Outline

- An Overview of Compilation
  Introduction, compilation sequence, compilation models
- GCC: The Great Compiler Challenge
  Difficulties in understanding GCC
- Meeting the GCC Challenge: CS 715
  The course plan

Part 1

Introduction to Compilation

Binding

Nothing is known except the problem
Overall strategy, algorithm, data structures etc.
Functions, variables, their types etc.

Machine instructions, registers etc.

Addresses of functions, external data etc.

Actual addresses of code and data

Values of variables

No. of unbound objects

Conceptualisation Coding Compiling Linking Loading Execution

Time

Uday Khedker
GRC, IIT Bombay
Implementation Mechanisms

Implementation Mechanisms as “Bridges”

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

High and Low Level Abstractions

Input C statement

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

Spim Assembly Equivalent

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

Condition

Input C statement

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

Spim Assembly Equivalent

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

Input C statement

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

Spim Assembly Equivalent

\[
\begin{align*}
\text{lw} & \; \text{t0}, \; 4(\text{fp}) \; ; \; \text{t0} \leftarrow \; b \\
\text{slti} & \; \text{t0}, \; \text{t0}, \; 10 \; ; \; \text{t0} \leftarrow \; \text{t0} < 10 \\
\text{not} & \; \text{t0} \; ; \; \text{t0} \leftarrow \; \neg \text{t0} \\
\text{bgtz} & \; \text{t0}, \; \text{L0} : \; ; \; \text{if} \; \text{t0} \geq 0 \; \text{goto} \; \text{L0} \\
\text{lw} & \; \text{t0}, \; 4(\text{fp}) \; ; \; \text{t0} \leftarrow \; b \\
\text{b} & \; \text{L1} : \; ; \; \text{goto} \; \text{L1} \\
\text{L0} : & \; \text{lw} \; \text{t0}, \; 8(\text{fp}) \; ; \; \text{L0} : \; \text{t0} \leftarrow \; c \\
\text{L1} : & \; \text{sw} \; 0(\text{fp}), \; \text{t0} \; ; \; \text{L1} : \; \text{a} \leftarrow \; \text{t0}
\end{align*}
\]

Implementation Mechanisms

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

Language Implementation Models

Analysis → Synthesis → Compilation → Execution

Language Processor Models

Front End → Optimizer → Back End

C, C++ → Virtual Machine

Java, C#
Typical Front Ends

Parser

AST or Linear IR + Symbol Table

Scanner

Tokens

Parse Tree

Semantic Analyzer

Syntab Handler

Error Handler

Typical Back Ends

m/c Ind. IR

m/c Ind. Optimizer

− Compile time evaluations
− Eliminating redundant computations

Code Generator

m/c Dep. IR

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

Instruction Scheduler

Register Allocator

Peephole Optimizer

Assembly Code

The Structure of a Simple Compiler

Front End

Parser

AST

Instruction Selector

Insn

Assembly Emitter

Scanner

Semantic Analyser

Syntab Handler

Register Allocator

Back End

Assembly Program

Part 2

An Overview of Compilation Phases
Translation Sequence in Our Compiler: Parsing

a = b < 10? b : c;

Input

AsgnStmnt

Lhs = E;

name E ? E : E

name E < E name name

name num

Parse Tree

Issues:

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

Translation Sequence in Our Compiler: Semantic Analysis

a = b < 10? b : c;

Input

AsgnStmnt

Lhs = E;

name E ? E : E

name E < E name name

name num

Parse Tree

Abstract Syntax Tree

(with attributes)

Issues:

• Symbol tables
  Have variables been declared? What are their types?
  What is their scope?
• Type consistency of operators and operands
  The result of computing b < 10? is bool and not int

Translation Sequence in Our Compiler: IR Generation

a = b < 10? b : c;

Input

AsgnStmnt

Lhs = E;

name E ? E : E

name E < E name name

name num

Parse Tree

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 = b < 10? b : c;

Input

AsgnStmnt

Lhs = E;

name E ? E : E

name E < E name name

name num

Parse Tree

Abstract Syntax Tree

(with attributes)

Instruction List

T_0 ← b
T_0 ← T_0 < 10
T_0 ← ! T_0
if T_0 > 0 goto L0:
T_1 ← b
goto L1:
L0: T_1 ← c
L1: a ← T_1

Issues:

