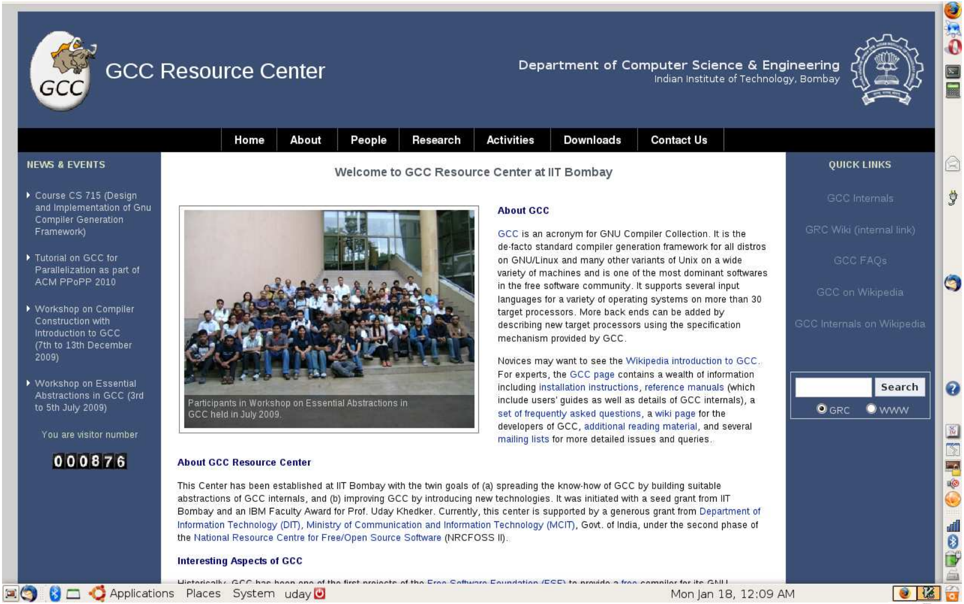
GCC Resource Center

National Resource Center for F/OSS, Phase II

GCC Resource Center is a part of NRCFOSS (II)

- Sponsored by Department of Information Technology (DIT), Ministry of Information and Communication Technology
- CDAC Chennai is the coordinating agency of NRCFOSS (II)
- Participating agencies
  CDAC Chennai, CDAC Mumbai, CDAC Hyderabad, IIT Bombay, IIT Madras, Anna University,
- Project investigators of GCC Resource Center

Uday Khedker:	Professor, Dept. of CSE, IIT Bombay
Supratim Biswas:	Professor, Dept. of CSE, IIT Bombay
Amitabha Sanyal:	Professor, Dept. of CSE, IIT Bombay


GCC
GCC Resource Center
Department of Computer Science & Engineering
Indian Institute of Technology, Bombay
Home
About
People
Research
Activities
Downloads
Contact Us
NEWS & EVENTS
Course CS 715 (Design and Implementation of Gnu Compiler Generation Framework)
Tutorial on GCC for Parallelization as part of ACM PPoPP 2010
Workshop on Compiler Construction with Introduction to GCC (7th to 13th December 2009)
Workshop on Essential Abstractions in GCC (3rd to 5th July 2009)
You are visitor number
000876
Welcome to GCC Resource Center at IIT Bombay
Participants in Workshop on Essential Abstractions in GCC held in July 2009.
About GCC
GCC is an acronym for GNU Compiler Collection. It is the de-facto standard compiler generation framework for all distros on GNU/Linux and many other variants of Unix on a wide variety of machines and is one of the most dominant softwares in the free software community. It supports several input languages for a variety of operating systems on more than 30 target processors. More back ends can be added by describing new target processors using the specification mechanism provided by GCC.
Novices may want to see the Wikipedia introduction to GCC. For experts, the GCC page contains a wealth of information including installation instructions, reference manuals (which include users' guides as well as details of GCC internals), a set of frequently asked questions, a wiki page for the developers of GCC, additional reading material, and several mailing lists for more detailed issues and queries.
About GCC Resource Center
This Center has been established at IIT Bombay with the twin goals of (a) spreading the know-how of GCC by building suitable abstractions of GCC internals, and (b) improving GCC by introducing new technologies. It was initiated with a seed grant from IIT Bombay and an IBM Faculty Award for Prof. Uday Khedker. Currently, this center is supported by a generous grant from Department of Information Technology (DIT), Ministry of Communication and Information Technology (MCIT), Govt. of India, under the second phase of the National Resource Centre for Free/Open Source Software (NRCFOSS II).
Interesting Aspects of GCC
QUICK LINKS
GCC Internals
GRC Wiki (internal link)
GCC FAQs
GCC on Wikipedia
GCC Internals on Wikipedia
Search
GRC
WWW
Applications
Places
System
uday
Mon Jan 18, 12:09 AM

Objectives of GCC Resource Center

1. To support the open source movement
   Providing training and technical know-how of the GCC framework to academia and industry.
2. To include better technologies in GCC
   Whole program optimization, Optimizer generation, Tree tiling based instruction selection.
3. To facilitate easier and better quality deployments/enhancements of GCC
   Restructuring GCC and devising methodologies for systematic construction of machine descriptions in GCC.
4. To bridge the gap between academic research and practical implementation
   Designing suitable abstractions of GCC architecture

Broad Research Goals of GCC Resource Center

- Using GCC as a means
  - Adding new optimizations to GCC
  - Adding flow and context sensitive analyses to GCC (In particular, pointer analysis)
  - Translation validation of GCC
  - Linear types in GCC
- Using GCC as an end in itself
  - Changing the retargetability mechanism of GCC
  - Cleaning up the machine descriptions of GCC
  - Systematic construction of machine descriptions
  - Facilitating optimizer generation in GCC

