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
Nothing is known except the problem
Binding

Overall strategy, algorithm, data structures etc.

No. of unbound objects

Conceptualisation

Time

Sep 2010

Uday Khedker, IIT Bombay
Binding

Functions, variables, their types etc.

Conceptualisation  Coding

No. of unbound objects

Time

Sep 2010
Uday Khedker, IIT Bombay
Binding

No. of unbound objects

Conceptualisation  Coding  Compiling

Machine instructions, registers etc.

Time
Binding

No. of unbound objects

Addresses of functions, external data etc.

Conceptualisation  Coding  Compiling  Linking

Time
Binding

No. of unbound objects

Conceptualisation  Coding  Compiling  Linking  Loading

Actual addresses of code and data

Time

Sep 2010

Uday Khedker, IIT Bombay
### Binding

<table>
<thead>
<tr>
<th>Values of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Loading</td>
</tr>
<tr>
<td>Linking</td>
</tr>
<tr>
<td>Compiling</td>
</tr>
<tr>
<td>Coding</td>
</tr>
<tr>
<td>Conceptualisation</td>
</tr>
<tr>
<td>No. of unbound objects</td>
</tr>
</tbody>
</table>

Sep 2010

Uday Khedker, IIT Bombay
Implementation Mechanisms

Source Program → Translator → Target Program → Machine
Implementation Mechanisms

Source Program → Translator → Target Program → Machine

Source Program → Input Data → Interpreter → Machine
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

```
Program Specification
↓
Translation
↓
Machine
```
Implementation Mechanisms as “Bridges”

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

Program Specification

Translation

Interpretation

Machine
Implementation Mechanisms as “Bridges”

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

Program Specification

Translation

Interpretation

Machine

State: Variables
Operations: Expressions, Control Flow

State: Memory, Registers
Operations: Machine Instructions
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

\[ a = \text{b}<10?\text{b}:\text{c}; \]

Spim Assembly Equivalent

```
lw $t0, 4($fp) ; \quad \text{t0} \leftarrow \text{b} \quad \# \text{Is b smaller}
slti $t0, $t0, 10 ; \quad \text{t0} \leftarrow \text{t0} < 10 \quad \# \text{than 10?}
not $t0, $t0 ; \quad \text{t0} \leftarrow \neg \text{t0}
bgtz $t0, L0: ; \quad \text{if t0} > 0 \text{ goto L0}
lw $t0, 4($fp) ; \quad \text{t0} \leftarrow \text{b} \quad \# \text{YES}
b L1: ; \quad \text{goto L1}
L0: lw $t0, 8($fp) ; \quad \text{L0: t0} \leftarrow \text{c} \quad \# \text{NO}
L1: sw 0($fp), $t0 ; \quad \text{L1: a} \leftarrow \text{t0}
```
High and Low Level Abstractions

Input C statement

\[ a = b < 10 ? b : c; \]

Spim Assembly Equivalent

\[
\begin{align*}
\text{lw} & \quad \text{$t0, 4($fp\)}; \quad \text{t0 <- b} \quad \# \text{Is b smaller} \\
\text{slti} & \quad \text{$t0, t0, 10\)}; \quad \text{t0 <- t0 < 10} \quad \# \text{than 10?} \\
\text{not} & \quad \text{$t0, t0\)}; \quad \text{t0 <- !t0} \\
\text{bgtz} & \quad \text{$t0, L0\)}; \quad \text{if t0 > 0 goto L0} \\
\text{lw} & \quad \text{$t0, 4($fp\)}; \quad \text{t0 <- b} \quad \# \text{YES} \\
\text{lw} & \quad \text{$t0, 8($fp\)}; \quad \text{t0 <- c} \quad \# \text{NO} \\
\text{sw} & \quad \text{0($fp), t0\)}; \quad \text{a <- t0} \\
\end{align*}
\]

NOT Condition

True Part

False Part
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
```

NOT Condition

Conditional jump

Fall through

True Part

False Part

NOT Condition

Input C statement

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

- Translation = Analysis + Synthesis
- Interpretation = Analysis + Execution
Implementation Mechanisms

- Translation = Analysis + Synthesis
- Interpretation = Analysis + Execution

- Translation Instructions $\rightarrow$ Equivalent Instructions
Implementation Mechanisms

- Translation = Analysis + Synthesis
  
- Interpretation = Analysis + Execution

- Translation Instructions $\rightarrow$ Equivalent Instructions

- Interpretation Instructions $\rightarrow$ Actions Implied by Instructions
Language Implementation Models

- Analysis
- Synthesis
- Execution
- Compilation
- Interpretation
Language Processor Models

- Front End
- Optimizer
- Back End
- Virtual Machine

Programming Languages:
- C, C++
- Java, C#
Typical Front Ends

Parser
Typical Front Ends

Source Program

Parser

Tokens

Scanner
Typical Front Ends

Source Program → Scanner → Tokens → Parser → AST or Linear IR + Symbol Table → Semantic Analyzer

Parser

AST

Parse Tree

Sep 2010
Uday Khedker, IIT Bombay
Typical Front Ends

Source Program \rightarrow Parser \rightarrow AST or Linear IR + Symbol Table

Scanner \rightarrow Tokens \rightarrow AST

Semantic Analyzer \rightarrow AST

Symtab Handler \rightarrow Error Handler
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

M/c Ind. IR

M/c Ind. Optimizer

M/c Ind. IR

Code Generator

M/c Dep. IR
Typical Back Ends

- Compile time evaluations
- Eliminating redundant computations
- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

M/c Ind. Optimizer → M/c Ind. IR → Code Generator → M/c Dep. IR

M/c Dep. Optimizer

Assembly Code

Sep 2010

Uday Khedker, IIT Bombay
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

- Scanner
- Semantic Analyser
- Symtab Handler

Parser

Source Program
The Structure of a Simple Compiler

- **Parser**
- **Scanner**
- **Semantic Analyser**
- **Syntab Handler**
- **Instruction Selector**
- **Register Allocator**
- **Assembly Emitter**

Source Program \(\rightarrow\) AST \(\rightarrow\) Insn \(\rightarrow\) Assembly Program
The Structure of a Simple Compiler

Front End
- Scanner
- Semantic Analyser
- Syntab Handler
- Parser

Back End
- Instruction Selector
- Register Allocator
- Assembly Emitter
- Assembly Program

Source Program

AST

Insn
Translation Sequence in Our Compiler: Parsing

Input

a=b<10?b:c;
Translation Sequence in Our Compiler: Parsing

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

Parse Tree:

```
AsgnStmt
   /
  /   
Lhs  =  E  
    /   /  
   name E  ?  E  :
   /   /   /   /   
  E  <  E  name  name
   /   /     /   
 name num name
```

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

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

**Input**

```
AsgnStmnt
```

```
Lhs = E ;
```

```
name E ? E : E
```

```
E < E name name
```

```
name num
```

**Parse Tree**
Translation Sequence in Our Compiler: Semantic Analysis

Input

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

Parse Tree

```
AsgnStmtn

Lhs = E ;

```

Abstract Syntax Tree (with attributes)

```
name (a, int) ?: (int)

(name b, int) (c, int)

(name b, int) (10, int)

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

Input

Parse Tree

Abstract Syntax Tree (with attributes)
Translation Sequence in Our Compiler: IR Generation

Input:

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

**Tree List**

<table>
<thead>
<tr>
<th>Input</th>
<th>Tree List</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = b &lt; 10 ? b : c;</code></td>
<td><code>T_0</code></td>
</tr>
<tr>
<td></td>
<td><code>b &lt; 10</code></td>
</tr>
<tr>
<td></td>
<td><code>T_0</code></td>
</tr>
<tr>
<td></td>
<td><code>IfGoto</code></td>
</tr>
<tr>
<td></td>
<td><code>T_1</code></td>
</tr>
<tr>
<td></td>
<td><code>Goto</code></td>
</tr>
<tr>
<td></td>
<td><code>L0:</code></td>
</tr>
<tr>
<td></td>
<td><code>L1:</code></td>
</tr>
</tbody>
</table>

**Parse Tree**

```
AsgnStmt
  Lhs = E ;
  name E ? E : E
  E < E name name

Parse Tree

= name (a,int) ?: (int)
  < bool name (b,int) name (c,int)
  name (b,int) num (10,int)
```

**Abstract Syntax Tree (with attributes)**

**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

\[ a = \text{b < 10?b: c;} \]

**Input**

**Tree List**

- \( T_0 = \text{b < 10} \)
- \( T_0 \text{ IfGoto} \)
- \( T_0 \text{ Not L0:} \)
- \( T_0 = \text{b} \)
- \( T_1 = \text{b} \)
- \( T_0 \text{ Goto L1:} \)
- \( T_1 \text{ L0:} \)
- \( T_1 = \text{c} \)
- \( T_1 \text{ L1:} \)
- \( T_1 = \text{a} \)