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

```
\begin{align*}
\text{Input} & \quad \text{AsgnStmt} \\
\text{Lhs} & = \quad E \\
E & ? \quad E \quad : \quad E \\
E & < \quad \quad \text{name} \quad \text{name} \\
\text{name} & \text{num} \\
= & \quad \text{parse tree} \\
\text{name} & \text{(a,int)} \quad ?: \quad \text{(int)} \quad < \\
\text{name} & \text{(b,int)} \quad \text{name} \quad \text{name} \quad \text{name} \\
\text{L0:} & \quad \text{T0} = \quad \text{T0} \quad \text{L1:} \\
\text{L1:} & \quad \text{T1} = \quad \text{b} \\
\text{T0} & \quad \text{b} \quad \text{T0} \quad \text{L0:} \\
\text{Goto} & \quad \text{b} \quad \text{goto L1:} \\
\text{T1} & \quad \text{c} \\
\text{L0:} & \quad \text{T1} = \quad \text{c} \\
\text{L1:} & \quad \text{T1} = \quad \text{T1} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{b} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{T0} \quad \text{<} \quad \text{10} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{!} \quad \text{T0} \\
\text{if} \quad \text{T0} \quad \text{>} \quad \text{0} \quad \text{goto} \quad \text{L0:} \\
\text{T1} & \quad \text{a} \quad \text{L0:} \\
\text{goto} \quad \text{L1:} \\
\text{T1} & \quad \text{c} \\
\text{L1:} & \quad \text{a} \quad \text{←} \quad \text{T1} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{b} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{T0} \quad \text{<} \quad \text{10} \\
\text{T0} & \quad \quad \quad \text{T0} \quad \text{←} \quad \text{!} \quad \text{T0} \\
\text{if} \quad \text{T0} \quad \text{>} \quad \text{0} \quad \text{goto} \quad \text{L0:} \\
\end{align*}
```

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

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

Assembly Code

Retargetability in Aho Ullman Model

### Compilation Models

**Aho Ullman Model**
- **Front End**: AST
- **Optimizer**: m/c indep. IR
- **Code Generator**: Target Program

### Davidson Fraser Model**
- **Front End**: AST
- **Expander**: register transfers
- **Optimizer**: register transfers
- **Recognizer**: Target Program

---

### Aho Ullman: Instruction selection
- over optimized IR using
- intelligent tree tiling based algorithms

### Davidson Fraser: Instruction selection
- over AST using
- simple full tree matching based algorithms that generate
- naive code which is
  - machine dependent, and is
  - optimized subsequently

---

### Instruction selection
- over optimized IR using
- intelligent tree tiling based algorithms

### Key idea in retargetability:
- Machine independent IR is expressed in the form of trees
- Machine instructions are described in the form of trees
- Trees in the IR are tiled using the instruction trees
Retargetability in Davidson Fraser Model

Instruction selection
- over AST using
- simple full tree matching based algorithms that generate
- naive code which is
  - machine dependent, and is
  - optimized subsequently

Key idea in retargetability:
- Register transfers are machine specific but
- their form is machine independent

Full Tree Matching (Davidson Fraser Model)

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Machine Instructions</th>
<th>Subject Tree (IR)</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>a + b</td>
<td>t * b c</td>
</tr>
<tr>
<td>- reg - reg</td>
<td>a - + b</td>
<td>a + t</td>
</tr>
<tr>
<td>* reg * reg</td>
<td>a * b c</td>
<td>a + t</td>
</tr>
</tbody>
</table>
**Full Tree Matching (Davidson Fraser Model)**

Instructions are viewed as independent non-composable rules

---

**Tree Tiling (Aho Ullman Model)**

Instructions are viewed as composable rules
Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Machine Instructions</th>
<th>Subject Tree (IR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg Reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← + Reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← * Reg</td>
<td></td>
</tr>
</tbody>
</table>

Uday Khedker
GRC, IIT Bombay
Why is Understanding GCC Difficult?

