

Workshop on Essential Abstractions in GCC

Gray Box Probing of GCC Translation Sequence

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

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



July 2009

Outline

- Overview of translation sequence in GCC
- Overview of intermediate representations
- Intermediate representations of programs across important phases

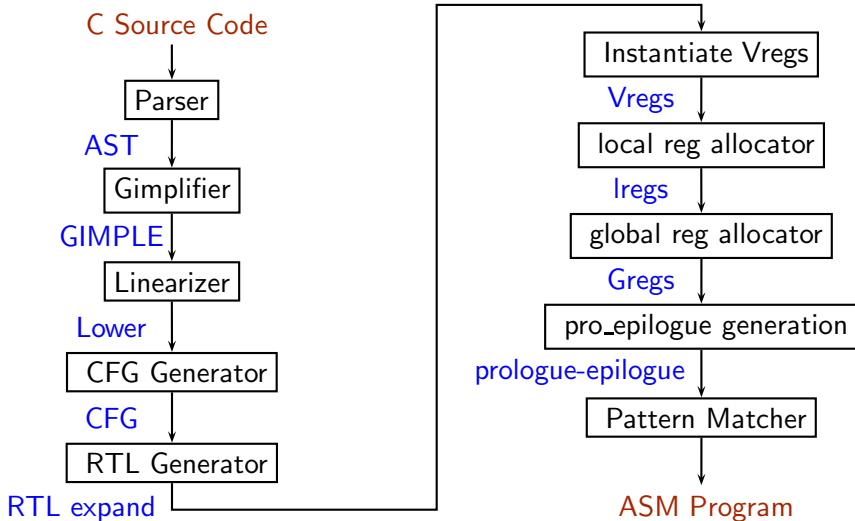


What is Gray Box Probing?

- 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 of a system



Important Phases of GCC



Lowering of abstraction!



Phases of GCC

To see output after each pass use the option
`-fdump-<ir>-<pass>`

- `<ir>`
 - ▶ `-tree-<pass>`
 - ▶ gimple
 - ▶ original
 - ▶ cfg etc.
 - ▶ Use `-all` to see all dumps
 - ▶ `-rtl-<pass>`
 - ▶ expand
 - ▶ greg
 - ▶ vreg etc
 - ▶ Use `-all` to see all dumps

Example: `gcc -fdump-tree-all -fdump-rtl-all test.c`



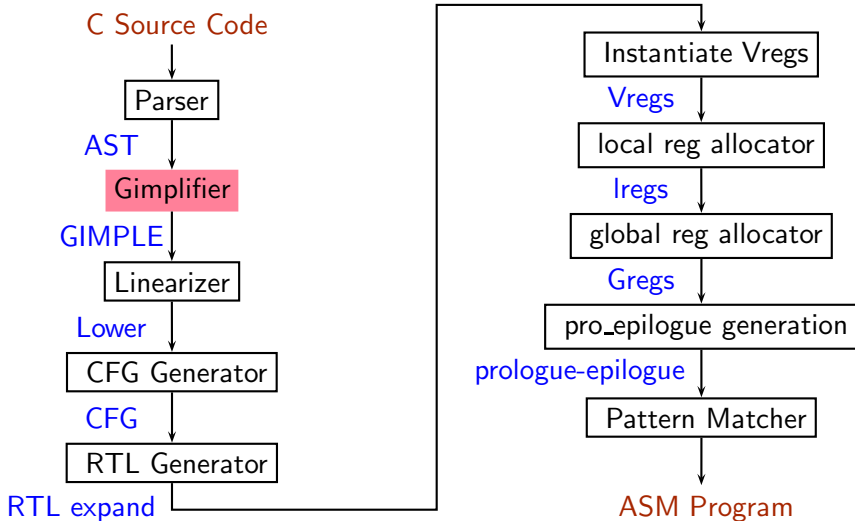
GCC Internal Representation

Example: test.c

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



Important Phases of GCC



Gimplifier

- Three-address representation derived from GENERIC by breaking down into tuples of no more than 3 operands
- Temporaries are introduced to hold intermediate values needed to compute complex expressions.
- Control structures are explicated into conditional jumps.



