

Workshop on Essential Abstractions in GCC

First Level Gray Box Probing

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay



1 July 2011

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



Part 1

Preliminaries

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



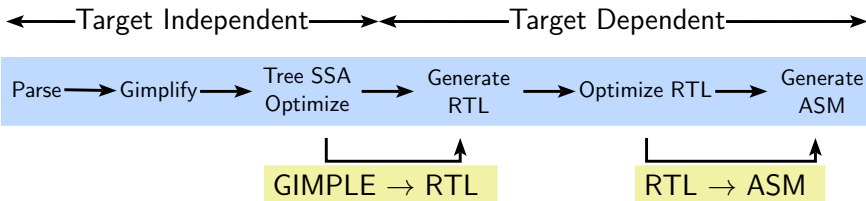
First Level Gray Box Probing of GCC

- Restricted to the most important translations in GCC



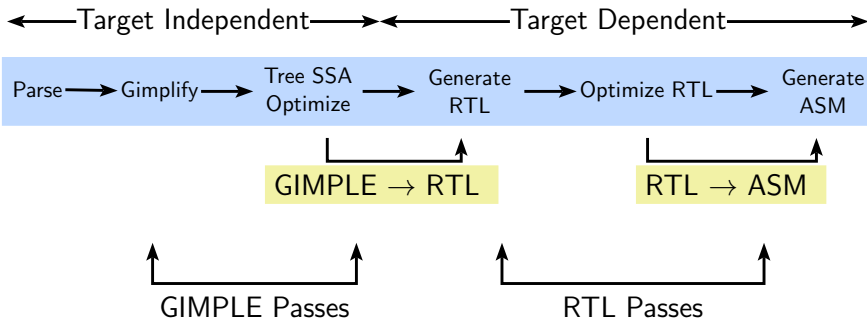
Basic Transformations in GCC

Transformation from a language to a *different* language



Basic Transformations in GCC

Transformation from a language to a *different* language



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



Passes On GIMPLE in GCC 4.6.0

Pass Group	Examples	Number of passes
Lowering	GIMPLE IR, CFG Construction	10
Simple Interprocedural Passes (Non-LTO)	Conditional Constant Propagation, Inlining, SSA Construction	38
Regular Interprocedural Passes (LTO)	Constant Propagation, Inlining, Pointer Analysis	10
LTO generation passes		02
Other Intraprocedural Optimizations	Constant Propagation, Dead Code Elimination, PRE Value Range Propagation, Rename SSA	65
Loop Optimizations	Vectorization, Parallelization, Copy Propagation, Dead Code Elimination	28
Generating RTL		01
<i>Total number of passes on GIMPLE</i>		154



Passes On RTL in GCC 4.6.0

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



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 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



Total Number of Dumps

Optimization Level	Number of Dumps	Goals
Default	47	Fast compilation
O1	134	
O2	158	
O3	168	
O _s	156	Optimize for space



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

138t.cplxlower0

143t.optimized

224t.statistics

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

163r.reginfo

183r.outof_cfglayout

184r.split1

186r.dfinit

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

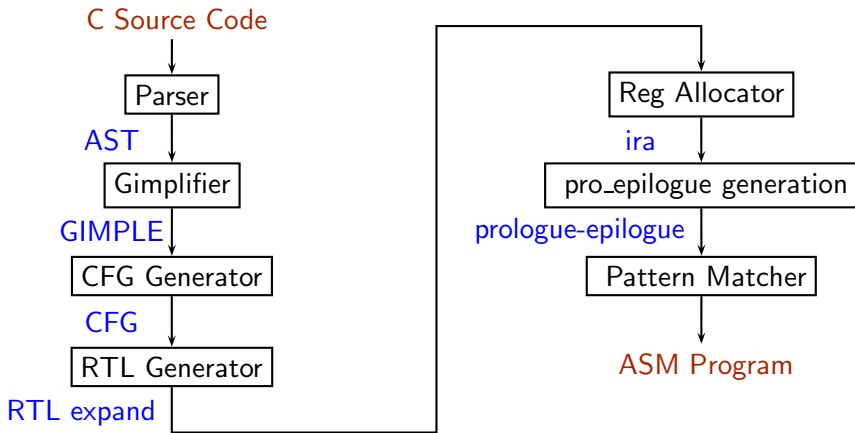
222r.final

223r.dfinish

assembly



Passes for First Level Graybox Probing of GCC



Lowering of abstraction!