• Some of the obvious reasons:
  ◦ Comprehensiveness
    GCC is a production quality framework in terms of completeness and practical usefulness
  ◦ Open development model
    Could lead to heterogeneity. Design flaws may be difficult to correct
  ◦ Rapid versioning
    GCC maintenance is a race against time. Disruptive corrections are difficult

• Deeper technical reasons:
  ◦ GCC is not a compiler but a compiler generation framework
    Two distinct gaps that need to be bridged:
      ◦ Input-output of the generation framework
      ◦ Input-output of the generated compiler
  ◦ GCC generated compiler uses a derivative of the Davidson-Fraser model of compilation
    ◦ Early instruction selection
    ◦ Machine dependent intermediate representation
    ◦ Simplistic instruction selection and retargetability mechanism
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, RISC/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

## Comprehensiveness of GCC: Size

### Overall size

<table>
<thead>
<tr>
<th>Language</th>
<th>Subdirectories</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc-4.4.2</td>
<td>3794</td>
<td>62301</td>
</tr>
<tr>
<td>gcc-4.5.0</td>
<td>4056</td>
<td>65639</td>
</tr>
<tr>
<td>gcc-4.6-20101225</td>
<td>4369</td>
<td>70374</td>
</tr>
</tbody>
</table>

### Core size (src/gcc)

<table>
<thead>
<tr>
<th>Language</th>
<th>Subdirectories</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc-4.4.2</td>
<td>257</td>
<td>30163</td>
</tr>
<tr>
<td>gcc-4.5.0</td>
<td>283</td>
<td>32723</td>
</tr>
<tr>
<td>gcc-4.6-20101225</td>
<td>335</td>
<td>35986</td>
</tr>
</tbody>
</table>

### Machine Descriptions (src/gcc/config)

<table>
<thead>
<tr>
<th>Language</th>
<th>Subdirectories</th>
<th>.c files</th>
<th>.h files</th>
<th>.md files</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc-4.4.2</td>
<td>36</td>
<td>241</td>
<td>426</td>
<td>206</td>
</tr>
<tr>
<td>gcc-4.5.0</td>
<td>42</td>
<td>275</td>
<td>478</td>
<td>206</td>
</tr>
<tr>
<td>gcc-4.6-20101225</td>
<td>42</td>
<td>269</td>
<td>486</td>
<td>251</td>
</tr>
</tbody>
</table>
### Line Count of gcc-4.6.2-20101225/gcc snapshot

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Comment %</th>
<th>Blank</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>13296</td>
<td>1254253</td>
<td>282582</td>
<td>18.4%</td>
<td>283766</td>
<td>1820601</td>
</tr>
<tr>
<td>ada</td>
<td>4196</td>
<td>636876</td>
<td>294321</td>
<td>31.6%</td>
<td>217401</td>
<td>1148598</td>
</tr>
<tr>
<td>cpp</td>
<td>7418</td>
<td>184186</td>
<td>52163</td>
<td>22.1%</td>
<td>54048</td>
<td>290397</td>
</tr>
<tr>
<td>fortranfixed</td>
<td>2086</td>
<td>67988</td>
<td>1521</td>
<td>2.2%</td>
<td>9079</td>
<td>75858</td>
</tr>
<tr>
<td>assembler</td>
<td>132</td>
<td>31092</td>
<td>7243</td>
<td>18.9%</td>
<td>4770</td>
<td>43105</td>
</tr>
<tr>
<td>autconf</td>
<td>3</td>
<td>26996</td>
<td>10</td>
<td>0.0%</td>
<td>3383</td>
<td>30389</td>
</tr>
<tr>
<td>fortranfree</td>
<td>652</td>
<td>10898</td>
<td>2376</td>
<td>17.9%</td>
<td>1314</td>
<td>14588</td>
</tr>
<tr>
<td>objective&lt;</td>
<td>391</td>
<td>10155</td>
<td>1654</td>
<td>14.0%</td>
<td>2830</td>
<td>14639</td>
</tr>
<tr>
<td>make</td>
<td>3</td>
<td>5340</td>
<td>1027</td>
<td>16.1%</td>
<td>814</td>
<td>7181</td>
</tr>
<tr>
<td>scheme</td>
<td>1</td>
<td>2775</td>
<td>153</td>
<td>5.2%</td>
<td>328</td>
<td>3256</td>
</tr>
<tr>
<td>shell</td>
<td>5</td>
<td>2482</td>
<td>538</td>
<td>17.8%</td>
<td>328</td>
<td>3348</td>
</tr>
<tr>
<td>awk</td>
<td>7</td>
<td>1022</td>
<td>251</td>
<td>19.7%</td>
<td>187</td>
<td>1460</td>
</tr>
<tr>
<td>perl</td>
<td>1</td>
<td>772</td>
<td>205</td>
<td>21.0%</td>
<td>137</td>
<td>1114</td>
</tr>
<tr>
<td>haskell</td>
<td>48</td>
<td>149</td>
<td>0</td>
<td>0.0%</td>
<td>16</td>
<td>165</td>
</tr>
<tr>
<td>matlab</td>
<td>2</td>
<td>57</td>
<td>0</td>
<td>0.0%</td>
<td>7</td>
<td>64</td>
</tr>
<tr>
<td>Total:</td>
<td>28258</td>
<td>2242738</td>
<td>647591</td>
<td>22.4%</td>
<td>579484</td>
<td>346983</td>
</tr>
</tbody>
</table>