Gimple

Gimple

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

Source

```
while (a <= 7)  
{  
    a = a+1;  
}
```



Gimple

Gimple

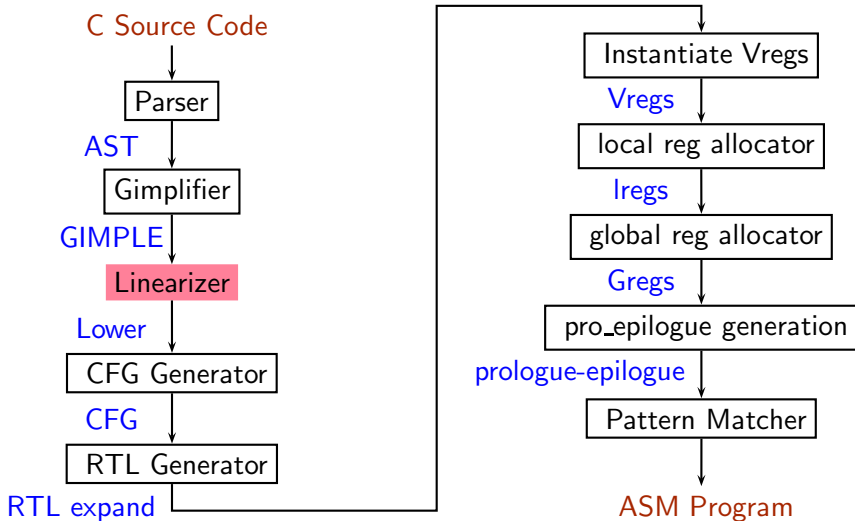
```
if (a <= 12)
{
    D.1199 = a + b;
    a = D.1199 + c;
}
else
{
}
```

Source

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



Important Phases of GCC



Gimple - Lower

Lower

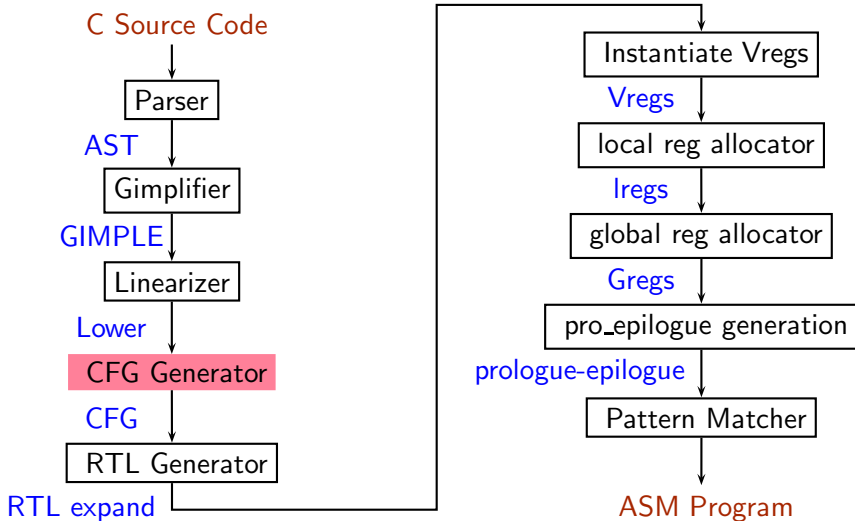
```
if (a <= 12) goto <D.1200>;  
else goto <D.1201>;  
<D.1200>;  
D.1199 = a + b;  
a = D.1199 + c;  
<D.1201>;  
return;
```

