First Level Gray Box Probing

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
• Conclusions
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  \[\xleftarrow{\text{Parse}}\xrightarrow{\text{Simplify}}\xrightarrow{\text{Optimize}}\xrightarrow{\text{Tree SSA}}\xrightarrow{\text{Generate RTL}}\xrightarrow{\text{Optimize RTL}}\xrightarrow{\text{Generate ASM}}\xrightarrow{\text{RTL}}\xrightarrow{\text{ASM}}\xrightarrow{\text{GIMPLE}}\xrightarrow{\text{RTL}}\xrightarrow{\text{GIMPLE Passes}}\xrightarrow{\text{RTL Passes}}\]
Transformation Passes in GCC 4.6.0

- A total of 207 unique pass names initialized in
  ${SOURCE}/gcc/passes.c
  Total number of passes is 241.
  - 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

<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>10</td>
</tr>
<tr>
<td>Simple Interprocedural</td>
<td>Conditional Constant Propagation, Inlining, SSA Construction</td>
<td>38</td>
</tr>
<tr>
<td>Passes (Non-LTO)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regular Interprocedural</td>
<td>Constant Propagation, Inlining, Pointer Analysis</td>
<td>10</td>
</tr>
<tr>
<td>Passes (LTO)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LTO generation passes</td>
<td></td>
<td>02</td>
</tr>
<tr>
<td>Other Intraprocedural</td>
<td>Constant Propagation, Dead Code Elimination, PRE Value Range Propagation, Rename SSA</td>
<td>65</td>
</tr>
<tr>
<td>Optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Vectorization, Parallelization, Copy Propagation, Dead Code Elimination</td>
<td>28</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>
</tbody>
</table>
### Passes On RTL in GCC 4.6.0

<table>
<thead>
<tr>
<th>Pass Group</th>
<th>Examples</th>
<th>Number of passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intraprocedural Optimizations</td>
<td>CSE, Jump Optimization, Dead Code Elimination, Jump Optimization</td>
<td>27</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Loop Invariant Movement, Peeling, Unswitching</td>
<td>07</td>
</tr>
<tr>
<td>Machine Dependent Optimizations</td>
<td>Register Allocation, Instruction Scheduling, Peephole Optimizations</td>
<td>50</td>
</tr>
<tr>
<td>Assembly Emission and Finishing</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td><strong>Total number of passes on RTL</strong></td>
<td></td>
<td><strong>87</strong></td>
</tr>
</tbody>
</table>

**Notes**

Finding Out List of Optimizations

Along with the associated flags

- 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 5)
  
  `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

### 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>158</td>
<td></td>
</tr>
<tr>
<td>O3</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>Os</td>
<td>156</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
- 009t.omplower
- 010t.lower
- 012t.eh
- 013t.cfg
- 017t.ssa
- 018t.veclower
- 019t.inline_param1
- 020t.einline
- 037t.release_ssa
- 038t.inline_param2
- 044i.whole-program
- 048i.inline

IPA dumps (i)
- 000i.cgraph
- 014i.visibility
- 015i.early_local_cleanups
- 044i.whole-program
- 048i.inline

RTL dumps (r)
- 144r.expand
- 145r.sibling
- 147r.initvals
- 148r.unshare
- 149r.vregs
- 150r.into_cfglayout
- 151r.jump

138t.cplxlower
143t.optimized
224t.statistics
ipa dumps (i)
000i.cgraph
014i.visibility
015i.early_local_cleanups
044i.whole-program
048i.inline

163r.reginfo
183r.outof_cfglayout
184r.split
186r.dfini
187r.mode_sw
188r.asmcons
191r.ira
194r.split2
198r.pro_and_epilogue
211r.stack
212r.alignments
215r.mach
216r.barriers
220r.shorten
221r.nothrow
222r.final
223r.dfinish
assembly

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay

Passes for First Level Graybox Probing of GCC

C Source Code
Parser
AST
Gimplifier
GIMPLE
CFG Generator
RTL Generator
RTL expand

Reg Allocator
ira
pro_epilogue_generation
prologue-epilogue
Pattern Matcher
ASM Program

Lowering of abstraction!
Part 2

Examining GIMPLE Dumps

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

```c
int a;

int main()
{
    int x = 10;
    int y = 5;
    x = a + x * y;
    y = y - a * x;
}
```

Global variables are treated as "memory locations" and local variables are treated as "registers".

---

GIMPLE: 1-D Array Accesses

```c
int main()
{
    int a[3], x;
    x = a[1] + a[2];
    a[0] = a[1] + a[1]*x;
}
```
GIMPLE: 2-D Array Accesses

```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];
}
```

- No notion of "addressable memory" in GIMPLE.
- Array reference is a single operation in GIMPLE and is linearized in RTL during expansion.