**Parse Tree**

**Abstract Syntax Tree (with attributes)**

- \( \text{name (a, int)} \)
- \( \text{?: (int)} \)
- \( \text{< (bool)} \)
- \( \text{name (b, int)} \)
- \( \text{name (c, int)} \)
- \( \text{name (b, int)} \)
- \( \text{num (10, int)} \)
Translation Sequence in Our Compiler: Instruction Selection

\[ a = b < 10 ? b : c; \]

Input

Tree List

\[
T_0 = b < 10
\]

\[
\text{IfGoto}
\]

\[
\text{Not} \quad \text{L0:}
\]

\[
T_0
\]

\[
T_1 = b
\]

\[
\text{Goto}
\]

\[
\text{L1:}
\]

\[
L0: =
\]

\[
T_1 = c
\]

\[
L1: =
\]

\[
a = T_1
\]

Parse Tree

Abstract Syntax Tree (with attributes)

Issues:

- Cover trees with as few machine instructions as possible
- Use temporaries and local registers
Translation Sequence in Our Compiler: Emitting Instructions

Input

\[ a = b < 10 \text{?} b : c; \]

Tree List

\[ T_0 \]

\[ \Rightarrow b < 10 \]

\[ T_1 \]

\[ b \]

\[ \Rightarrow \text{IfGoto} \]

\[ \text{Not} \]

\[ \Rightarrow \text{L0:} \]

\[ T_0 \]

\[ T_1 \]

\[ \Rightarrow \text{Goto} \]

\[ \Rightarrow \text{L1:} \]

\[ L0: \]

\[ T_1 \]

\[ \Rightarrow \text{c} \]

\[ L1: \]

\[ \Rightarrow a \]

\[ T_1 \]

Parse Tree

Abstract Syntax Tree (with attributes)

\[ = \]

\[ \Rightarrow \text{name} (a, \text{int}) \text{? : (int)} \]

\[ \Rightarrow < \]

\[ \Rightarrow \text{name} (b, \text{int}) \text{num (10, int)} \]

Instruction List

\[ T_0 \leftarrow b \]

\[ T_0 \leftarrow T_0 < 10 \]

\[ T_0 \leftarrow \neg T_0 \]

if \( T_0 > 0 \) goto L0:

\[ T_1 \leftarrow b \]

\[ \text{goto L1:} \]

\[ L0: \]

\[ T_1 \leftarrow c \]

\[ L1: \]

\[ a \leftarrow T_1 \]
Translation Sequence in Our Compiler: Emitting Instructions

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

**Input**

**AsgnStmt**

```
Lhs = E
```

**E**

```
? E : E
```

**E**

```
< name name name num
```

**Tree List**

```
T0 = b < 10
```

**IfGoto**

```
Not L0:
```

**L0:**

```
T0 = c
```

**L1:**

```
a ← T1
```

**AsgnStmt**

**Issues:**

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

**Abstract Syntax Tree (with attributes)**

```
name (a, int) ?: (int) < name (b, int) name (c, int)
```

**Instruction List**

```
T0 ← b
T0 ← T0 < 10
T0 ← ! T0
if T0 > 0 goto L0:
T1 ← b
goto L1:
L0: T1 ← c
L1: a ← T1
```

**Assembly Code**

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

Sep 2010

Uday Khedker, IIT Bombay
Part 3

GCC ≡ The Great Compiler Challenge
What is GCC?

- For the GCC developer community: The GNU Compiler Collection
- For other compiler writers: The Great Compiler Challenge 😊
The Gnu Tool Chain

Source Program

\[ \text{gcc} \]

Target Program
The Gnu Tool Chain

Source Program

\[ \text{gcc} \]

Target Program

cc1
The Gnu Tool Chain

Source Program

\[ \text{gcc} \]

\[ \rightarrow \]

\[ \text{cc1} \]

\[ \leftrightarrow \]

\[ \text{cpp} \]

Target Program
The Gnu Tool Chain

Source Program

⇒

Target Program

gcc

⇒

cc1

⇒

cpp

⇒

as
The GNU Tool Chain

Source Program

gcc

Target Program

cc1

cpp

as

ld
The Gnu Tool Chain

Source Program

 gcc

cc1

as

ld

glibc/newlib

cpp

Target Program
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:**

Sep 2010

Uday Khedker, IIT Bombay
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,
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:**
  - **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:**

- **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:**
  - **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:**
  - **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), IA-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:**
    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:**
  - **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:**
  - **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:**
  - **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:**
  - **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:**

- **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:**
  
  ▶ **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:**

  - **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:**
  
  ▶ **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:**
    

  - **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
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

Compiler Generation Framework

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions
The Architecture of GCC

Compiler Generation Framework

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

Parser
Gimliifier
Tree SSA Optimizer
RTL Generator
Optimizer
Code Generator

Source Program
Generated Compiler (cc1)
Assembly Program

Sep 2010
Uday Khedker, IIT Bombay
The Architecture of GCC

Compiler Generation Framework

Input Language

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Source Program

Generated Compiler (cc1)

Assembly Program

Sep 2010

Uday Khedker, IIT Bombay
The Architecture of GCC

Compiler Generation Framework

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

Selected → Copied → Generated

Parser → Gimiplifier → Tree SSA Optimizer → RTL Generator → Optimizer → Code Generator

Generated Compiler (cc1)

Source Program → Assembly Program

Development Time → Build Time → Use Time

Sep 2010 Uday Khedker, IIT Bombay
Comprehensiveness of GCC: Size

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

(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

```c
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
  
  ```c
  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 cscape
Another Example of The Generation Related Gap

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

<table>
<thead>
<tr>
<th>File</th>
<th>Line</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect2.c</td>
<td>1111</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>fp-test.c</td>
<td>85</td>
<td>main (void )</td>
</tr>
<tr>
<td>gcc.c</td>
<td>6803</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gcov-dump.c</td>
<td>76</td>
<td>main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>gcov iov.c</td>
<td>29</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gcov.c</td>
<td>355</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genattr.c</td>
<td>89</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genattrtab.c</td>
<td>4439</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genautomata.c</td>
<td>9475</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genchecksum.c</td>
<td>67</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gencodes.c</td>
<td>51</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconditions.c</td>
<td>209</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconfig.c</td>
<td>261</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconstants.c</td>
<td>50</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genemit.c</td>
<td>825</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genextract.c</td>
<td>401</td>
<td>main (int argc, char **argv)</td>
</tr>
</tbody>
</table>
Another Example of The Generation Related Gap

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

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

<table>
<thead>
<tr>
<th>Files</th>
<th>i386</th>
<th>mips</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.md</td>
<td>35766</td>
<td>12930</td>
</tr>
<tr>
<td>*.c</td>
<td>28643</td>
<td>12572</td>
</tr>
<tr>
<td>*.h</td>
<td>15694</td>
<td>5105</td>
</tr>
</tbody>
</table>
## Meeting the GCC Challenge

<table>
<thead>
<tr>
<th>Goal of Understanding</th>
<th>Methodology</th>
<th>Needs Examining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation sequence of programs</td>
<td>Gray box probing</td>
<td>No</td>
</tr>
<tr>
<td>Build process</td>
<td>Customizing the configuration and building</td>
<td>Yes</td>
</tr>
<tr>
<td>Retargetability issues and machine descriptions</td>
<td>Incremental construction of machine descriptions</td>
<td>No</td>
</tr>
<tr>
<td>IR data structures and access mechanisms</td>
<td>Adding passes to massage IRs</td>
<td>No</td>
</tr>
<tr>
<td>Retargetability mechanism</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Makefiles</th>
<th>Source</th>
<th>MD</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Meeting the GCC Challenge

<table>
<thead>
<tr>
<th>Goal of Understanding</th>
<th>Methodology</th>
<th>Needs Examining</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Makefiles</td>
</tr>
<tr>
<td>Translation sequence of programs</td>
<td>Gray box probing</td>
<td>No</td>
</tr>
<tr>
<td>Build process</td>
<td>Customizing the configuration and building</td>
<td>Yes</td>
</tr>
<tr>
<td>Retargetability issues and machine</td>
<td>Incremental construction of machine descriptions</td>
<td>No</td>
</tr>
<tr>
<td>IR data structures and access</td>
<td>Adding passes to massage IRs</td>
<td>No</td>
</tr>
<tr>
<td>mechanism</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Retargetability mechanism</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Sep 2010 Uday Khedker, IIT Bombay
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