Part 2

Examining GIMPLE Dumps

Gimplifier

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



GIMPLE: Composite Expressions Involving Local and Global Variables

test.c

```
int a;

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

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

test.c.004t.gimple

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

Global variables are treated as “memory locations” and local variables are treated as “registers”



GIMPLE: Composite Expressions Involving Local and Global Variables

```
test.c
```

```
int a;
```

```
int main()
```

```
{
```

```
    int x = 10;
```

```
    int y = 5;
```

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

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

```
}
```

```
test.c.004t.gimple
```

```
x = 10;
```

```
y = 5;
```

```
D.1954 = x * y;
```

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

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

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

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

```
y = y - D.1957;
```

Global variables are treated as “memory locations” and local variables are treated as “registers”



GIMPLE: Composite Expressions Involving Local and Global Variables

```
test.c
```

```
int a;
```

```
int main()
```

```
{
```

```
  int x = 10;
```

```
  int y = 5;
```

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

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

```
}
```

```
test.c.004t.gimple
```

```
x = 10;
```

```
y = 5;
```

```
D.1954 = x * y;
```

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

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

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

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

```
y = y - D.1957;
```

Global variables are treated as “memory locations” and local variables are treated as “registers”



GIMPLE: Composite Expressions Involving Local and Global Variables

```
test.c
```

```
int a;
```

```
int main()
```

```
{
```

```
    int x = 10;
```

```
    int y = 5;
```

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

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

```
}
```

```
test.c.004t.gimple
```

```
x = 10;
```

```
y = 5;
```

```
D.1954 = x * y;
```

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

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

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

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

```
y = y - D.1957;
```

Global variables are treated as “memory locations” and local variables are treated as “registers”



GIMPLE: 1-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 1-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 1-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 1-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 1-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 2-D Array Accesses

test.c

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

test.c.004t.gimple

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



GIMPLE: 2-D Array Accesses

test.c

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

test.c.004t.gimple

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

- 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

test.c

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

test.c.004t.gimple

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

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



GIMPLE: Use of Pointers

test.c

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

test.c.004t.gimple

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

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



GIMPLE: Use of Structures

test.c

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

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```



GIMPLE: Use of Structures

test.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";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```



GIMPLE: Use of Structures

test.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";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```



GIMPLE: Use of Structures