GIMPLE: Use of Pointers

```c
int main()
{
    int * D.1953; int * * a; int * b; int c;
    b = &c;
    a = &b;
    **a = 10; /* c = 10 */
}
```

Notes
### GIMPLE: Use of Structures

```c
typedef struct address { char *name; } ad;
typedef struct student { int roll; ad *ct; } st;

int main()
{
    st *s;
s = malloc(sizeof(st));
s->roll = 1;
s->ct = malloc(sizeof(ad));
s->ct->name = "Mumbai";
}
```

### Notes

```c
int main()
{
    int *p_a, a[3];
p_a = &a[0];
    *p_a = 10;
    *(p_a+1) = 20;
    *(p_a+2) = 30;
}
```
```c
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
    if (a<=12) goto <D.1197>;
    else goto <D.1198>;
    <D.1197>:
        a = a + 1;
    <D.1198>:
    if (a <= 7) goto <D.1196>;
    else goto <D.1198>;
    <D.1196>:
    a = a + 1;
    <D.1197>:
        if (a <= 7) goto <D.1196>;
    else goto <D.1198>;
    <D.1198>:
        a = a+b+c;
    }
```
Control Flow Graph: Textual View

```
test.c.004t.gimple
if (a <= 12) goto <D.1200>; else goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

```
test.c.013t.cfg

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

test.c.013t.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;

False  True

1 July 2011    Graybox Probing-I: Examining GIMPLE Dumps

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
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: Call Graphs for Recursive Functions

```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");
}
```

---

Inspection GIMPLE When in Doubt (1)

```c
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>x</th>
<th>y</th>
<th>(y + x)</th>
<th>(y + x) + y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
int x=2, y=3;
int 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>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</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>7</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

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

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

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

\[
\begin{array}{c|c}
x & 2 \\
y & 3 \\
x = x + 1; /* 3 */ \\
D.1572 = y + x; /* 6 */ \\
y = y + 1; /* 4 */ \\
x = D.1572 + y; /* 10 */ \\
y = y + 1; /* 5 */ \\
\end{array}
\]
• How is \( a[i] = i++ \) handled? This is an undefined behaviour as per C standards.

• What is the order of parameter evaluation? For a call \( f(getX(), getY()) \), is the order left to right? arbitrary? Is the evaluation order in GCC consistent?

• Understanding complicated declarations in C can be difficult What does the following declaration mean:

\[
\text{int} * ((\text{*MYVAR}) (\text{int})) [10];
\]

Hint: Use \(-fdump-tree-original-raw-verbose\) option. The dump to see is 003t.original
RTL for i386: Arithmetic Operations (1)

Translation of \( a = a + 1 \)

Dump file: test.c.144r.expand

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

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
RTL for i386: Arithmetic Operations (1)

Translation of \( a = a + 1 \)

**Dump file:** test.c.144r.expand

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

\( a \) is a local variable allocated on stack
RTL for i386: Arithmetic Operations (1)

Translation of \( a = a + 1 \)

**Dump file:** test.c.144r.expand

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

} t.c:24 -1 (nil)

\[\text{parallel} \quad \text{set} \quad \text{clobber} \quad \text{reg:CC}\]

\text{Essential Abstractions in GCC}

GCC Resource Center, IIT Bombay
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

**Dump file:** test.c.144r.expand

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

Output with slim suffix:

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

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

(insn 12 11 13 4 (parallel [
  (set (mem/c/i:SI
       (plus:SI
         (reg/f:SI 54 virtual-stack-vars)
         (const_int -4 [0xffffffffc])) [0 a+0 S4 A32])
       (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))
]) t.c:24 -1 (nil))
Additional Information in RTL

(insn 12 11 13 4 (parallel [
  (set (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))
]) t.c:24 -1 (nil))

Notes

Scalar that is not a part of an aggregate

Memory reference that does not trap
Additional Information in RTL

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

1 July 2011  
Graybox Probing-I: Examining RTL Dumps

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay

Notes
RTL for i386: Arithmetic Operations (2)

Translation of \( a = a + 1 \) when \( a \) is a global variable

**Dump file:** test.c.144r.expand

```
(insn 11 10 12 4 (set
  (reg:SI 64 [ a.0 ])
  (mem/c/i:SI (symbol_ref:SI ("a")
    <var_decl 0xb7d8d000 a>) [0 a+0 S4 A32])) t.c:26 -1 (nil))

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

(insn 13 12 14 4 (set
  (mem/c/i:SI (symbol_ref:SI ("a")
    <var_decl 0xb7d8d000 a>) [0 a+0 S4 A32])
  (reg:SI 63 [ a.1 ])) t.c:26 -1 (nil))
```

Load a into reg64

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
RTL for i386: Arithmetic Operations (2)

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

**Dump file:** test.c.144r.expand

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

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

(inan 13 12 14 4 (set
  (mem/c/i:SI (symbol_ref:SI ("a")
    <var_decl 0xb7d8d000 a>) [0 a+0 S4 A32])
  (reg:SI 63 [ a.1 ])) t.c:26 -1 (nil))
```