### Line Count of gcc-4.6.2-20101225/gcc

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Comment %</th>
<th>Blank</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>15638</td>
<td>1452351</td>
<td>319224</td>
<td>18.0%</td>
<td>321806</td>
<td>2093381</td>
</tr>
<tr>
<td>ada</td>
<td>4591</td>
<td>678321</td>
<td>315386</td>
<td>31.7%</td>
<td>233848</td>
<td>1227555</td>
</tr>
<tr>
<td>cpp</td>
<td>8527</td>
<td>248085</td>
<td>60722</td>
<td>19.7%</td>
<td>66383</td>
<td>375190</td>
</tr>
<tr>
<td>fortranfixed</td>
<td>2767</td>
<td>90244</td>
<td>1871</td>
<td>2.0%</td>
<td>12800</td>
<td>104915</td>
</tr>
<tr>
<td>assembler</td>
<td>138</td>
<td>31871</td>
<td>7506</td>
<td>19.1%</td>
<td>4882</td>
<td>44259</td>
</tr>
<tr>
<td>autconf</td>
<td>3</td>
<td>28604</td>
<td>12</td>
<td>0.0%</td>
<td>4011</td>
<td>32627</td>
</tr>
<tr>
<td>scheme</td>
<td>7</td>
<td>13731</td>
<td>1192</td>
<td>8.0%</td>
<td>1524</td>
<td>16447</td>
</tr>
<tr>
<td>objective&lt;</td>
<td>810</td>
<td>25860</td>
<td>4492</td>
<td>14.8%</td>
<td>7436</td>
<td>37788</td>
</tr>
<tr>
<td>shell</td>
<td>678</td>
<td>12713</td>
<td>2893</td>
<td>18.5%</td>
<td>1575</td>
<td>17181</td>
</tr>
<tr>
<td>awk</td>
<td>4</td>
<td>6124</td>
<td>1070</td>
<td>14.9%</td>
<td>893</td>
<td>8087</td>
</tr>
<tr>
<td>perl</td>
<td>1</td>
<td>5441</td>
<td>2835</td>
<td>34.3%</td>
<td>702</td>
<td>8978</td>
</tr>
<tr>
<td>haskell</td>
<td>5</td>
<td>2814</td>
<td>576</td>
<td>17.0%</td>
<td>378</td>
<td>768</td>
</tr>
<tr>
<td>matlab</td>
<td>1</td>
<td>512</td>
<td>0</td>
<td>0.0%</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total:</td>
<td>33338</td>
<td>2600787</td>
<td>718924</td>
<td>21.7%</td>
<td>657002</td>
<td>397671</td>
</tr>
</tbody>
</table>
Open Source and Free Software Development Model

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

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

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

A combination of the two seems more sensible

The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

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

The Architecture of GCC

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

#### Language Specific Code
- Compiler Generation Framework
- Language and Machine Independent Generic Code
- Machine Description