test.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";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```



GIMPLE: Pointer to Array

test.c

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

    p_a = &a[0];

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

test.c.004t.gimple

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

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



GIMPLE: Pointer to Array

test.c

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

    p_a = &a[0];

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

test.c.004t.gimple

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

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



GIMPLE: Pointer to Array

test.c

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

    p_a = &a[0];

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

test.c.004t.gimple

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

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



GIMPLE: Pointer to Array

test.c

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

    p_a = &a[0];

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

test.c.004t.gimple

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

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



GIMPLE: Translation of Conditional Statements

test.c

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

test.c.004t.gimple

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



GIMPLE: Translation of Conditional Statements

test.c

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

test.c.004t.gimple

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



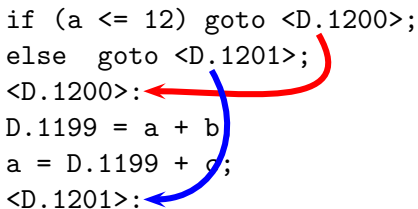
GIMPLE: Translation of Conditional Statements

test.c

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

test.c.004t.gimple

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



GIMPLE: Translation of Loops

test.c

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

test.c.004t.gimple

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




GIMPLE: Translation of Loops

test.c

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

test.c.004t.gimple

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



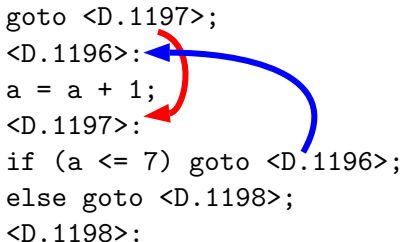
GIMPLE: Translation of Loops

test.c

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

test.c.004t.gimple

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



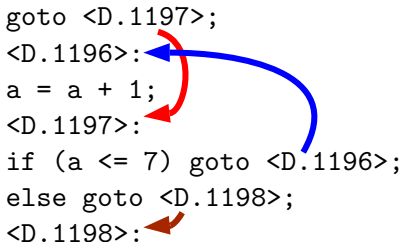
GIMPLE: Translation of Loops

test.c

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

test.c.004t.gimple

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



Control Flow Graph: Textual View

test.c.004t.gimple

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

test.c.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: 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: 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: 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: 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>;

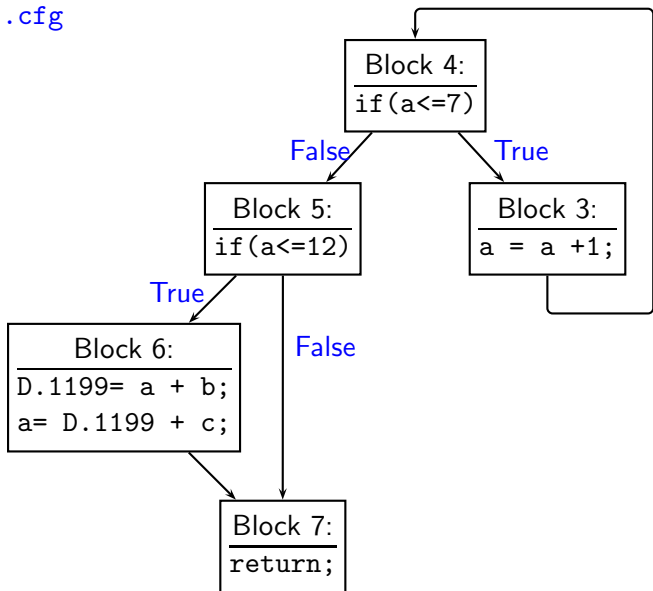
<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```



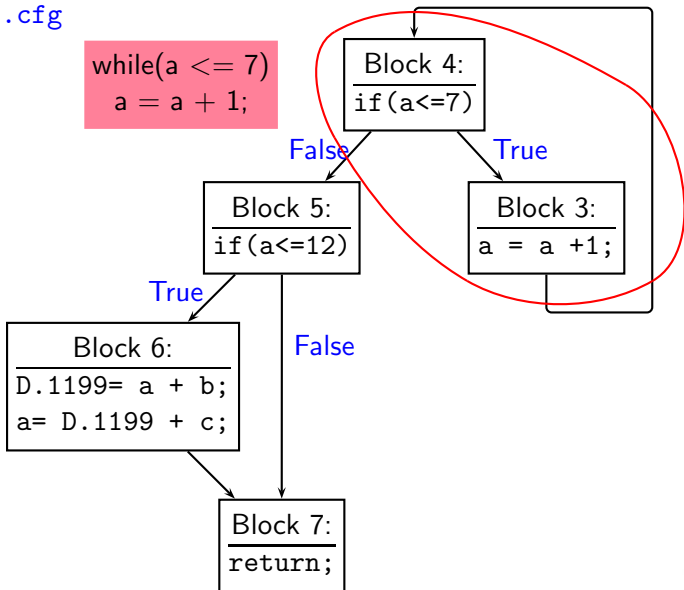
Control Flow Graph: Pictorial View

test.c.013t.cfg



Control Flow Graph: Pictorial View

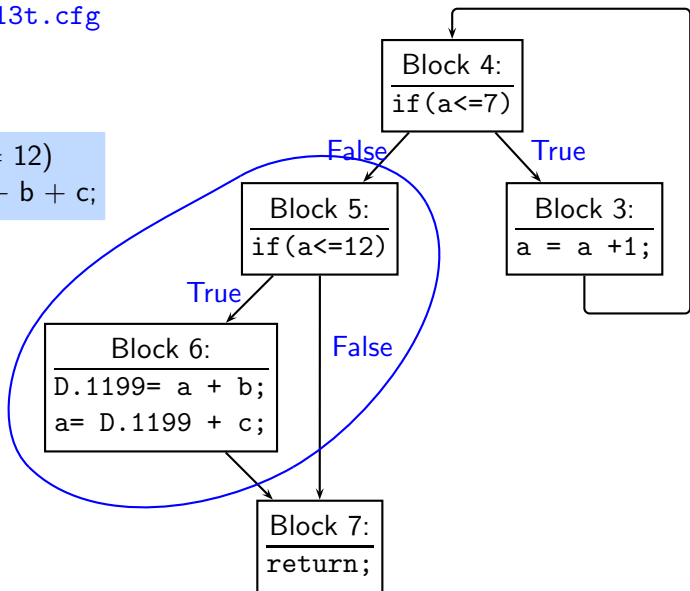
test.c.013t.cfg



Control Flow Graph: Pictorial View

test.c.013t.cfg

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



GIMPLE: Function Calls and Call Graph

test.c

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

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

test.c.000i.cgraph

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



GIMPLE: Function Calls and Call Graph

test.c

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

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

test.c.000i.cgraph

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



GIMPLE: Function Calls and Call Graph

test.c

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

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

test.c.000i.cgraph

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



GIMPLE: Function Calls and Call Graph

test.c

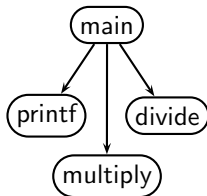
```
extern int divide(int, int);
int multiply(int a, int b)
{
    return a*b;
}

