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

Conceptualisation Coding Compiling Linking Loading Execution

No. of unbound objects

30 June 2011
Implementation Mechanisms

Source Program

Translator

Target Program

Interpreter

Machine

Source Program

Input Data

Translator

Target Program

Machine

Implementation Mechanisms as “Bridges”

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

Program Specification

State : Variables
Operations: Expressions, Control Flow

Translation

Interpretation

Interpretation

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 = b<10?b:c;

Conditional jump

Condition

Fall through

False Part

True Part

Spim Assembly Equivalent

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

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

Spim Assembly Equivalent

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

Implementation Mechanisms

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

```
Translation Instructions \(\rightarrow\) Equivalent Instructions
Interpretation Instructions \(\rightarrow\) Actions Implied by Instructions
```

Language Implementation Models

**Analysis** \(\rightarrow\) **Synthesis** \(\rightarrow\) **Compilation**

**Analysis** \(\rightarrow\) **Execution** \(\rightarrow\) **Interpretation**

Language Processor Models

- **C, C++**
- **Java, C#**
An Overview of Compilation Phases

The Structure of a Simple Compiler

Front End
- Parser
- AST
- Instruction Selector
- Register Allocator
- Assembly Emitter

Back End
- Insn
- Assembly Program

Translation Sequence in Our Compiler: Parsing

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

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

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

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
Overview: An Overview of Compilation Phases

Translation Sequence in Our Compiler: IR Generation

**Issues:**
- Convert to maximal trees which can be implemented without altering control flow
  - Simplifies instruction selection and scheduling, register allocation etc.
- Linearise control flow by flattening nested control constructs

Translation Sequence in Our Compiler: Instruction Selection

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

Translation Sequence in Our Compiler: Emitting Instructions

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

Part 3

Compilation Models
### Compilation Models

**Aho Ullman Model**
- Front End
- AST
- Optimizer
- Target Indep. IR
- Code Generator
- Target Program

**Davidson Fraser Model**
- Front End
- AST
- Expander
- Register Transfers
- Optimizer
- Register Transfers
- Recognizer
- Target Program

### Essential Abstractions in GCC

**Typical Front Ends**
- Source Program
- Tokens
- Parse Tree
- Symtab Handler
- Error Handler

- Parser
- AST or Linear IR
- Symbol Table

- Scanner
- AST
- Semantic Analyzer

### Typical Back Ends in Aho Ullman Model

- m/c Ind. IR
- m/c Ind. Optimizer
- Code Generator
- m/c Dep. IR

- Register Allocator
- Instruction Scheduler
- Peephole Optimizer
- Assembly Code

**Aho Ullman: Instruction selection**
- over optimized IR using
- cost based tree pattern matching

**Davidson Fraser: Instruction selection**
- over AST using
- structural tree pattern matching
- naive code which is
  - target dependent, and is
  - optimized subsequently
Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th>Instruction Selection</th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine independent IR is expressed in the form of trees</td>
<td>Machine instructions are described in the form of trees</td>
<td>Trees in the IR are &quot;covered&quot; using the instruction trees</td>
</tr>
<tr>
<td>Cost based tree pattern matching</td>
<td>Structural tree pattern matching</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine independent</td>
<td>Key Insight: Register transfers are target specific but their form is target independent</td>
<td></td>
</tr>
</tbody>
</table>

Greedy Property

Essential Abstractions in GCC

What is GCC?