Load a into reg64

$\text{reg63 = reg64 + 1}$

store reg63 into a
Translation of $a = a + 1$ when $a$ is a global variable

**Dump file:** test.c.144r.expand

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

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

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

**Dump file:** test.c.144r.expand

```
(insn 10 9 11 4 (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))
 ])
)
```

Load a into reg64
reg63 = reg64 + 1
store reg63 into a

Output with slim suffix
r64:SI=[‘a’]
{r63:SI=r64:SI+0x1;
clobber flags:CC;
}{’a’}=r63:SI
RTL for i386: Arithmetic Operations (3)

Translation of \( a = a + 1 \) when \( a \) is a formal parameter

**Dump file:** test.c.144r.expand

```c
(intn 10 9 11 4 (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]))
            )
            (clobber (reg:CC 17 flags))
         ))
   ) t1.c:25 -1 (nil))
```

Access through argument pointer register instead of frame pointer register
No offset required?

Output with slim suffix

\{\[r53:SI]=[r53:SI]+0x1;\}
\{clobber flags:CC;\}
RTL for i386: Arithmetic Operation (4)

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

**Dump file:** test.c.144r.expand

```assembly
(insn 10 9 11 4 (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))
    )]
   )) t1.c:25 -1 (nil))
```

Offset 4 added to the argument pointer register
When $a$ is the first parameter, its offset is 0!
Output with slim suffix

```
{[r53:SI+0x4]=[r53:SI+0x4]+0x1;
clobber flags:CC;
}
```
**RTL for spim: Arithmetic Operations**

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

**Dump file:** test.c.144r.expand

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

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

**Notes**

**RTL for i386: Control Flow**

What does this represent?

```plaintext
pc = r17 <0 ? label(12) : pc
```

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
RTL for i386: Control Flow

Translation of

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

**Dump file:** test.c.144r.expand

```assembly
(insn 8 7 9 (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])) test.c:7 -1 (nil))
```

```assembly
(insn 9 8 10 (set (reg:CCGC 17 flags)
    (compare:CCGC (reg:SI 61)
        (const_int -4 [0xffffffffc])) [0 b+0 S4 A32]))) test.c:7 -1 (nil))
```

```assembly
(jump_insn 10 9 0 (set (pc)
    (if_then_else (le (reg:CCGC 17 flags)
        (const_int 0 [0x0])
        (label_ref 13)
        (pc))) test.c:7 -1 (nil)
    -> 13)
```

---

Observing Register Allocation for i386

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

```assembly
(test.c.188r.asmcons
(observable dump before register allocation)
(insn 10 9 11 3 (set (reg:SI 59)
    (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xffffffffc])) [0 a+0 S4 A32])) 44 *movsi
```

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

```assembly
(test.c.188r.asmcons
(observable dump before register allocation)
(insn 10 9 11 3 (set (reg:SI 59)
    (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xffffffffc])) [0 a+0 S4 A32])) 44 *movsi
```

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

```assembly
(test.c.188r.asmcons
(observable dump before register allocation)
(insn 10 9 11 3 (set (reg:SI 59)
    (mem/c/i:SI (plus:SI (reg/f:SI 20 frame)
        (const_int -4 [0xffffffffc])) [0 a+0 S4 A32])) 44 *movsi
```
Observing Register Allocation for i386

test.c.188r.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]))
### 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>

---

**Essential Abstractions in GCC**

**GCC Resource Center, IIT Bombay**

---

**Prologue and Epilogue: spim**

**Dump file:** test.c.197r.pro_and_epilogue

```assembly
(insn 17 3 18 2
  (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8])
   (reg:SI 31 $ra)) test.c:2 -1 (nil))

(insn 18 17 19 2
  (set (mem:SI (plus:SI (reg/f:SI 29 $sp) [const_int -4 [...]]) [...])
   (reg/f:SI 29 $sp)) test.c:2 -1 (nil))

(insn 19 18 20 2 (set
  (mem:SI (plus:SI (reg/f:SI 29 $sp) [const_int -8 [...]]) [...])
   (reg/f:SI 30 $fp)) test.c:2 -1 (nil))

(insn 20 19 21 2 (set (reg/f:SI 30 $fp)
  (reg/f:SI 29 $sp)) -1 (nil))

(insn 21 20 22 2 (set (reg/f:SI 29 $sp)
  (plus:SI (reg/f:SI 30 $fp)
   (const_int -32 [...]))) test.c:2 -1 (nil))
```

---

**Essential Abstractions in GCC**

**GCC Resource Center, IIT Bombay**

---
Part 4

Examining Assembly Dumps

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
    addl %edx, %eax
    addl -12(%ebp), %eax
    movl %eax, -4(%ebp)
.L6:

while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
1 July 2011
Graybox Probing-I: Examining Assembly Dumps
42/43

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
 addl %edx, %eax
 addl -12(%ebp), %eax
 movl %eax, -4(%ebp)

.L6:

while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
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
    addl %edx, %eax
    addl -12(%ebp), %eax
    movl %eax, -4(%ebp)
.L6:

while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
• Source code is transformed into assembly by lowering the abstraction level step by step to bring it close to the machine

• 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