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

test.c.000i.cgraph

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

call graph



GIMPLE: Function Calls and Call Graph

test.c

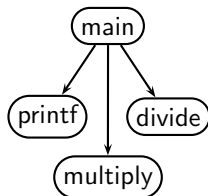
```
extern int divide(int, int);
int multiply(int a, int b)
{
    return a*b;
}

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

test.c.000i.cgraph

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

call graph



GIMPLE: Call Graphs for Recursive Functions

test.c

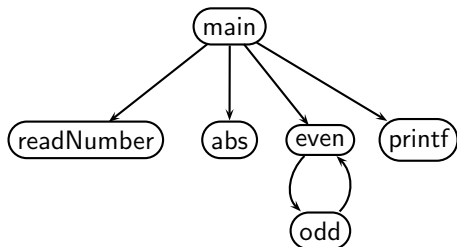
```
int even(int n)
{ if (n == 0) return 1;
  else return (!odd(n-1));
}

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

main()
{ int n;

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

call graph



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

$$\left| \begin{array}{l} x \\ y \\ (y + x) \\ (y + x) + y \end{array} \right| \begin{array}{l} 2 \\ 3 \\ \\ \end{array}$$



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

$$\left| \begin{array}{l} x \\ y \\ (y + x) \\ (y + x) + y \end{array} \right| \begin{array}{l} 3 \\ 3 \\ \\ \end{array}$$



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

$$\begin{array}{l|l} x & 3 \\ y & 3 \\ (y + x) & 6 \\ (y + x) + y & \end{array}$$



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

$$\begin{array}{|l|l|} \hline x & 3 \\ y & 4 \\ (y + x) & 6 \\ (y + x) + y & \\ \hline \end{array}$$



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

$$\begin{array}{|l} x \\ y \\ (y + x) \\ (y + x) + y \end{array} \left| \begin{array}{l} 3 \\ 5 \\ 6 \end{array} \right.$$



Inspect GIMPLE When in Doubt (1)

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

What are the values of x and y?

x = 10 , y = 5

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



Inspect GIMPLE When in Doubt (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;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```

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



Inspect GIMPLE When in Doubt (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 (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;  
x = D.1572 + y;  
y = y + 1;
```



Inspect GIMPLE When in Doubt (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;
```



Inspect GIMPLE When in Doubt (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 (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; /* 5 */
```



Inspect GIMPLE When in Doubt (2)

- 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 :

```
int * (* (*MYVAR) (int) ) [10];
```

Hint: Use `-fdump-tree-original-raw-verbose` option. The dump to see is `003t.original`



Part 3

Examining RTL Dumps

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 [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))  
) t.c:24 -1 (nil))
```

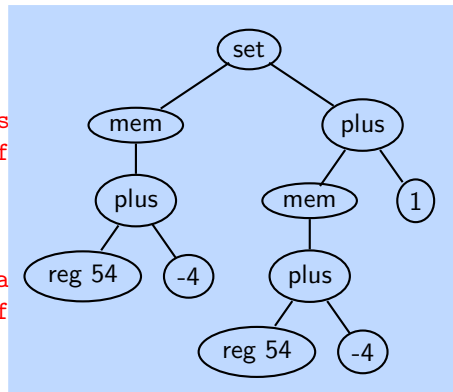


RTL for i386: Arithmetic Operations (1)

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

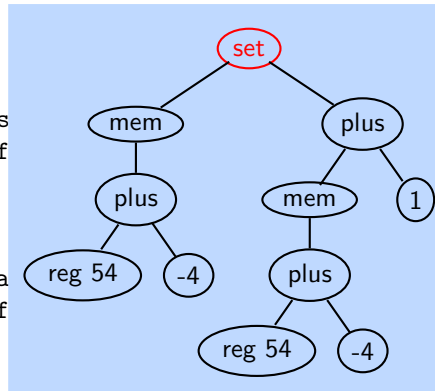


RTL for i386: Arithmetic Operations (1)

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



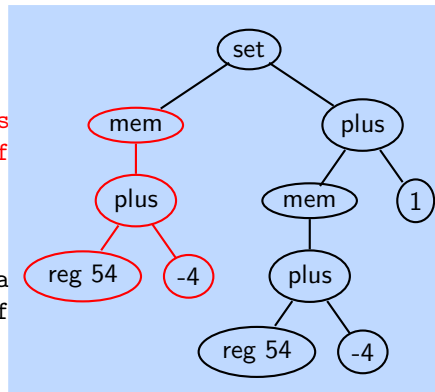
RTL for i386: Arithmetic Operations (1)

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

a is a local variable
allocated on stack



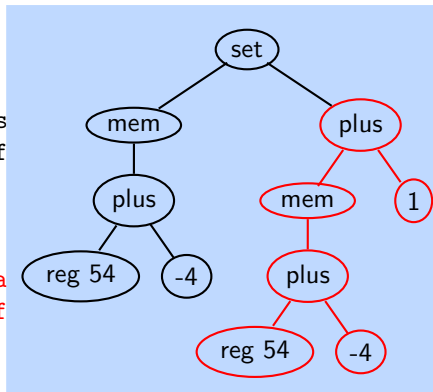
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

Dump file: test.c.144r.expand

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

a is a local variable
allocated on stack



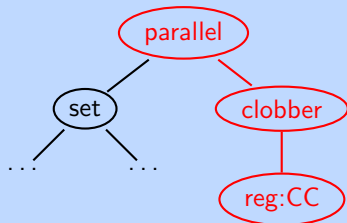
RTL for i386: Arithmetic Operations (1)

Translation of $a = a + 1$

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

```
(insn 12 11 13 4 (parallel [
  ( set (mem/c/i:SI
        (plus:SI
          (reg/f:SI 54 virtual-s
            (const_int -4 [0xffff
              (plus:SI
                (mem/c/i:SI
                  (plus:SI
                    (reg/f:SI 54 virtua
                      (const_int -4 [0xff
                        (const_int 1 [0x1])))
                    (clobber (reg:CC 17 flags))
                  ])) t.c:24 -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.144r.e Output with slim suffix

```
(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])
    (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))
    ]) t.c:24 -1 (nil))
  { [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 [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))  
) 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]))
    (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))
]) t.c:24 -1 (nil))

```

Current Instruction



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])  
    (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))  
  ]) t.c:24 -1 (nil))
```