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

The GNU Tool Chain for C

Source Program

```
cc1
CPP
as
GCC
ld
```

Target Program
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

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

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, TGCC (m68k variant), Z8000, PIC24/dsPIC, NEC SX architecture
### Comprehensive of GCC: Size

- **Overall size**
  - **Subdirectories**
    - gcc-4.4.2: 3794
    - gcc-4.5.0: 4056
    - gcc-4.6.0: 4383
  - **Files**
    - gcc-4.4.2: 62301
    - gcc-4.5.0: 65639
    - gcc-4.6.0: 71096

- **Core size (src/gcc)**
  - **Subdirectories**
    - gcc-4.4.2: 257
    - gcc-4.5.0: 283
    - gcc-4.6.0: 336
  - **Files**
    - gcc-4.4.2: 30163
    - gcc-4.5.0: 32723
    - gcc-4.6.0: 36503

- **Machine Descriptions (src/gcc/config)**
  - **Subdirectories**
    - .c files: 36
    - .h files: 426
    - .md files: 206
  - Total files: 69739

---

### Line Count of gcc-4.4.2

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>15638</td>
<td>1840245</td>
<td>394582</td>
<td>17.7%</td>
</tr>
<tr>
<td>cpp</td>
<td>58522</td>
<td>873755</td>
<td>160474</td>
<td>17.1%</td>
</tr>
<tr>
<td>java</td>
<td>6342</td>
<td>681659</td>
<td>643045</td>
<td>48.5%</td>
</tr>
<tr>
<td>ada</td>
<td>4306</td>
<td>638557</td>
<td>294881</td>
<td>31.0%</td>
</tr>
<tr>
<td>autoconf</td>
<td>76</td>
<td>449046</td>
<td>393</td>
<td>0.1%</td>
</tr>
<tr>
<td>make</td>
<td>38</td>
<td>110604</td>
<td>22900</td>
<td>0.9%</td>
</tr>
<tr>
<td>html</td>
<td>480</td>
<td>103808</td>
<td>5096</td>
<td>0.2%</td>
</tr>
<tr>
<td>fortran90</td>
<td>2164</td>
<td>73156</td>
<td>16770</td>
<td>2.1%</td>
</tr>
<tr>
<td>assembler</td>
<td>138</td>
<td>42360</td>
<td>9607</td>
<td>1.8%</td>
</tr>
<tr>
<td>shell</td>
<td>137</td>
<td>39347</td>
<td>8532</td>
<td>1.8%</td>
</tr>
<tr>
<td>fortran90</td>
<td>660</td>
<td>11052</td>
<td>2582</td>
<td>1.7%</td>
</tr>
<tr>
<td>object.p</td>
<td>395</td>
<td>10967</td>
<td>1706</td>
<td>1.4%</td>
</tr>
<tr>
<td>automake</td>
<td>63</td>
<td>60148</td>
<td>2853</td>
<td>12.5%</td>
</tr>
<tr>
<td>perl</td>
<td>24</td>
<td>4114</td>
<td>1138</td>
<td>21.7%</td>
</tr>
<tr>
<td>scheme</td>
<td>1</td>
<td>2773</td>
<td>153</td>
<td>5.2%</td>
</tr>
<tr>
<td>cdb</td>
<td>9</td>
<td>878</td>
<td>506</td>
<td>8.5%</td>
</tr>
<tr>
<td>python</td>
<td>6</td>
<td>1139</td>
<td>21</td>
<td>15.7%</td>
</tr>
<tr>
<td>awk</td>
<td>9</td>
<td>1127</td>
<td>324</td>
<td>22.3%</td>
</tr>
<tr>
<td>pascal</td>
<td>4</td>
<td>1046</td>
<td>141</td>
<td>11.9%</td>
</tr>
<tr>
<td>csharp</td>
<td>5</td>
<td>2402</td>
<td>526</td>
<td>17.8%</td>
</tr>
<tr>
<td>dcl</td>
<td>2</td>
<td>497</td>
<td>99</td>
<td>16.6%</td>
</tr>
<tr>
<td>tcl</td>
<td>1</td>
<td>392</td>
<td>113</td>
<td>22.4%</td>
</tr>
<tr>
<td>haskell</td>
<td>45</td>
<td>149</td>
<td>2</td>
<td>15.8%</td>
</tr>
<tr>
<td>emacslisp</td>
<td>1</td>
<td>59</td>
<td>21</td>
<td>26.2%</td>
</tr>
<tr>
<td>matlab</td>
<td>2</td>
<td>57</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Total</td>
<td>30163</td>
<td>66119</td>
<td>155790</td>
<td>24.1%</td>
</tr>
</tbody>
</table>