Transformation from a language to a *different* language

Target Independent \[\rightarrow\] Target Dependent

\[\text{Parse} \rightarrow \text{Gimplify} \rightarrow \text{Tree SSA Optimize} \rightarrow \text{Generate RTL} \rightarrow \text{Optimize RTL} \rightarrow \text{Generate ASM}\]

\[\text{GIMPLE} \rightarrow \text{RTL} \quad \text{RTL} \rightarrow \text{ASM}\]
Basic Transformations in GCC

Transformation from a language to a *different* language

Target Independent \(\leftarrow\) Target Dependent

- Parse
- Simplify
- Tree SSA
- Optimize
- Generate RTL
- Optimize RTL
- Generate ASM

\[\text{GIMPLE} \rightarrow \text{RTL}\]

\[\text{RTL} \rightarrow \text{ASM}\]

GIMPLE Passes

RTL Passes
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

<table>
<thead>
<tr>
<th>Pass Group</th>
<th>Examples</th>
<th>Number of passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowering</td>
<td>GIMPLE IR, CFG Construction</td>
<td>12</td>
</tr>
<tr>
<td>Interprocedural Optimizations</td>
<td>Conditional Constant Propagation, Inlining, SSA Construction, LTO</td>
<td>49</td>
</tr>
<tr>
<td>Intraprocedural Optimizations</td>
<td>Constant Propagation, Dead Code Elimination, PRE</td>
<td>42</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Vectorization, Parallelization</td>
<td>27</td>
</tr>
<tr>
<td>Remaining Intraprocedural Optimizations</td>
<td>Value Range Propagation, Rename SSA</td>
<td>23</td>
</tr>
<tr>
<td>Generating RTL</td>
<td></td>
<td>01</td>
</tr>
<tr>
<td>Total number of passes on GIMPLE</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>Pass Group</td>
<td>Examples</td>
<td>Number of passes</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Intraprocedural Optimizations</td>
<td>CSE, Jump Optimization</td>
<td>21</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Loop Invariant Movement, Peeling, Unswitching</td>
<td>7</td>
</tr>
<tr>
<td>Machine Dependent Optimizations</td>
<td>Register Allocation, Instruction Scheduling, Peephole Optimizations</td>
<td>54</td>
</tr>
<tr>
<td>Assembly Emission and Finishing</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td>Total number of passes on RTL</td>
<td></td>
<td>85</td>
</tr>
</tbody>
</table>
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

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
216rnothrow
217r.final
218r.dfinish
## Total Number of Dumps

<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>Number of Dumps</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>47</td>
<td>Fast compilation</td>
</tr>
<tr>
<td>O1</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>O2</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>O3</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>Os</td>
<td>154</td>
<td>Optimize for space</td>
</tr>
</tbody>
</table>
### 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
- 188r.ira
- 191r.split2
- 193r.pro_and_epilogue
- 206r.stack
- 207r.alignments
- 210r.mach
- 211r.barriers
- 215r.shorten
- 216r.nothrow
- 217r.final
- 218r.dfinish

#### rtl dumps (r)
- 141r.expand
- 142r.sibling
- 144r.initvals
- 145r.unshare
- 146r.vregs
- 147r.into_cfglayout
- 148r.jump
- 160r.reginfo

#### assembly output

---

Sep 2010

Uday Khedker, IIT Bombay
Passes for First Level Graybox Probing of GCC

C Source Code

Parser

AST

Gimplifier

GIMPLE

CFG Generator

CFG

RTL Generator

RTL expand

Reg Allocator

ira

pro_epilogue generation

Pattern Matcher

prologue-epilogue

ASM Program

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
test.c

```c
int a;

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

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

test.c.004t.gimple

```c
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

```c
int a;

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

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

```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

```c
int a;

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

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

}
```

test.c.004t.gimple

```c
x = 10;
y = 5;
D.1954 = x * y;
```

```c
a.0 = a;
```

```c
x = D.1954 + a.0;
```

```c
a.1 = a;
```

```c
D.1957 = a.1 * x;
```

```c
y = y - D.1957;
```
### GIMPLE: Composite Expressions Involving Local and Global Variables

```
int a;

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

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

```
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

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

test.c.004t.gimple

```c
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

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

test.c.004t.gimple

```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;
```
test.c

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

test.c.004t.gimple

```c
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;
```
test.c

int main()
{
    int a[3], x;
    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;
    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

```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

```c
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

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

```gim
main ()
{
    int * D.1953;
    int * * a;
    int * b;
    int c;
    b = &c;
    a = &b;
    D.1953 = *a;
    *D.1953 = 10;
}
```
GIMPLE: Use of Structures

```c
#include <stdio.h>

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";
}
```

Sep 2010
Uday Khedker, IIT Bombay
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";
}
**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";
}
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";

    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;
        D.2055 = s->city;
        D.2055->name = &"Mumbai"[0];
    }
GIMPLE: Pointer to Array

test.c

```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

```c
main ()
{
    int * D.2048;
    int * D.2049;
    int * p_a;
    int a[3];
    p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
}
```
GIMPLE: Pointer to Array

test.c

```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

```c
main ()
{
    int * D.2048;
    int * D.2049;
    int * p_a;
    int a[3];
    p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
    D.2048 = p_a + 4;
    D.2049 = p_a + 8;
    *D.2048 = 20;
    *D.2049 = 30;
}
```
test.c

```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

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

    p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
}
```
GIMPLE: Pointer to Array

```c
int main()
{
    int *p_a, a[3];
    p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
}
```

```gimple
main ()
{
    int * D.2048;
    int * D.2049;
    int * p_a;
    int a[3];
    p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
}
```
test.c

```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

```c
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

```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

```c
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

```c
#include <stdio.h>

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

```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

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

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

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

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

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

```
<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

```plaintext
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

```plaintext
<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**

```c
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**

```c
<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`

```c
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`

```c
<bb 5>:
if (a <= 12)
goto <bb 6>;
else
go to <bb 7>;
<bb 6>:
D.1199 = a + b;
a = D.1199 + c;
<bb 7>:
return;
```
### Control Flow Graph: Textual View

**test.c.004t.gimple**

```c
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**

```c
<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

Block 4:
if(a<=7)

Block 5:
if(a<=12)

Block 3:
a = a +1;

Block 6:
D.1199 = a + b;
a = D.1199 + c;

Block 7:
return;
Control Flow Graph: Pictorial View

test.c.012t.cfg

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

False True

Block 3:
a = a + 1;

Block 4:
if(a<=7)

True False

Block 5:
if(a<=12)

Block 6:
D.1199= a + b;
a= D.1199 + c;

Block 7:
return;
Control Flow Graph: Pictorial View

test.c.012t.cfg

Block 4:
\[ \text{if}(a \leq 7) \]

Block 5:
\[ \text{if}(a \leq 12) \]

Block 3:
\[ a = a + 1; \]

Block 6:
\[ D.1199 = a + b; \]
\[ a = D.1199 + c; \]

Block 7:
\[ \text{return}; \]
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);
}
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);
}
GIMPLE: Function Calls and Call Graph

```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);
}
```

```
printf/3(-1) @0xb73c7ac8 availability:not called by: main/1 (1.00 per call)
calls:
divide/2(-1) @0xb73c7a10 availability:not called by: main/1 (1.00 per call)
calls:
main/1(1) @0xb73c7958 availability:ava 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 availability:ava called by: main/1 (1.00 per call)
calls:
```
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);
}
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

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");
}
int x=2, y=3;

x = y++ + ++x + ++y;

What are the values of x and y?
int x=2, y=3;

x = y++ + ++x + ++y;

What are the values of x and y?

x = 10, y = 5
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of \( x \) and \( y \)?
x = 10, y = 5

\[
\begin{array}{c|c|c}
\text{x} & \text{y} & (y + x) + y \\
\hline
2 & 3 & 11
\end{array}
\]
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of x and y?
\[ x = 10, y = 5 \]
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of x and y?
x = 10, y = 5
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of $x$ and $y$?
x = 10, y = 5

<table>
<thead>
<tr>
<th></th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>4</td>
</tr>
<tr>
<td>$(y + x)$</td>
<td>6</td>
</tr>
<tr>
<td>$(y + x) + y$</td>
<td></td>
</tr>
</tbody>
</table>
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of x and y?
x = 10, y = 5
int x=2,y=3;
x = y++ + ++x + ++y;

What are the values of x and y?
x = 10, y = 5

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>(y + x)</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>(y + x) + y</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
int x=2, y=3;
x = y++ + ++x + ++y;

What are the values of x and y?
x = 10 , y = 5

\[
\begin{array}{c|c}
  x & 3 \\
  y & 5 \\
  (y + x) & 6 \\
  (y + x) + y & 11 \\
\end{array}
\]

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

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

What are the values of x and y?

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

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

What are the values of x and y?
x = 10 , y =5
```

```c
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 [0xffffffffc])) [0 a+0 S4 A32])
     (plus:SI
      (mem/c/i:SI
       (plus:SI
        (reg/f:SI 54 virtual-stack-vars)
        (const_int -4 [0xffffffffc])) [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 [0xffffffff]
        (plus:SI
          (mem/c/i:SI
            (plus:SI
              (reg/f:SI 54 virtual-s
                (const_int -4 [0xff]
                  (const_int 1 [0x1]))))
            (clobber (reg:CC 17 flags)))
          ) -1 (nil)))
    ])
  ])
```