Previous Instruction



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]))
    (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))
]) t.c:24 -1 (nil))

```


Next Instruction



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])  
  (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))  
) t.c:24 -1 (nil))
```

Basic Block



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])  
  (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))  
) t.c:24 -1 (nil))
```

File name: Line number



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])  
    (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))  
  ]) t.c:24 -1 (nil))
```

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 [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))  
) t.c:24 -1 (nil))
```

scalar that is not a part of an aggregate



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])  
  (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))  
) t.c:24 -1 (nil))
```

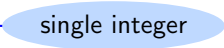
register that holds a pointer



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])  
    (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))  
  ]) t.c:24 -1 (nil))
```

single integer



RTL for i386: Arithmetic Operations (2)

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

Dump file: test.c.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))
```



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+
  )

(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



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+
  )
  (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

reg63 = reg64 + 1



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+
    ]))

(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
 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.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+
    )
  )
  )

(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
 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

```
(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))  
) t1.c:25 -1 (nil))
```



RTL for i386: Arithmetic Operations (3)

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-
        (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



RTL for i386: Arithmetic Operations (3)

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-
        (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?



RTL for i386: Arithmetic Operations (3)

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-
        (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`

```
(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))
```



RTL for i386: Arithmetic Operation (4)

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

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

```
(insn 10 9 11 4 (parallel [  
  (set  
    (mem/c/i:SI  
      (plus:SI  
        (reg/f:SI 53 virtual-  
          (const_int 4 [0x4])))  
      (plus:SI  
        (mem/c/i:SI  
          (plus:SI  
            (reg/f:SI 53 virtu  
              (const_int 4 [0x4]  
            (const_int 1 [0x1])))  
        (clobber (reg:CC 17 flags))  
      ]) t1.c:25 -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.144r.expand`

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

Offset 4 added to the argument pointer register

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



RTL for i386: Arithmetic Operation (4)

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

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

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

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

```
(insn 7 6 8 4 (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)) test.c:6 -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.144r.expand

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

```
(insn 7 6 8 4 (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)) test.c:6 -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.144r.expand

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

```
(insn 7 6 8 4 (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)) test.c:6 -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.144r.expand

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

```
(insn 7 6 8 4 (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)) test.c:6 -1 (nil))
```

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



RTL for i386: Control Flow

What does this represent?

```
(jump_insn 15 14 16 4 (set (pc)
  (if_then_else (lt (reg:CCGC 17 flags)
    (const_int 0 [0x0]))
    (label_ref 12)
    (pc))) p1.c:6 -1 (nil)
(nil)
-> 12)
```



RTL for i386: Control Flow

What does this represent?

```
(jump_insn 15 14 16 4 (set (pc)
  (if_then_else (lt (reg:CCGC 17 flags)
    (const_int 0 [0x0]))
    (label_ref 12)
    (pc))) p1.c:6 -1 (nil)
(nil)
-> 12)
```

$pc = r17 < 0 ? \text{label}(12) : pc$



RTL for i386: Control Flow

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

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

```
(insn 8 7 9 (set (reg:SI 61)
  (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
    (const_int -8 [0xffffffff]))) [0 a+0 S4 A32])) test.c:7 -1
(insn 9 8 10 (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 [0xfffffff8]))) [0 b+0 S4 A32]))) test.c:7
(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)
```



RTL for i386: Control Flow

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

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

```
(insn 8 7 9 (set (reg:SI 61)
  (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
    (const_int -8 [0xffffffff]))) [0 a+0 S4 A32])) test.c:7 -1
(insn 9 8 10 (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 [0xfffffff8]))) [0 b+0 S4 A32]))) test.c:7
(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)
```



RTL for i386: Control Flow

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

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

```
(insn 8 7 9 (set (reg:SI 61)
  (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
    (const_int -8 [0xffffffff]))) [0 a+0 S4 A32])) test.c:7 -1
(insn 9 8 10 (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 [0xfffffff8]))) [0 b+0 S4 A32]))) test.c:7
(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

test.c

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

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

(insn 11 10 12 3 (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 test.c:5 (nil))

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



Observing Register Allocation for i386

```
test.c
```

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

```
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
```

```
(insn 11 10 12 3 (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 test.c:5 (nil))
```

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



Observing Register Allocation for i386

test.c

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

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

(insn 11 10 12 3 (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 test.c:5 (nil))

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



Observing Register Allocation for i386

test.c

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

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

(insn 11 10 12 3 (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 test.c:5 (nil))

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



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]))
```