### Line Count of gcc-4.4.0

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>19328</td>
<td>210037</td>
<td>444743</td>
<td>17.5%</td>
</tr>
<tr>
<td>cpp</td>
<td>75502</td>
<td>918870</td>
<td>229541</td>
<td>18.9%</td>
</tr>
<tr>
<td>java</td>
<td>5545</td>
<td>682135</td>
<td>645950</td>
<td>31.7%</td>
</tr>
<tr>
<td>ada</td>
<td>4112</td>
<td>664372</td>
<td>323200</td>
<td>48.3%</td>
</tr>
<tr>
<td>autoconf</td>
<td>79</td>
<td>308968</td>
<td>422</td>
<td>0.1%</td>
</tr>
<tr>
<td>html</td>
<td>481</td>
<td>144559</td>
<td>5681</td>
<td>3.8%</td>
</tr>
<tr>
<td>make</td>
<td>138</td>
<td>114590</td>
<td>2435</td>
<td>2.1%</td>
</tr>
<tr>
<td>fortran90</td>
<td>2535</td>
<td>88005</td>
<td>1611</td>
<td>2.1%</td>
</tr>
<tr>
<td>assembler</td>
<td>197</td>
<td>45098</td>
<td>10062</td>
<td>18.3%</td>
</tr>
<tr>
<td>shell</td>
<td>136</td>
<td>39709</td>
<td>5534</td>
<td>10.4%</td>
</tr>
<tr>
<td>scheme</td>
<td>7</td>
<td>13725</td>
<td>1192</td>
<td>8.6%</td>
</tr>
<tr>
<td>fortran90</td>
<td>700</td>
<td>12955</td>
<td>2888</td>
<td>18.3%</td>
</tr>
<tr>
<td>object.p</td>
<td>396</td>
<td>10783</td>
<td>1833</td>
<td>14.5%</td>
</tr>
<tr>
<td>automake</td>
<td>54</td>
<td>5298</td>
<td>194</td>
<td>12.5%</td>
</tr>
<tr>
<td>perl</td>
<td>25</td>
<td>4144</td>
<td>1139</td>
<td>21.0%</td>
</tr>
<tr>
<td>awk</td>
<td>50</td>
<td>2800</td>
<td>436</td>
<td>13.5%</td>
</tr>
<tr>
<td>python</td>
<td>5</td>
<td>2084</td>
<td>135</td>
<td>17.1%</td>
</tr>
<tr>
<td>cdb</td>
<td>9</td>
<td>879</td>
<td>506</td>
<td>8.5%</td>
</tr>
<tr>
<td>dcl</td>
<td>2</td>
<td>402</td>
<td>84</td>
<td>17.3%</td>
</tr>
<tr>
<td>tcl</td>
<td>1</td>
<td>392</td>
<td>113</td>
<td>22.4%</td>
</tr>
<tr>
<td>haskell</td>
<td>45</td>
<td>149</td>
<td>2</td>
<td>15.8%</td>
</tr>
<tr>
<td>emacslisp</td>
<td>1</td>
<td>59</td>
<td>21</td>
<td>26.2%</td>
</tr>
<tr>
<td>matlab</td>
<td>2</td>
<td>57</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Total</td>
<td>50126</td>
<td>114881</td>
<td>557150</td>
<td>24.1%</td>
</tr>
</tbody>
</table>

---

Overview: GCC ≡ The Great Compiler Challenge
## Essential Abstractions in GCC

### GCC Resource Center, IIT Bombay

<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>78588</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>autoconf</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>17.0%</td>
<td>1314</td>
<td>14588</td>
</tr>
<tr>
<td>objective_c</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>ocaml</td>
<td>5</td>
<td>2482</td>
<td>538</td>
<td>17.8%</td>
<td>328</td>
<td>3348</td>
</tr>
<tr>
<td>shell</td>
<td>16</td>
<td>2256</td>
<td>712</td>
<td>24.0%</td>
<td>374</td>
<td>3342</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>346981</td>
</tr>
</tbody>
</table>