Sep 2010 Uday Khedker, IIT Bombay
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 [0xffffffff])
         (plus:SI
           (mem/c/i:SI
            (plus:SI
              (reg/f:SI 54 virtual-s
               (const_int -4 [0xffffffff])
              (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

\[(\text{insn } 12 \ 11 \ 13 \ 4 \ t.c:24 \ (\text{parallel } [\]
\quad (\text{set } (\text{mem/c/i:SI})
\quad (\text{plus:SI})
\quad (\text{reg/f:SI 54 virtual-vars})
\quad (\text{const_int } -4 \ [0xffffff])
\quad (\text{plus:SI})
\quad (\text{mem/c/i:SI})
\quad (\text{plus:SI})
\quad (\text{reg/f:SI 54 virtual-vars})
\quad (\text{const_int } -4 \ [0xffffff])
\quad (\text{const_int 1 } [0x1])))
\quad (\text{clobber (reg:CC 17 flags)})\]
\) -1 (nil))

\( a \) is a local variable allocated on stack.

### Diagram

```
    set
     /|
    / |
mem  plus
     /|
    / |
plus 1
    /|
   / |
reg 54 -4
    /|
   / |
reg 54 -4
```
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-
          (const_int -4 [0xffffffff])
       (plus:SI
         (mem/c/i:SI
          (plus:SI
            (reg/f:SI 54 virtual-
              (const_int -4 [0xffffffff])
            (mem/c/i:SI
              (plus:SI
                (reg/f:SI 54 virtual-
                  (const_int -4 [0xffffffff])
            (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 [0xffffffff])  
      (plus:SI  
        (mem/c/i:SI  
          (plus:SI  
            (reg/f:SI 54 virtual-s  
              (const_int -4 [0xffffffff])  
            (const_int 1 [0x1])))  
          (clobber (reg:CC 17 flags)))  
        )) -1 (nil))

side-effect of plus may modify condition code register non-deterministically
```
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: test.c.141r.expand

Output with slim suffix

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

Sep 2010 Uday Khedker, IIT Bombay
(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 [0xffffffffc])) [0 a+0 S4 A32])
   (plus:SI
    (mem/c/i:SI
     (plus:SI
      (reg/f:SI 54 virtual-stack-vars)
      (const_int -4 [0xffffffffc])) [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 [0xffffffffc])) [0 a+0 S4 A32])  
(plus:SI  
(mem/c/i:SI  
(plus:SI  
(reg/f:SI 54 virtual-stack-vars)  
(const_int -4 [0xffffffffc])) [0 a+0 S4 A32])  
(const_int 1 [0x1])))  
(clobber (reg:CC 17 flags)))  
]) -1 (nil))
(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 [0xffffffffc])) [0 a+0 S4 A32]) (plus:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars) (const_int -4 [0xffffffffc])) [0 a+0 S4 A32]) (const_int 1 [0x1]))) (clobber (reg:CC 17 flags)) ])) -1 (nil))
(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 [0xffffffffc])) [0 a+0 S4 A32])) (plus:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars) (const_int -4 [0xffffffffc])) [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 [0xffffffffc])) [0 a+0 S4 A32])  
      (plus:SI  
        (mem/c/i:SI  
          (plus:SI  
            (reg/f:SI 54 virtual-stack-vars)  
            (const_int -4 [0xffffffffc])) [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 [0xffffffffc]) ) [0 a+0 S4 A32]) 
    (plus:SI 
      (mem/c/i:SI 
        (plus:SI 
          (reg/f:SI 54 virtual-stack-vars) 
          (const_int -4 [0xffffffffc]) ) [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 [0xffffffffc]) [0 a+0 S4 A32]
    )
    (plus:SI
      (mem/c/i:SI
        (plus:SI
          (reg/f:SI 54 virtual-stack-vars)
          (const_int -4 [0xffffffffc]) [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 [0xffffffffc])) [0 a+0 S4 A32])
    (plus:SI
      (mem/c/i:SI
        (plus:SI
          (reg/f:SI 54 virtual-stack-vars)
          (const_int -4 [0xffffffffc])) [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 [0xffffffffc])) [0 a+0 S4 A32]) 
    (plus:SI 
      (mem/c/i:SI 
        (plus:SI 
          (reg/f:SI 54 virtual-stack-vars) 
          (const_int -4 [0xffffffffc])) [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 [0xfffffffff]) [0 a+0 S4 A32])) (plus:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars) (const_int -4 [0xfffffffff]) [0 a+0 S4 A32])) (const_int 1 [0x1]))) (clobber (reg:CC 17 flags)) ])) -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 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 S4 A32]
   (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 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 S4 A32]) [0 a+0 S4 A32]
  (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 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 S4 A32]
  (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 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 S4 A32]
  (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 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 S4 A32 ])
    (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

Sep 2010 Uday Khedker, IIT Bombay
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))
```
Translation of \( a = a + 1 \) when \( a \) is a formal parameter

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

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

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

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

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

```c
(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-incoming-
                     (const_int 1 [0x1])))
               (reg:SI 53 virtual-incoming-
                  (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

```c
{[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-args) 
      (const_int 4 [0x4])))
  (plus:SI 
    (mem/c/i:SI 
      (plus:SI 
        (reg/f:SI 53 virtual-args) 
        (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

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

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

\[
\begin{align*}
\text{(insn 10 9 11 4 t1.c:25 (parallel }} & \text{(set}} \\
\text{  (mem/c/i:SI} & \text{ (plus:SI}} \\
\text{    (reg/f:SI 53 virtual-} & \text{ (const_int 4 [0x4]))} \\
\text{    (plus:SI} & \text{ (mem/c/i:SI}} \\
\text{      (reg/f:SI 53 virtual-} & \text{ (const_int 4 [0x4])})} \\
\text{      (const_int 1 [0x1]))}) \\
\text{  (clobber (reg:CC 17 flags)))} \\
\text{])) -1 (nil))}
\end{align*}
\]

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

(stack($fp - 4)=r40)

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

Sep 2010
Uday Khedker, IIT Bombay
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

\begin{align*}
\text{(insn 7 6 8 4 test.c:6 (set (reg:SI 39))} \\
&\quad (\text{mem/c/i:SI (plus:SI (reg/f:SI 33 \text{ virtual-stack-vars})}) \\
&\quad \quad (\text{const_int -4 [...]}) [\ldots]) -1 (\text{nil})) \\
\text{(insn 8 7 9 4 test.c:6 (set (reg:SI 40))} \\
&\quad (\text{plus:SI (reg:SI 39}) \\
&\quad \quad (\text{const_int 1 [...]})]) -1 (\text{nil})) \\
\text{(insn 9 8 10 4 test.c:6 (set}} \\
&\quad (\text{mem/c/i:SI (plus:SI (reg/f:SI 33 \text{ virtual-stack-vars})}) \\
&\quad \quad (\text{const_int -4 [...]}) [\ldots]) \\
&\quad (\text{reg:SI 40})) -1 (\text{nil}))
\end{align*}

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

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

```
(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

\[
\begin{align*}
\text{(insn 8 7 9 4 test.c:6 (set (reg:SI 40) (plus:SI (reg:SI 39) (const_int 1 [...])))) -1 (nil))} \\
\text{(insn 9 8 10 4 test.c:6 (set (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars) (const_int -4 [...])) [...]) (reg:SI 40)) -1 (nil))}
\end{align*}
\]

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?

(jumpInsn 15 14 16 4 p1.c:6 (set (pc)
    (ifThenElse (lt (reg:CCGC 17 flags)
        (constInt 0 [0x0]))
        (labelRef 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 [0xffffffff8])) [0 a+0 S4 A32])) -1 (nil))

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

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

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 [0xffffffffc])) [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 [0xffffffff8])) [0 b+0
                (clobber (reg:CC 17 flags))
            )]
        ]
    )
)

(insn 12 11 22 3 test.c:5 (set
    (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xffffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi3_1 (nil))
)
Observing Register Allocation for i386

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