Activation Record Structure in Spim

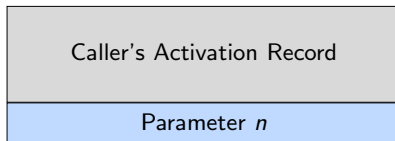


Caller's Activation Record

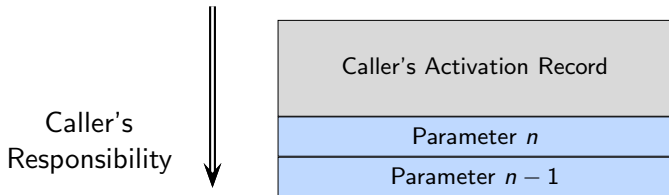


Activation Record Structure in Spim

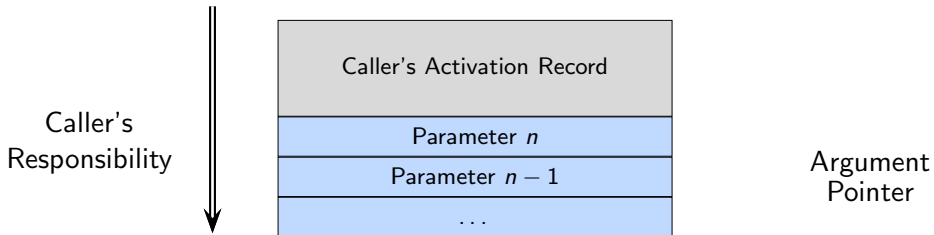
Caller's
Responsibility



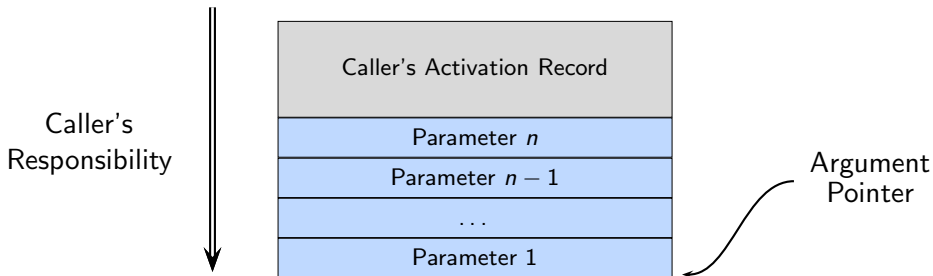
Activation Record Structure in Spim



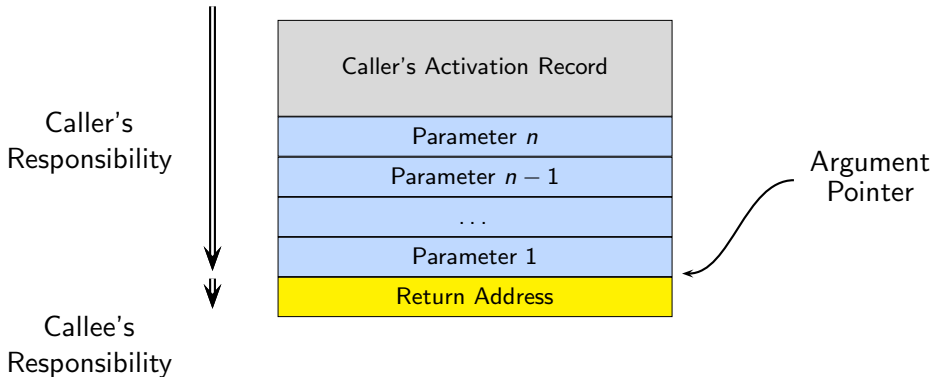
Activation Record Structure in Spim



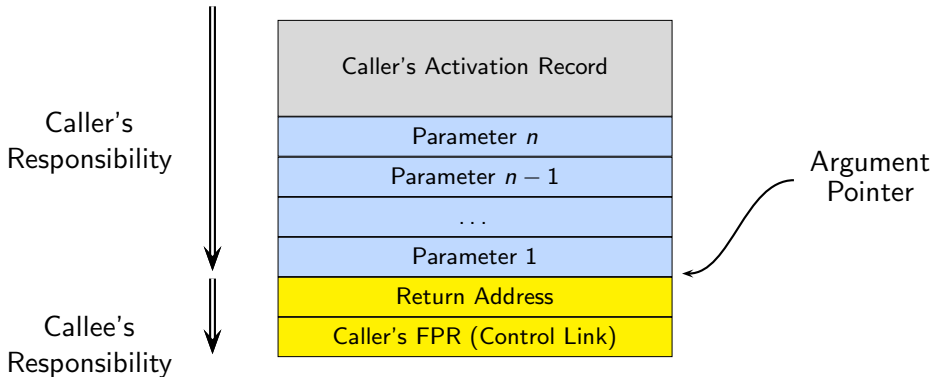
Activation Record Structure in Spim



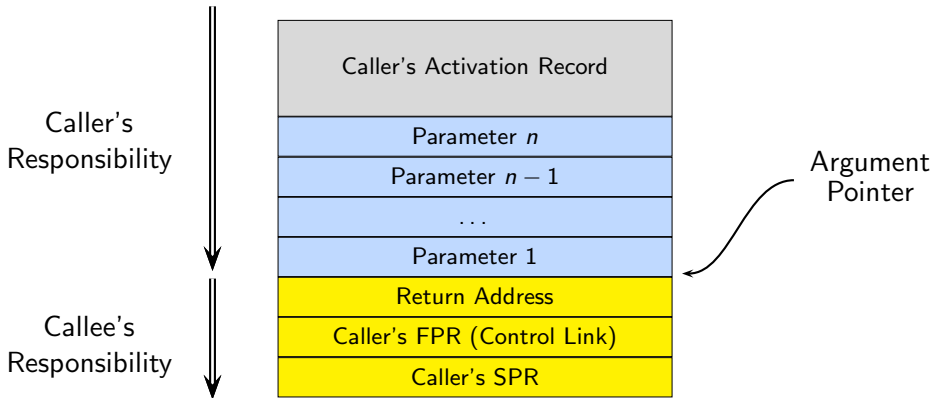
Activation Record Structure in Spim



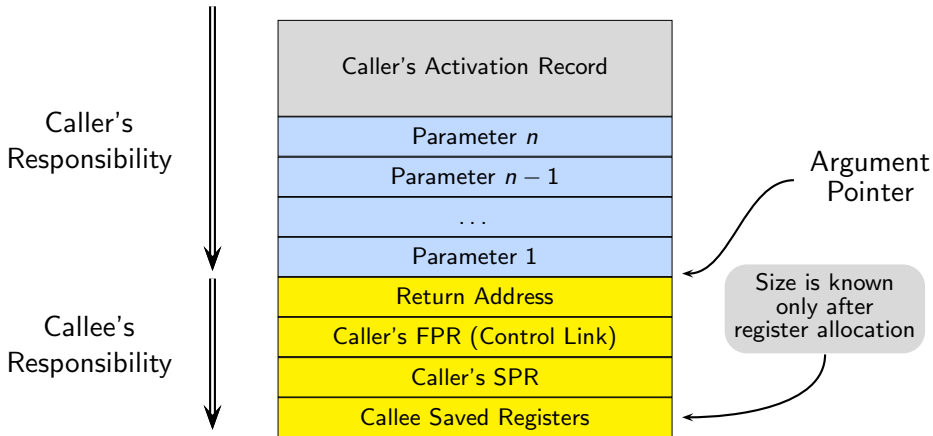
Activation Record Structure in Spim



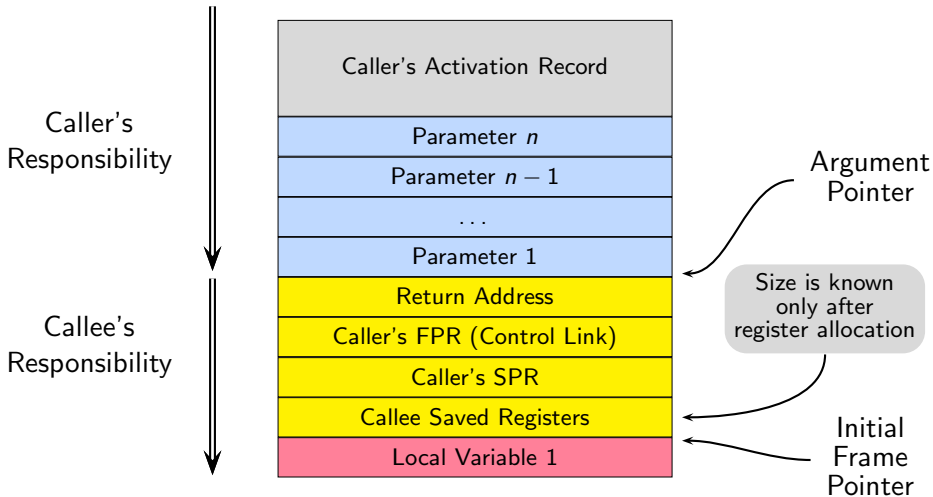
Activation Record Structure in Spim



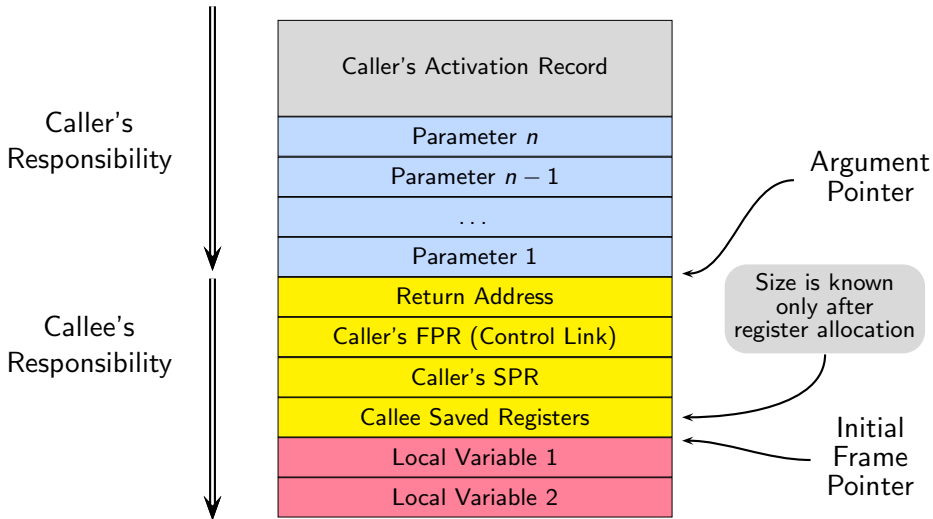