### Why is Understanding GCC 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
      - The target specification and the generated compiler
      - Input-output of the generated compiler
    - A source program and the generated assembly program
- GCC generated compiler uses a derivative of the Davidson-Fraser model of compilation
  - Early instruction selection
  - Machine dependent intermediate representation
  - Simplistic instruction selection and retargatibility mechanism

### Why is Understanding GCC 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
      - The target specification and the generated compiler
      - Input-output of the generated compiler
    - A source program and the generated assembly program
- GCC generated compiler uses a derivative of the Davidson-Fraser model of compilation
  - Early instruction selection
  - Machine dependent intermediate representation
  - Simplistic instruction selection and retargatibility mechanism
The Architecture of GCC

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

Compiler Generation Framework

Parser
Gimplifier
Tree SSA Optimizer
RTL Generator
Optimizer
Code Generator

Source Program
Generated Compiler (cc1)
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-4.6.0/gcc` using `cscope -R`
  - 7027 occurrences!
- What if we do not search recursively?
Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.6.0/gcc using cscope

```
File Line
0 collect2.c 1076 main (int argc, char **argv)
1 fp-test.c 85 main (void )
2 gcc.c 6092 main (int argc, char **argv)
3 gcov-dump.c 76 main (int argc ATTRIBUTE_UNUSED, char **argv)
4 gcov-iov.c 29 main (int argc, char **argv)
5 gcov.c 360 main (int argc, char **argv)
6 genattr.c 164 main (int argc, char **argv)
7 genattrtab.c 4820 main (int argc, char **argv)
8 genautomata.c 9459 main (int argc, char **argv)
9 genchecksum.c 97 main (int argc, char ** argv)
 a gencodes.c 51 main (int argc, char **argv)
b gencondition.c 209 main (int argc, char **argv)
c genconfig.c 261 main (int argc, char **argv)
d genconstants.c 79 main (int argc, char **argv)
e genemit.c 830 main (int argc, char **argv)
f genenums.c 48 main (int argc, char **argv)
```

GCC Retargetability Mechanism

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

Symptoms:
- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
- Size in terms of line counts in gcc-4.6.0

<table>
<thead>
<tr>
<th>Files</th>
<th>i386</th>
<th>mips</th>
<th>arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.md</td>
<td>38722</td>
<td>15534</td>
<td>30938</td>
</tr>
<tr>
<td>*.c</td>
<td>39579</td>
<td>16766</td>
<td>26164</td>
</tr>
<tr>
<td>*.h</td>
<td>17869</td>
<td>5667</td>
<td>18711</td>
</tr>
<tr>
<td>Total</td>
<td>96170</td>
<td>37969</td>
<td>75913</td>
</tr>
</tbody>
</table>
## Meeting the GCC Challenge

### Workshop Coverage

<table>
<thead>
<tr>
<th>External View</th>
<th>Internal View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine descriptions</td>
<td>Front end hooks</td>
</tr>
<tr>
<td>Configuration and building</td>
<td>Retargetability mechanism</td>
</tr>
<tr>
<td>Gray box probing</td>
<td>Pass structure</td>
</tr>
<tr>
<td>Parallelization and Vectorization</td>
<td>Static and dynamic plugin mechanisms</td>
</tr>
</tbody>
</table>

---

<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 No No</td>
</tr>
<tr>
<td>Build process</td>
<td>Customizing the configuration and building</td>
<td>Yes No No</td>
</tr>
<tr>
<td>Retargetability issues and machine descriptions</td>
<td>Incremental construction of machine descriptions</td>
<td>No No Yes</td>
</tr>
<tr>
<td>IR data structures and access mechanisms</td>
<td>Adding passes to massage IRs</td>
<td>No Yes Yes</td>
</tr>
<tr>
<td>Retargetability mechanism</td>
<td>Yes Yes Yes</td>
<td></td>
</tr>
</tbody>
</table>