```asm
(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 [0xffffffffc])) [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 [0xffffffff8])) [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 [0xffffffffc])) [0 a+0 S4
        (reg:SI 60)) 44 *movsi1_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 [0xfffffffffc])) [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 [0xffffffff8])) [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 [0xfffffffffc])) [0 a+0 S4
    )
    (reg:SI 60)) 44 *movsi1_1 (nil))
```
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 [0xffffffff8])) ) [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))

Sep 2010
Uday Khedker, IIT Bombay
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

Caller’s Activation Record
Activation Record Structure in Spim

- Caller’s Responsibility
- Caller’s Activation Record
- Parameter $n$
Activation Record Structure in Spim

- Caller’s Activation Record
  - Parameter $n$
  - Parameter $n - 1$

Caller’s Responsibility
Activation Record Structure in Spim

Caller’s Responsibility

Caller’s Activation Record

Parameter $n$

Parameter $n - 1$

...
Activation Record Structure in Spim

Caller’s Activation Record

- Parameter $n$
- Parameter $n - 1$
- ...  
- Parameter 1

Caller’s Responsibility

Argument Pointer
Activation Record Structure in Spim

**Caller’s Activation Record**

- Parameter $n$
- Parameter $n - 1$
- ...  
- Parameter 1

**Return Address**

**Caller’s Responsibility**

**Callee’s Responsibility**
Activating Record Structure in Spim

Caller’s Activation Record

Parameter n
Parameter n - 1
...
Parameter 1

Return Address

Caller’s FPR (Control Link)

Caller’s Responsibility

Caller’s Responsibility

Callee’s Responsibility

Argument Pointer

Sep 2010

Uday Khedker, IIT Bombay
Activation Record Structure in Spim

- Caller’s Activation Record
  - Parameter $n$
  - Parameter $n - 1$
  - ...  
  - Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR

Caller’s Responsibility

Callee’s Responsibility

Argument Pointer
Activation Record Structure in Spim

**Caller’s Activation Record**
- Parameter $n$
- Parameter $n - 1$
- ...
- Parameter 1

**Caller’s Responsibility**
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers

**Callee’s Saved Registers**
- Argument Pointer

Size is known only after register allocation.

Sep 2010

Uday Khedker, IIT Bombay
### Activation Record Structure in Spim

#### Caller’s Activation Record
- Parameter $n$
- Parameter $n - 1$
- ...  
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers
- Local Variable 1

#### Caller’s Responsibility
- Argument Pointer
- Size is known only after register allocation

#### Callee’s Responsibility
- Initial Frame Pointer
### Activation Record Structure in Spim

**Caller’s Activation Record**

- Parameter \( n \)
- Parameter \( n - 1 \)
- \( \ldots \)  
- Parameter 1

**Return Address**

**Caller’s FPR (Control Link)**

**Caller’s SPR**

**Callee Saved Registers**

- Local Variable 1
- Local Variable 2

**Argument Pointer**

- Size is known only after register allocation

**Initial Frame Pointer**

**Callee’s Responsibility**

**Caller’s Responsibility**
Activation Record Structure in Spim

**Caller’s Activation Record**

- Parameter $n$
- Parameter $n - 1$
- ...  
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR

**Callee Saved Registers**

- Callee Saved Registers
- Local Variable 1
- Local Variable 2
- ...  

**Argument Pointer**

Size is known only after register allocation

**Initial Frame Pointer**

Sep 2010  
Uday Khedker, IIT Bombay
Activation Record Structure in Spim

Caller’s Activation Record
- Parameter $n$
- Parameter $n-1$
- ...
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers
- Local Variable 1
- Local Variable 2
- ...
- Local Variable $n$

Argument Pointer
- Size is known only after register allocation
- Initial Frame Pointer
- Stack Pointer

Caller’s Responsibility
- Parameter $n$
- Parameter $n-1$
- ...
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers
- Local Variable 1
- Local Variable 2
- ...
- Local Variable $n$

Callee’s Responsibility
- Argument Pointer
- Size is known only after register allocation
- Initial Frame Pointer
- Stack Pointer

Sep 2010

Uday Khedker, IIT Bombay
## RTL for Function Calls in spim

<table>
<thead>
<tr>
<th>Calling function</th>
<th>Called function</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Allocate memory for actual parameters on stack</td>
<td>• Allocate memory for return value (push)</td>
</tr>
<tr>
<td>• Copy actual parameters</td>
<td>• Store mandatory callee save registers (push)</td>
</tr>
<tr>
<td><strong>Call function</strong></td>
<td>• Set frame pointer</td>
</tr>
<tr>
<td>• Get result from stack (pop)</td>
<td>• Allocate local variables (push)</td>
</tr>
<tr>
<td>• Deallocate memory for activation record (pop)</td>
<td><strong>Execute code</strong></td>
</tr>
<tr>
<td></td>
<td>• Put result in return value space</td>
</tr>
<tr>
<td></td>
<td>• Deallocate local variables (pop)</td>
</tr>
<tr>
<td></td>
<td>• Load callee save registers (pop)</td>
</tr>
<tr>
<td></td>
<td>• Return</td>
</tr>
</tbody>
</table>
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
**Dump file: test.s**

```assembly
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;
}
```

Sep 2010 Uday Khedker, IIT Bombay
**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

```assembly
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;
}
```

Sep 2010 Uday Khedker, IIT Bombay
i386 Assembly

**Dump file: test.s**

```assembly
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;
}
```

Sep 2010 Uday Khedker, IIT Bombay
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

```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;
}
```

- What does this program return?
Example Program 1

```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;
    }

    return a;
}
```

- What does this program return?
- 12
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 -02 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;
}
Example Program 1

Program ccp.c

```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
a = 1
b = 2
c = 3
n = c * 2
```
Example Program 1

Program ccp.c

```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: `a = 1`
- B2: `b = 2`
- B2: `c = 3`
- B2: `n = c * 2`
- B4: If `a <= n` goto B3
- B3: `a = a + 1`
Example Program 1

Program ccp.c

```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: a = 1
- B3: a = a + 1
- B4: if a <= n goto B3
  - F: D.1200 = a + b
  - T: a = D.1200 + c
- B5: if a <= 11 goto B6
  - T: a = a + 1
- B6:

Sep 2010 Uday Khedker, IIT Bombay
Example Program 1

Program ccp.c

```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**: $a = 1$
- **B3**: $b = 2$
- **B4**: $c = 3$
- **B5**: $n = c \times 2$
- **B6**: if $a \leq n$ goto B3
- **B7**: if $a \leq 11$ goto B6
- **D.1200 = a + b**
- **D.1201 = a**
- **F**
- **T**
- **F**
- **T**
- **return a**
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

- **B2**: a = 1; b = 2; c = 3; n = c * 2
- **B4**: if a ≤ n goto B3
- **B5**: if a ≤ 11 goto B6
- **B3**: a = a + 1
- **B6**: D.1200 = a + b; a = D.1200 + c
- **B7**: D.1201 = a; return D.1201

Dump file ccp.c.012t.cfg

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

Sep 2010

Uday Khedker, IIT Bombay
Control Flow Graph: Pictorial and Textual View

Control flow graph

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

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

Dump file ccp.c.012t.cfg

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

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

Sep 2010 Uday Khedker, IIT Bombay
Control Flow Graph: Pictorial and Textual View

Control flow graph

Directions:
- **F**: Follow the *false* path.
- **T**: Follow the *true* path.

**B2**

\[
\begin{align*}
a &= 1 \\
b &= 2 \\
c &= 3 \\
n &= c \times 2
\end{align*}
\]

**B4**

\[
\text{if } a \leq n \text{ goto } B3
\]

**B3**

\[
a = a + 1
\]

**B5**

\[
\text{if } a \leq 11 \text{ goto } B6
\]

**B6**

\[
\begin{align*}
D.1200 &= a + b \\
a &= D.1200 + c
\end{align*}
\]

**B7**

\[
D.1201 = a \\
\text{return } D.1201
\]

**Dump file ccp.c.012t.cfg**

\[
<bb 5>:
\]

\[
\text{if } (a \leq 11) \\
\text{goto } <bb 6>;
\]

\[
\text{else} \\
\text{goto } <bb 7>;
\]

\[
<bb 6>:
\]

\[
D.1200 = a + b; \\
a = D.1200 + c;
\]
Control Flow Graph: Pictorial and Textual View

### Control Flow Graph

```
| B2       | a = 1  
| b = 2  
| c = 3  
| n = c * 2 |

| B4       | if a ≤ n goto B3 |

| B5       | if a ≤ 11 goto B6 |
|          | a = a + 1 B3 |

F

| B6       | D.1200 = a + b   |
|          | a = D.1200 + c   |

F

| B7       | D.1201 = a       |
|          | return D.1201    |
```