GRC Training Programs

Title	Target	Objectives	Mode	Duration
Workshop on Essential Abstractions in GCC	People interested in deploying or enhancing GCC	Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals	Lectures, demonstrations, and practicals (experiements and assignments)	Three days
Tutorial on Essential Abstractions in GCC	People interested in knowing about issues in deploying or enhancing GCC	Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals	Lectures and demonstrations	One day
Workshop on Compiler Construction with Introduction to GCC	College teachers	Explaining the theory and practice of compiler contruction and illustrating them with the help of GCC	Lectures, demonstrations, and practicals (experiements and assignments)	Seven days
Tutorial on Demystifying GCC Compilation	Students	Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)	Lectures and demonstrations	Half day

GRC Training Programs

Title	Target	Objectives	Mode	Duration
Workshop on Essential Abs... GCC	People interested	Explaining the essential abstractions in GCC to ... a quick ramp up ... GCC Internals	Lectures, demonstrations, and practicals (experiements and assignments)	Three days
Tutorial on Essential Abstractions in GCC	People interested in knowing ... issues i... deploying or enhancing GCC	Explaining the essential	Lectures and ...onstrations	One day
Workshop on Compiler Construction with Introduction to GCC	College teachers	Explaining the theory and ...tice of compiler ...n and ...ting them with the help of GCC	Lectures, demonstrations, and practicals (experiements and assignments)	Seven days
Tutorial on	Students	Explaining the translation ...CC ...ng ...du... produced by GCC)	Lectures and	Half day

3, 4, and 5 July, 2009
IIT Bombay, Mumbai

(modified version) 9 Jan 2010
ACM PPoPP, Banglore

7-13 Dec 2009,
IIT Bombay, Mumbai

20 Jan 2010, Cummins College, Pune

20 Feb 2010, IIITDM, Jabalpur

06 March 2010, SGGS IET, Nanded

27 March 2010, RSCoE , Pune

25 Apr 2010, Punjabi Univ., Patiala

GRC Training Programs

Title	Target	Objectives	Mode	Duration
Workshop on Essential Abs GCC	People interested	Explaining the essential abstractions in GCC e a qui GCC Inte	Lectures, and assignments)	Three days
Tutorial on Essential Abstractions in GCC	People interested in knowing issues i deploying or enhancing GCC	Explaining the essential	Lectures and monstrations	One day
Workshop on Compiler Construction with Introduction to GCC	College teachers	Explaining the theory and ctice of compiler n and ting them wi help of GCC	Lectures, tions, and assignments)	Seven days
Tutorial on	Students	Explaining the translation C ng du GCC)	Lectures and	Half day

3, 4, and 5 July, 2009
IIT Bombay, Mumbai

5-8 July, 2010
IIT Bombay, Mumbai

(modified version) 9 Jan 2010
ACM PPoPP, Banglore

7-13 Dec 2009,
IIT Bombay, Mumbai

10-15 July 2010,
MIT, Pune

20 Jan 2010,
Cummins College, Pune

20 Feb 2010,
IIITDM, Jabalpur

06 March 2010,
SGGS IET, Nanded

27 March 2010,
RSCoE ,
Pune

25 Apr 2010,
Punjabi Univ.,
Patiala

GRC Training Programs

CS 715: The Design and Implementation of GNU Compiler Generation Framework

- 6 credits semester long course for M.Tech. (CSE) students at IIT Bombay
- Significant component of experimentation with GCC
- Introduced in 2008-2009

Part 7

Conclusions

GCC as a Compiler Generation Framework

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems

GCC as a Compiler Generation Framework

GCC is a strange paradox

- Practically very successful
    - Readily available without any restrictions
    - Easy to use
    - Easy to examine compilation without knowing internals
    - Available on a wide variety of processors and operating systems
    - Can be retargeted to new processors and operating systems
- Quite adhoc

GCC as a Compiler Generation Framework

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation

GCC as a Compiler Generation Framework

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation
  - Needs significant improvements in terms of better algorithms
    Retargetability mechanism, interprocedural optimizations, parallelization, vectorization,

First Level Gray Box Probing of GCC

- Source code is transformed into assembly by lowering the abstraction level step by step to bring it close to machine architecture
- This transformation can be understood to a large extent by observing inputs and output of the different steps in the transformation
- In gcc, the output of almost all the passes can be examined
- The complete list of dumps can be figured out by the command

```
man gcc
```

Gray Box Probing for Optimization

- GCC performs many machine independent optimizations
- The dumps of optimizations are easy to follow, particularly at the GIMPLE level
- It is easy to prepare interesting test cases and observe the effect of transformations
- One optimization often leads to another
  Hence GCC performs many optimizations repeatedly
  (eg. copy propagation, dead code elimination)

GCC Resource Center at IIT Bombay

- Our Goals
  - Demystifying GCC
  - A dream to improve GCC
  - Spreading GCC know-how
- Our Strength
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
- On the horizon
  - Enhancements to data flow analyser
  - Overall re-design of instruction selection mechanism

Acknowledgments

- Many people have directly and indirectlt contributed
  - Amitabha Sanyal, Supratim Biswas
  - Past and present students and project engineers associated with GCC effort at IIT Bombay
    Notably, Dr. Abhijat Vichare and Ms. Sameera Deshpande
  - Participants of GCC Workshops
  - Students of CS715
- Initial version of gray box probing slides was prepared by Ashish Mishra with advice from Amitabha Sanyal
- Examples for machine independent optimizations were prepared by Bageshri Sathe
- Funding
  DIT, IITB, IBM Faculty Award

Last but not the least . . .

Thank You!