Gimple

```
if (a <= 12)  
{  
    D.1199 = a + b;  
    a = D.1199 + c;  
}
```



Important Phases of GCC



Lower - CFG

CFG

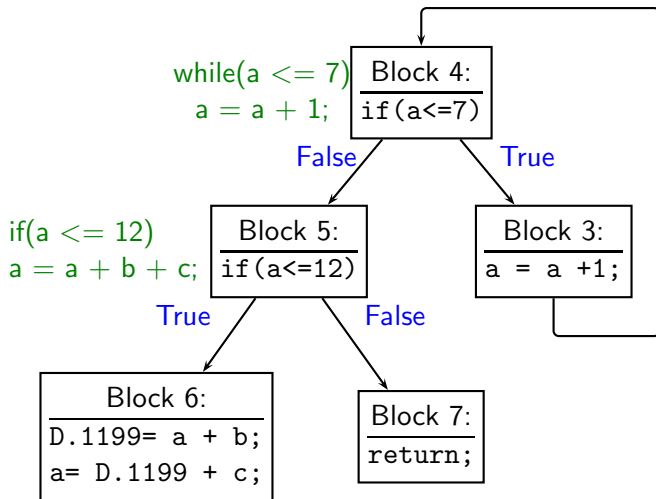
```
# BLOCK 5
if (a <= 7)
    goto <bb 6>;
else
    goto <bb 7>;
# SUCC: 6 (true) 7 (false)
# BLOCK 6
D.1199 = a + b;
a = D.1199 + c;
# SUCC: 7 (fallthru)
# BLOCK 7
return;
# SUCC: EXIT
```

Lower

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



Control Flow Graph

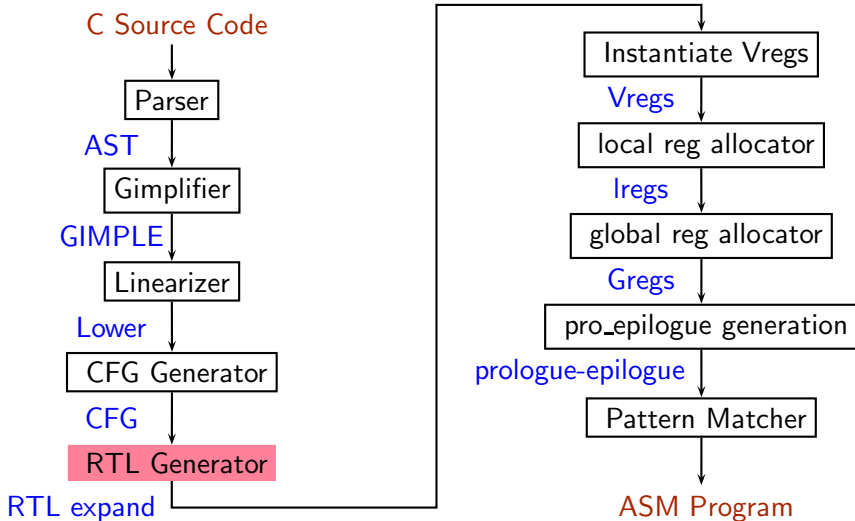


Decisions that have been taken

- Three-address representation is generated
- All high level control flow structures are made explicit.
- Source code divided into interconnected blocks of sequential statements.
- This is a convenient structure for later analysis.



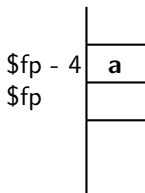
Important Phases of GCC



CFG - RTL Expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 0 test.c:6 (parallel [
  (set (mem/c/i:SI (plus:SI
    (reg/f:SI 54 virtual-stack-vars)
    (const int -4 [...])) [...])
    (plus:SI
      (mem/c/i:SI (plus:SI
        (reg/f:SI 54 virtual-stack-vars)
        (const int -4 [...])) [...])
        (const int 1 [...])))
  (clobber (reg:CC 17 flags))
]) -1 (nil))
```



CFG - RTL Expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 0 test.c:6 (parallel [
  ( set (mem/c/i:SI (plus:SI
    (reg/f:SI 54 virtual-stack-vars)
    (const int -4 [...])) [...])
  (plus:SI
    (mem/c/i:SI (plus:SI
      (reg/f:SI 54 virtual-stack-vars)
      (const int -4 [...])) [...])
    (const int 1 [...])))
  (clobber (reg:CC 17 flags))
]) -1 (nil))
```