### Dump file ccp.c.012t.cfg

```
<bb 7>:
D.1201 = a;
return D.1201;
```
Single Static Assignment (SSA) Form

Control flow graph

B2
a = 1; b = 2
b = 3; n = c * 2

B4
if a \leq n \text{ goto B3}

B5
if a \leq 11 \text{ goto B6}

B6
D.1200 = a + b
a = D.1200 + c

B7
D.1201 = a
return D.1201

SSA Form
Single Static Assignment (SSA) Form

Control flow graph

 SSA Form

B2

\[ a = 1; \ b = 2 \]
\[ c = 3; \ n = c \times 2 \]

B4

if \( a \leq n \) goto B3

B5

if \( a \leq 11 \) goto B6

B6

D.1200 = a + b
a = D.1200 + c

B7

D.1201 = a
return D.1201

B3

a = a + 1
Single Static Assignment (SSA) Form

Control flow graph

SSA Form

B2

a = 1; b = 2
c = 3; n = c * 2

B4

if a ≤ n goto B3

B5

if a ≤ 11 goto B6

B6

D.1200 = a + b
a = D.1200 + c

B3

a = a + 1

B7

D.1201 = a
return D.1201
Single Static Assignment (SSA) Form

Control flow graph

SSA Form

B2

\[a = 1; b = 2\]
\[c = 3; n = c \times 2\]

B4

if \(a \leq n\) goto B3

B5

if \(a \leq 11\) goto B6

B6

\[D.1200 = a + b\]
\[a = D.1200 + c\]

B7

\[D.1201 = a\]
return \(D.1201\)

B3

\[a = a + 1\]
Single Static Assignment (SSA) Form

Control flow graph

SSA Form
Single Static Assignment (SSA) Form

Control flow graph

SSA Form

```
a = 1; b = 2
```

```
c = 3; n = c * 2
```

```
if a \leq n goto B3
```

```
B2
```

```
a = a + 1
```

```
B3
```

```
if a \leq 11 goto B6
```

```
B5
```

```
D.1200 = a + b
```

```
a = D.1200 + c
```

```
B6
```

```
D.1201 = a
```

```
return D.1201
```

```
B7
```

```
Uday Khedker, IIT Bombay
```

Sep 2010
Single Static Assignment (SSA) Form

Control flow graph

SSA Form

a = 1; b = 2
c = 3; n = c * 2

if a ≤ n goto B3

a = a + 1

if a ≤ 11 goto B6

D.1200 = a + b
a = D.1200 + c

D.1201 = a
return D.1201

a_{3} = 1; b_{4} = 2
c_{5} = 3; n_{6} = c_{5} * 2

a_{1} = \phi (a_{3}, a_{7})
if a_{1} ≤ n_{6}

a_{7} = a_{1} + 1

if a_{1} ≤ 11

D.1200_{8} = a_{1} + b_{4}
a_{9} = D.1200_{8} + c_{5}

D.1201_{10} = a_{2}
return D.1201_{10}
Properties of SSA Form

- A \( \phi \) 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 \( \phi \) functions as possible to ensure the above properties
SSA Form: Pictorial and Textual View

CFG in SSA form

B2

\[ a_3 = 1; b_4 = 2 \]
\[ c_5 = 3; n_6 = c_5 \times 2 \]

B4

\[ a_1 = \phi (a_3, a_7) \]
if \( a_1 \leq n_6 \)

B3

\[ a_7 = a_1 + 1 \]

B5

if \( a_1 \leq 11 \)

B6

\[ D.1200_8 = a_1 + b_4 \]
\[ a_9 = D.1200_8 + c_5 \]

B7

\[ a_2 = \phi (a_1, a_9) \]
\[ D.1201_10 = a_2 \]
return \( D.1201_10 \)

Dump file ccp.c.023t.ssa
IITM, Chennai  
GRC: First Level Gray Box Probing of GCC  
76/136

SSA Form: Pictorial and Textual View

CFG in SSA form

\[ a_3 = 1; \ b_4 = 2 \]
\[ c_5 = 3; \ n_6 = c_5 \times 2 \]

\[ a_1 = \phi (a_3, a_7) \]
if \( a_1 \leq n_6 \)

\[ a_7 = a_1 + 1 \]

\[ \text{if } a_1 \leq 11 \]

\[ D.1200_8 = a_1 + b_4 \]
\[ a_9 = D.1200_8 + c_5 \]

\[ a_2 = \phi (a_1, a_9) \]
\[ D.1201_10 = a_2 \]
return \[ D.1201_10 \]

Dump file ccp.c.023t.ssa

<bb 2>:
\[ a_3 = 1; \]
\[ b_4 = 2; \]
\[ c_5 = 3; \]
\[ n_6 = c_5 \times 2; \]
goto <bb 4>;

<bb 3>:
\[ a_7 = a_1 + 1; \]
CFG in SSA form

```plaintext
B2
\[ a_{3} = 1; \ b_{4} = 2 \]
\[ c_{5} = 3; \ n_{6} = c_{5} \times 2 \]

B4
\[ a_1 = \phi (a_{3}, a_{7}) \]
if \( a_1 \leq n_6 \)

B3
\[ a_7 = a_1 + 1 \]

B5
if \( a_1 \leq 11 \)

B6
\[ D.1200_8 = a_1 + b_4 \]
\[ a_9 = D.1200_8 + c_5 \]

B7
\[ a_2 = \phi (a_1, a_9) \]
\[ D.1201_10 = a_2 \]
return \( D.1201_10 \)
```

Dump file ccp.c.023t.ssa

```plaintext
<bb 4>:
# a_1 = PHI <a_{3}(2), a_{7}(3)>
if (a_1 \leq n_6)
goto <bb 3>
else
goto <bb 5>

<bb 5>:
if (a_1 \leq 11)
goto <bb 6>
else
goto <bb 7>
```
SSA Form: Pictorial and Textual View

CFG in SSA form

### B2
\[ a_3 = 1; b_4 = 2 \]
\[ c_5 = 3; n_6 = c_5 \times 2 \]

### B4
\[ a_1 = \phi (a_3, a_7) \]
\[ \text{if } a_1 \leq n_6 \]

### B3
\[ a_7 = a_1 + 1 \]
\[ \text{T} \]

### B5
\[ \text{if } a_1 \leq 11 \]
\[ \text{F} \]

### B6
\[ \text{F} \]
\[ D.1200_8 = a_1 + b_4 \]
\[ a_9 = D.1200_8 + c_5 \]

### B7
\[ a_2 = \phi (a_1, a_9) \]
\[ D.1201_10 = a_2 \]
\[ \text{return } D.1201_10 \]

### Dump file ccp.c.023t.ssa

\[ <\text{bb 6}>: \]
\[ D.1200_8 = a_1 + b_4; \]
\[ a_9 = D.1200_8 + c_5; \]

\[ <\text{bb 7}>: \]
\[ \# a_2 = \text{PHI } <a_1(5), a_9(6)> \]
\[ D.1201_10 = a_2; \]
\[ \text{return } D.1201_10; \]
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 Renaming

Input dump: ccp.c.023t.ssa

\[ \text{D.1201}_{10} = a_2; \]
\[ \text{return D.1201}_{10}; \]

Output dump: ccp.c.026t.copyrename1

\[ \text{a}_{10} = a_2; \]
\[ \text{return a}_{10}; \]
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>;

Sep 2010 Uday Khedker, IIT Bombay
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

\[ a_3 = 1; b_4 = 2 \]
\[ c_5 = 3; n_6 = c_5 \times 2 \]

\[ a_1 = \phi(a_3, a_7) \]
if \( a_1 \leq n_6 \)

\[ a_7 = a_1 + 1 \]

if \( a_1 \leq 11 \)

\[ D.1200_8 = a_1 + b_4 \]
\[ a_9 = D.1200_8 + c_5 \]

\[ a_2 = \phi(a_1, a_9) \]
\[ D.1201_{10} = a_2 \]
return \( D.1201_{10} \)
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>:
    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

B2

B4

\[ a_{-1} = \phi (1, a_{-7}) \]

if \( a_{-1} \leq 6 \)

B3

\[ a_{7} = a_{-1} + 1 \]

B5

if \( a_{-1} \leq 11 \)

B6

\[ D.1200_8 = a_{-1} + 2 \]
\[ a_{9} = D.1200_8 + 3 \]

B7

\[ a_{2} = \phi (a_{-1}, a_{9}) \]
return \( a_{2} \)
**IITM, Chennai**

**GRC: First Level Gray Box Probing of GCC**

### Loop Unrolling

```
B2

B4
a_1 = \phi(1, a_7)
if a_1 \leq 6
a_7 = a_1 + 1

B3 T

B5 F
if a_1 \leq 11

B6 T
D.1200_8 = a_1 + 2
a_9 = D.1200_8 + 3

B6 F

B7
a_2 = \phi(a_1, a_9)
return a_2
```
Loop Unrolling

B2

B4

\[ a_{-1} = \phi(1, a_{-7}) \]
if \( a_{-1} \leq 6 \)