#### Machine Dependent Code
- Generated
- Target Name

#### Development Time
- Selected
- Copied

#### Build Time
- Generated
- Use Time

#### Source Program
- Assembly 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;
  }
  ```
- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows
  ```
  ftree-loop-distribution
  Common Report Var(flag_tree_loop_distribution) Optimization
  Enable loop distribution on trees
  ```
- The required C statements are generated during the build

---

### Another Example of The Generation Related Gap

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

<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>generatoma.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 cscope

<table>
<thead>
<tr>
<th>File</th>
<th>Line</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>genflags.c</td>
<td>250</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gengenrtl.c</td>
<td>350</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gengtype.c</td>
<td>3694</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genmdeps.c</td>
<td>45</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genmodes.c</td>
<td>1376</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genopinit.c</td>
<td>469</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genoutput.c</td>
<td>1023</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genpeep.c</td>
<td>353</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genpreds.c</td>
<td>1404</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genrecog.c</td>
<td>2722</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>lto-optimizer.c</td>
<td>412</td>
<td>main (int argc, char *argv[])</td>
</tr>
<tr>
<td>main.c</td>
<td>33</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>mips-tdump.c</td>
<td>1393</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>mips-tdump.c</td>
<td>655</td>
<td>main (void)</td>
</tr>
<tr>
<td>mips-tfile.c</td>
<td>4695</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>tlink.c</td>
<td>61</td>
<td>main (const char *main);</td>
</tr>
</tbody>
</table>
The generated compiler uses an adaptation of the Davison Fraser model:
- Generic expander and recognizer
- Machine specific information is isolated in data structures
- Generating a compiler involves generating these data structures

### The GCC Challenge: Poor Retargetability Mechanism

**Symptoms:**
- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code.
- 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>

- Machine descriptions are difficult to construct, understand, debug, and enhance.

### Meeting the GCC Challenge

**Goal of Understanding**

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

**Part 5**

Meeting the GCC Challenge
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

Customizing the Configuration and Build Process

- Creating only cc1
- Creating bare metal cross build
  Complete tool chain without OS support
- Creating cross build with OS support

Incremental Construction of Machine Descriptions

- Define different levels of source language
- Identify the minimal set of features in the target required to support each level
- Identify the minimal information required in the machine description to support each level
  - Successful compilation of any program, and
  - Correct execution of the generated assembly program
- Interesting observations
  - It is the increment in the source language which results in understandable increments in machine descriptions rather than the increment in the target architecture
  - If the levels are identified properly, the increments in machine descriptions are monotonic
Adding Passes to Massage IRs

- Understanding the pass structure
- Understanding the mechanisms of traversing a call graph and a control flow graph
- Understanding how to access the data structures of IRs
- Simple exercises such as:
  - Count the number of copy statements in a program
  - Count the number of variables declared “const” in the program
  - Count the number of occurrences of arithmetic operators in the program
  - Count the number of references to global variables in the program

### CS 715 Coverage

- **An external view of GCC:**
  - Configuration and building
  - Gray box probing of GCC
  - Introduction to Gimple and RTL IRs
  - Parallelization and vectorization in GCC
  - Introduction to GCC Machine Descriptions

- **An internal view of GCC:** Walking the maze of GCC source code
  - Control flow and plugin structure of the core compiler,
  - Mechanisms for hooking up front ends, IR passes, and back ends
  - Examining and manipulating Gimple and RTL IRs
  - Design and implementation of GDFA
  - Machine descriptions and retargetability mechanism

### CS 715 Philosophy

- **Twin goals:**
  - *Learning how to learn GCC*
    - Our focus will be on
      - giving you some core information
      - showing you how to discover more information
  - *Striking a balance between theory and practice*
    - Our focus will be on showing you how to
      - discover concepts in a large code base and build abstractions
      - take concepts and update a large code base
      - relate the class room concepts of compilers to an industry strength compiler
CS 715 Pedagogy

- Introductory lecture for each topic followed by lab work
- Many lecture hours will be used as lab hours
  - We will meet at one place (NSL? GRC?)
    You are expected to do ssh to your own machine
  - Class room as labs using your laptops?
- Tools
  shell, make, cscope, ctags, gdb/ddd, screen, spim (mips simulator)

CS 715 Lab Work

- Lab exercises:
  - Pre-defined experiments
  - Ungraded
- Assignments: Graded lab work
  - Implementation to modify gcc
  - Graded
  - To be submitted in about a couple of weeks or 10 days
- Projects
  - Specific Study + Implementation
  - Graded
  - To be submitted in about a couple of months
  - Possible topics
    Extensions of GDFA, MD rewriting with newer constructs, Detailing the retargetability mechanism, MD parsers, garbage collection, MELT, investigating specific optimizations (such as constant splitting)

CS 715 Assessment Scheme

- No written examination
- Marks for lab work

<table>
<thead>
<tr>
<th>Head</th>
<th>Number</th>
<th>Weightage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignments</td>
<td>4</td>
<td>4 × 15 = 60</td>
</tr>
<tr>
<td>Project</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

- Assignments need not be same for all students
  However, they will be comparable and students would have the choice of opting for a particular assignment