Activation Record Structure in Spim



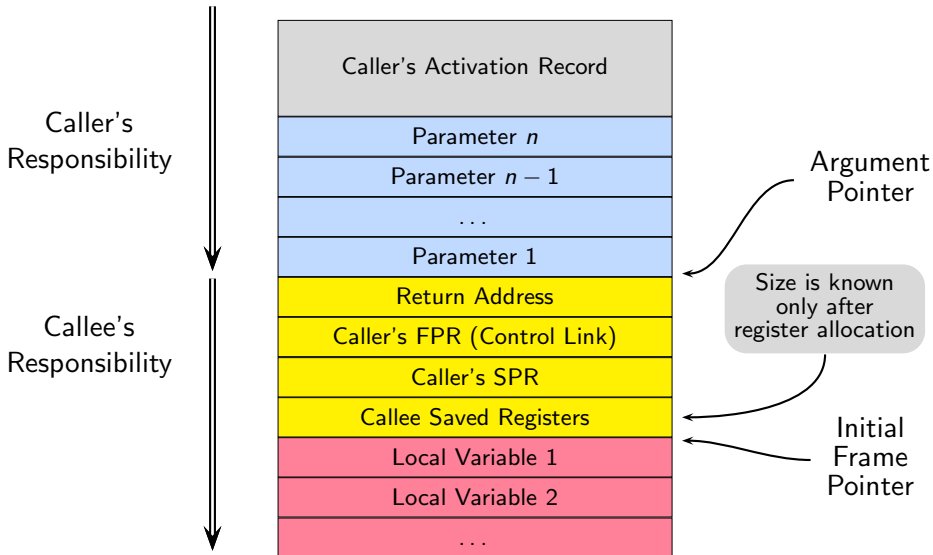
Activation Record Structure in Spim



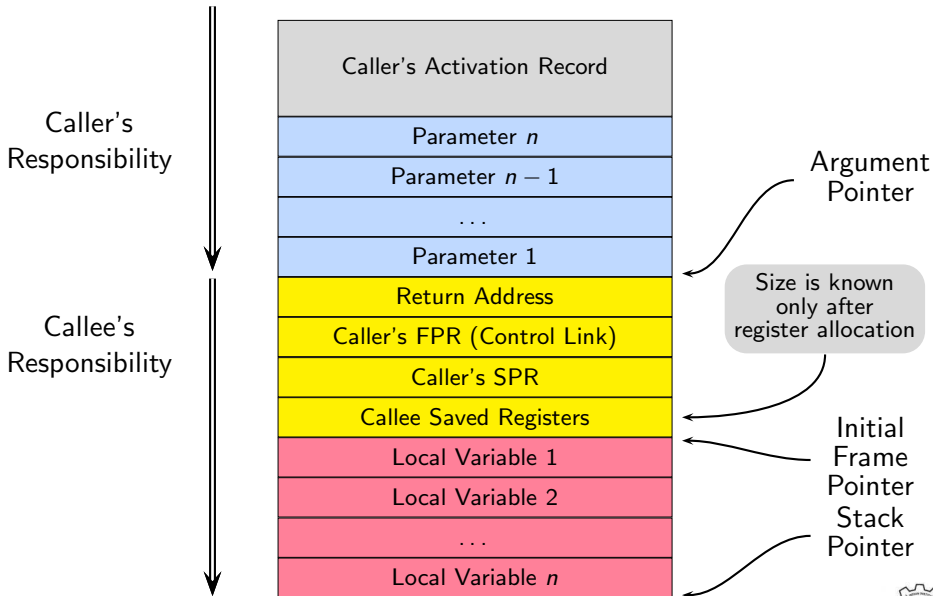
Activation Record Structure in Spim



Activation Record Structure in Spim



Activation Record Structure in Spim



RTL for Function Calls in spim

Calling function	Called function
<ul style="list-style-type: none">• Allocate memory for actual parameters on stack• Copy actual parameters• Call function• Get result from stack (pop)• Deallocate memory for activation record (pop)	<ul style="list-style-type: none">• Allocate memory for return value (push)• Store mandatory callee save registers (push)• Set frame pointer• Allocate local variables (push)• Execute code• Put result in return value space• Deallocate local variables (pop)• Load callee save registers (pop)• Return



Prologue and Epilogue: spim

Dump file: test.c.197r.pro_and_epilogue

```
(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))
```



Prologue and Epilogue: spim

Dump file: test.c.197r.pro_and_epilogue

```

(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))

```

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



Part 4

Examining Assembly Dumps

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;
}
```



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;
}
```



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;
}
```



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;
}
```



Part 5

Conclusions

Gray Box Probing of GCC: Conclusions

- 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
```