B3

if \( a_{-1} \leq 11 \)

B5

D.1200_8 = a_{-1} + 2
a_{-9} = D.1200_8 + 3

B6

a_{-2} = \phi(a_{-1}, a_{-9})
return a_{-2}

B7

if \( a_{-1} \leq 11 \)

T

F

T

F

a = 2

Sep 2010

Uday Khedker, IIT Bombay
Loop Unrolling

B2

B4

\( a_1 = \phi(1, a_7) \)
\[ \text{if } a_1 \leq 6 \]

B3

\( a_7 = a_1 + 1 \)

B5

\[ \text{if } a_1 \leq 11 \]

B6

\( D.1200.8 = a_1 + 2 \)
\( a_9 = D.1200.8 + 3 \)

B7

\( a_2 = \phi(a_1, a_9) \)
\[ \text{return } a_2 \]
The loop unrolling process involves:

1. Inlining the loop body:
   
   ```
   a_1 = \phi(1, a_7)
   \text{if } a_1 \leq 6
   ```

2. Executing the loop condition:
   
   ```
   a_7 = a_1 + 1
   ```

3. Handling the loop counter:
   
   ```
   \text{if } a_1 \leq 11
   ```

4. Updating variables:
   
   ```
   D.1200_8 = a_1 + 2
   a_9 = D.1200_8 + 3
   ```

5. Inlining the phi function:
   
   ```
   a_2 = \phi(a_1, a_9)
   ```

6. Returning the result:
   
   ```
   \text{return } a_2
   ```
Loop Unrolling

\[ a_1 = \phi(1, a_7) \]
\[ \text{if } a_1 \leq 6 \]
\[ a_7 = a_1 + 1 \]
\[ \text{if } a_1 \leq 11 \]
\[ D.1200_8 = a_1 + 2 \]
\[ a_9 = D.1200_8 + 3 \]
\[ a_2 = \phi(a_1, a_9) \]
\[ \text{return } a_2 \]
Loop Unrolling

B2

B4

a_1 = \phi (1, a_7)
if a_1 \leq 6

T

F

B3

a_7 = a_1 + 1

B5

if a_1 \leq 11

T

F

D.1200_8 = a_1 + 2
a_9 = D.1200_8 + 3

B6

B7

a_2 = \phi (a_1, a_9)
return a_2
Loop Unrolling

\[
\begin{align*}
B2 & : a_3 = 1; b_4 = 2 \\
B4 & : a_1 = \phi (1, a_7) \\
& \text{if } a_1 \leq 6 \\
& \quad a = 4 \\
B3 & : a_7 = a_1 + 1 \\
B5 & : \text{if } a_1 \leq 11 \\
B6 & : D.1200.8 = a_1 + 2 \\
& \quad a_9 = D.1200.8 + 3 \\
B7 & : a_2 = \phi (a_1, a_9) \\
& \text{return } a_2
\end{align*}
\]
B2

B4

a₁ = φ (1, a₇)
if a₁ ≤ 6

B3

a₇ = a₁ + 1

B5

if a₁ ≤ 11

T

B6

D.1200_8 = a₁ + 2
a₉ = D.1200_8 + 3

F

B7

a₂ = φ (a₁, a₉)
return a₂

T

a = 5
Loop Unrolling

\[ a_1 = \phi(1, a_7) \]
\[ \text{if } a_1 \leq 6 \]

\[ a_7 = a_1 + 1 \]

\[ \text{if } a_1 \leq 11 \]

\[ D.1200_8 = a_1 + 2 \]
\[ a_9 = D.1200_8 + 3 \]

\[ a_2 = \phi(a_1, a_9) \]

return \( a_2 \)

Sep 2010

Uday Khedker, IIT Bombay
Loop Unrolling

\[ a_{-1} = \phi (1, a_{-7}) \]

if \( a_{-1} \leq 6 \)

\[ a_{-7} = a_{-1} + 1 \]

if \( a_{-1} \leq 11 \)

\[ D.1200_8 = a_{-1} + 2 \]
\[ a_{-9} = D.1200_8 + 3 \]

\[ a_{-2} = \phi (a_{-1}, a_{-9}) \]

return \( a_{-2} \)
Loop Unrolling

B2

B4

\[ a_1 = \phi (1, a_7) \]
\[ \text{if } a_1 \leq 6 \]
a = 6

B3

\[ a_7 = a_1 + 1 \]

B5

\[ \text{if } a_1 \leq 11 \]

B6

\[ D.1200_8 = a_1 + 2 \]
\[ a_9 = D.1200_8 + 3 \]

B7

\[ a_2 = \phi (a_1, a_9) \]
return \( a_2 \)
Loop Unrolling

B2

B4
\[ a_{-1} = \phi (1, a_7) \]
if \( a_{-1} \leq 6 \)

B3
\[ a_7 = a_{-1} + 1 \]

B5
if \( a_{-1} \leq 11 \)

B6
\[ D.1200_8 = a_{-1} + 2 \]
\[ a_9 = D.1200_8 + 3 \]

B7
\[ a_{-2} = \phi (a_{-1}, a_9) \]
return \( a_{-2} \)

F

T

T

F

a = 7

Sep 2010 Uday Khedker, IIT Bombay
Loop Unrolling

B2

B4
\[ a_1 = \phi (1, a_7) \]
\[ \text{if } a_1 \leq 6 \]

B3
\[ a_7 = a_1 + 1 \]

B5
\[ \text{if } a_1 \leq 11 \]

B6
\[ D.1200_8 = a_1 + 2 \]
\[ a_9 = D.1200_8 + 3 \]

B7
\[ a_2 = \phi (a_1, a_9) \]
\[ \text{return } a_2 \]

F
T

T
F

a = 7
Loop Unrolling

B2

B4

B3

B5

B6

B7

B4

B5

B6

B7

\[ a_1 = \phi(1, a_7) \]

if \( a_1 \leq 6 \)

\[ a_7 = a_1 + 1 \]

if \( a_1 \leq 11 \)

\[ D.1200_8 = a_1 + 2 \]
\[ a_9 = D.1200_8 + 3 \]

\[ a_2 = \phi(a_1, a_9) \]

return \( a_2 \)
Complete Unrolling of Inner Loops

**Dump file:** ccp.c.058t.cunrolli

```plaintext
<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;
```

**Diagram:**

```
<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
```

---

Sep 2010

Uday Khedker, IIT Bombay
Another Round of Constant Propagation

Input:

\begin{align*}
\text{a}_{12} &= 2 \\
\text{a}_{14} &= \text{a}_{12} + 1 \\
\text{a}_{16} &= \text{a}_{14} + 1 \\
\text{a}_{18} &= \text{a}_{16} + 1 \\
\text{a}_{20} &= \text{a}_{18} + 1 \\
\text{a}_{22} &= \text{a}_{20} + 1 \\
\text{if} \quad \text{a}_{22} &\leq 11
\end{align*}

\text{B2}

\begin{align*}
\text{D.1200.8} &= \text{a}_{22} + 2 \\
\text{a}_{9} &= \text{D.1200.8} + 3
\end{align*}

\text{B3}

\begin{align*}
\text{a}_{2} &= \phi (\text{a}_{22}, \text{a}_{9}) \\
\text{return} \quad \text{a}_{2}
\end{align*}

\text{B4}

\text{Dump file: ccpp.c.059t.cpp2}

\text{<bb 2>:}

\begin{align*}
\text{a}_{22} &= 7; \\
\text{a}_{9} &= 12; \\
\text{return} \quad 12;
\end{align*}
Dead Code Elimination Using Copy Propagation

Dump file: ccp.c.059t.copyprop2

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

Dump file: ccp.c.066t.copyprop2

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

```c
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

Sep 2010
Uday Khedker, IIT Bombay
Compilation Command

$gcc -fdump-tree-all -fdump-rtl-all -O2 -S ccp.c
Example Program 2

```c
int f(int b, int c, int n)
{
    int a;
    do
    {
        a = b+c;
    } while (a <= n);
    return a;

}```
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 $\phi$ 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

```c
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
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: \(\$\{(SOURCE\_D)}/<\text{lang\ dir}\rangle\)
- 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)
    
    \[ \text{CPPFLAGS=-fexceptions ./configure --enable-cxx --prefix=/usr/local} \]
  - 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)

  \[
  \begin{align*}
  \text{make} \\
  \text{make check} \\
  \text{sudo make install} \\
  \text{sudo ldconfig}
  \end{align*}
  \]
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

configure
Configuring GCC

- configure.in
- config/*
- config.guess
- config.sub
- configure
Configuring GCC

- configure.in
- config/*
- config.guess
- config.sub
- config.log
- config.cache
- config.status

Sep 2010 Uday Khedker, IIT Bombay
Configuring GCC

- configure.in
- config/*
- config.guess
- config.sub
- config.h.in
- Makefile.in
- config.log
- config.cache
- config.status

Sep 2010 Uday Khedker, IIT Bombay
Configuring GCC

- configure.in
- config/*
- config.guess
- config.sub
- config.log
- config.cache
- config.status
- config.h.in
- Makefile.in
- Makefile
- config.h
## 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
- Download and untar the source
- `cd $(SOURCE_D)`
- `./configure`
- `make`
- `make install`

#### Steps in GCC
- Download and untar the source
- `cd $(BUILD)`
- `${SOURCE_D}/configure`
- `make`
- `make install`
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $(SOURCE_D)</td>
<td>• cd $(BUILD)</td>
</tr>
<tr>
<td>• ./configure</td>
<td>• $(SOURCE_D)/configure</td>
</tr>
<tr>
<td>• make</td>
<td>• make</td>
</tr>
<tr>
<td>• make install</td>
<td>• make install</td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th>BS = HS = TS</th>
<th>Native Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS = HS $\neq$ TS</td>
<td>Cross Build</td>
</tr>
<tr>
<td>BS $\neq$ HS $\neq$ TS</td>
<td>Canadian Cross</td>
</tr>
</tbody>
</table>

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

C

i386

i386

CC

cc
T Notation for a Compiler

input language

C → i386

i386 → CC
T Notation for a Compiler

input language

output language

C ➔ i386

i386 ➔ CC

C

i386

CC

Sep 2010

Uday Khedker, IIT Bombay
T Notation for a Compiler

Input language

Implementation or execution language

Output language

C

i386

cc

Sep 2010 Uday Khedker, IIT Bombay
T Notation for a Compiler

input language

output language

implementation or execution language

name of the translator
Bootstrapping: The Conventional View

Assembly language

Machine language

# Diagram

```
+-----------+     +-----------+
|   m/c     |     |   m/c     |
|assembly   |→    |machine    |
|language   |     |language  |
```

Sep 2010

Uday Khedker, IIT Bombay
Bootstrapping: The Conventional View

input language

implementation language

output language

Assembly language

Machine language

Ass

m/c

M/c
Bootstrapping: The Conventional View

- Implementation language
- Input language
- Output language
Bootstrapping: The Conventional View

- implementation language
- input language
- output language

Diagram:
- $C_0$
- m/c
- ass
- m/c

Level 0 C
Bootstrapping: The Conventional View

Level 1 C

C_1

C_0

input language

m/c

output language

implementation language
Bootstrapping: The Conventional View

- **Level 1 C**
- **input language**
- **output language**
- **implementation language**

- $C_0$
- $C_1$
- m/c
- ass
Bootstrapping: The Conventional View

- input language
- implementation language
- output language

Level n C

C_n

C_{n-1}

m/c
Bootstrapping: The Conventional View

- **Level n C**
- **C\(_n\)**
- **C\(_{n-1}\)**
- **C\(_{n-2}\)**
- **implementation language**
- **m/c**
- **input language**
- **output language**

**Sep 2010**

Uday Khedker, IIT Bombay
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

⇒ 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

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

Sep 2010

Uday Khedker, IIT Bombay
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

Source Program

GCC

Target Program

cc1
cpp
as
ld
glibc/newlib

GCC
A More Detailed Look at Building

Source Program

\[ \text{gcc} \]

\[ \text{cc1} \quad \text{cpp} \]

Partially generated and downloaded source is compiled into executables

\[ \text{as} \]

\[ \text{ld} \]

\[ \text{glibc/newlib} \]

Target Program
A More Detailed Look at Building

Source Program

\[\text{gcc}\]

Target Program

\[\text{cc1} \quad \text{cpp} \]

Partially generated and downloaded source is compiled into executables

\[\text{as} \quad \text{ld} \quad \text{glibc/newlib}\]

Existing executables are directly used

Sep 2010 Uday Khedker, IIT Bombay
A More Detailed Look at Building

Source Program

GCC

Target Program

cc1

cpp

as

ld

Partialy generated and downloaded source is compiled into executables

Existing executables are directly used

glibc/newlib
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
A More Detailed Look at Cross Build

**Stage 1 Build**

- **C**\(\rightarrow\) i386
- **cc**\(\rightarrow\) mips assembly

**Stage 2 Build**

- **C**\(\rightarrow\) mips.a
- **mips.a**\(\rightarrow\) i386

**Requirement:** \(BS = HS = \text{i386}, TS = \text{mips}\)

- *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
This is what we specify

- cd $(BUILD)
This is what we specify

- cd $(BUILD)
- $(SOURCE_D)/configure <options>
  configure output: customized Makefile
This is what we specify

- cd $(BUILD)
- $(SOURCE_D)/configure <options>
  configure output: customized Makefile
- make 2> make.err > make.log
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)
More Details of an Actual Stage 1 Build for C

GCC sources ➔ native cc + native binutils
More Details of an Actual Stage 1 Build for C

GCC sources

native cc +
native binutils

libraries

Sep 2010

Uday Khedker, IIT Bombay
More Details of an Actual Stage 1 Build for C

GCC sources → native cc + native binutils → libraries

- libc++: C preprocessor
- zlib: Data compression
- intl: Internationalization
- libdecnumber: Decimal floating point numbers
- libgomp: GNU OpenMP
More Details of an Actual Stage 1 Build for C

GCC sources →

- native cc + native binutils
  →
  - libraries
  - libiberty
  →
  - fixincl
  →
  - gen*
    →
    - cc1
      →
      - cpp

Uday Khedker, IIT Bombay

Sep 2010
More Details of an Actual Stage 1 Build for C

GCC sources

native cc + native binutils

libraries

libiberty

fixincl

gen*

cc1

cpp

xgcc
More Details of an Actual Stage 1 Build for C

 GCC sources

native cc + native binutils

libraries

libiberty

fixincl

gen*

cc1

cpp

xgcc

target binutils

libgcc
More Details of an Actual Stage 1 Build for C

GCC sources → native cc + native binutils

- libraries
- libiberty
- fixincl
- gen*
- cc1
- cpp

cc + binutils for stage 2

Uday Khedker, IIT Bombay
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 \texttt{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`.

Sep 2010 Uday Khedker, IIT Bombay
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 all 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
Welcome to GCC Resource Center at IIT Bombay

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 pages contain 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 (NRFCOSS I).

Interesting Aspects of GCC

Interestingly, GCC has been one of the first projects of the Free Software Foundation (FSF) to provide a free compiler for its GNU C++.
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
<table>
<thead>
<tr>
<th>Title</th>
<th>Target</th>
<th>Objectives</th>
<th>Mode</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop on Essential Abstractions in GCC</td>
<td>People interested in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Three days</td>
</tr>
<tr>
<td>Tutorial on Essential Abstractions in GCC</td>
<td>People interested in knowing about issues in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures and demonstrations</td>
<td>One day</td>
</tr>
<tr>
<td>Workshop on Compiler Construction with Introduction to GCC</td>
<td>College teachers</td>
<td>Explaining the theory and practice of compiler construction and illustrating them with the help of GCC</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Seven days</td>
</tr>
<tr>
<td>Tutorial on Demystifying GCC Compilation</td>
<td>Students</td>
<td>Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)</td>
<td>Lectures and demonstrations</td>
<td>Half day</td>
</tr>
<tr>
<td>Title</td>
<td>Target</td>
<td>Objectives</td>
<td>Mode</td>
<td>Duration</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>------------</td>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>Workshop on Essential Abstractions in GCC</td>
<td>People interested in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Three days</td>
</tr>
<tr>
<td>Tutorial on Essential Abstractions in GCC</td>
<td>People interested in knowing about issues in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures and demonstrations</td>
<td>One day</td>
</tr>
<tr>
<td>Workshop on Compiler Construction with Introduction to GCC</td>
<td>College teachers</td>
<td>Explaining the theory and practice of compiler construction and illustrating them with the help of GCC</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Seven days</td>
</tr>
<tr>
<td>Tutorial on Demystifying GCC Compilation</td>
<td>Students</td>
<td>Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)</td>
<td>Lectures and demonstrations</td>
<td>Half day</td>
</tr>
</tbody>
</table>

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

Sep 2010

Uday Khedker, IIT Bombay
# GRC Training Programs

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

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

- **5-8 July, 2010**  
  IIT Bombay, Mumbai

  ACM PPoPP, Banglore

- **20 Jan 2010, Cummins College, Pune**

- **27 March 2010, RSCoE, Pune**

- **25 Apr 2010, Punjabi Univ., Patiala**
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 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`.

Sep 2010
Uday Khedker, IIT Bombay
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

Sep 2010
Uday Khedker, IIT Bombay
Acknowledgments

- Many people have directly and indirectly 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!