CFG - RTL Expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 0 test.c:6 (parallel [
  ( set (mem/c/i:SI (plus:SI
    (reg/f:SI 54 virtual-stack-vars)
    (const int -4 [...])) [...])
    (plus:SI
      (mem/c/i:SI (plus:SI
        (reg/f:SI 54 virtual-stack-vars)
        (const int -4 [...])) [...])
      (const int 1 [...])))
  (clobber (reg:CC 17 flags))
]) -1 (nil))
```



CFG - RTL Expand

```
stack($fp - 4) = stack($fp - 4) + 1
|| flags=?
```

```
(insn 12 11 0 test.c:6 (parallel [
  ( set (mem/c/i:SI (plus:SI
    (reg/f:SI 54 virtual-stack-vars)
    (const int -4 [...])) [...])
  (plus:SI
    (mem/c/i:SI (plus:SI
      (reg/f:SI 54 virtual-stack-vars)
      (const int -4 [...])) [...])
    (const int 1 [...])))
  (clobber (reg:CC 17 flags))
]) -1 (nil))
```



RTL in Spim

```
a = a + 1;
```

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

```
(insn 7 6 8 test.c:6 (set (reg:SI 39)  
  (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)  
    (const_int -4 [...])) [...])) -1 (nil))  
(insn 8 7 9 test.c:6 (set (reg:SI 40)  
  (plus:SI (reg:SI 39)  
    (const_int 1 [...]))) -1 (nil))  
(insn 9 8 0 test.c:6 (set  
  (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)  
    (const_int -4 [...])) [...])  
  (reg:SI 40)) -1 (nil))
```



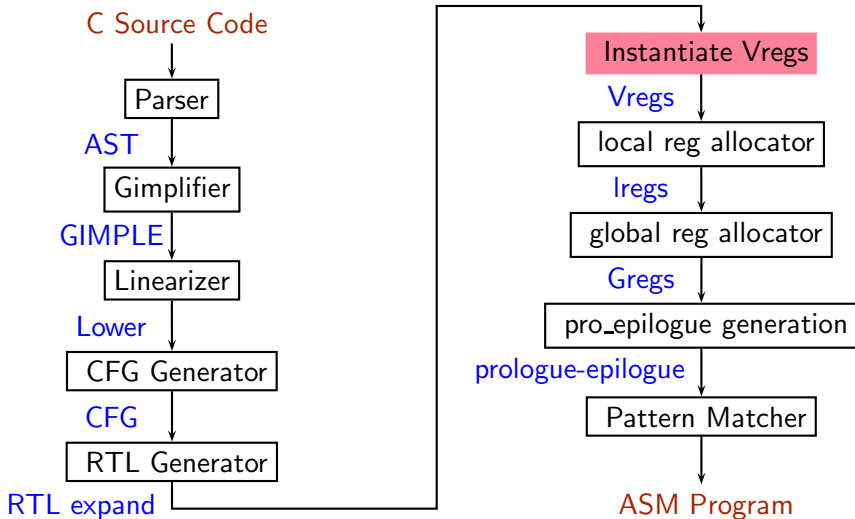
RTL generation

Decisions that are already taken

- A virtual stack is generated.
- RTL instruction is dependant upon architecture.
- Position of each variable is fixed with reference to frame pointer.



Important Phases of GCC



RTL Expand - Vreg

```
(insn 7 6 8 3 (set (reg:SI 39)
  (mem/c/i:SI (plus:SI (reg/f:SI 1 $at )
    (const_int -4 [...])) [...])) 4 *IITB_move_from_mem (nil))
(insn 8 7 9 3 (set (reg:SI 40)
  (plus:SI (reg:SI 39)
    (const_int 1 [...])))) 12 addsi3 (nil))
(insn 9 8 18 3 (set (mem/c/i:SI (plus:SI (reg/f:SI 1 $at )
  (const_int -4 [...])) [])
  (reg:SI 40)) 5 *IITB_move_to_mem (nil))
```



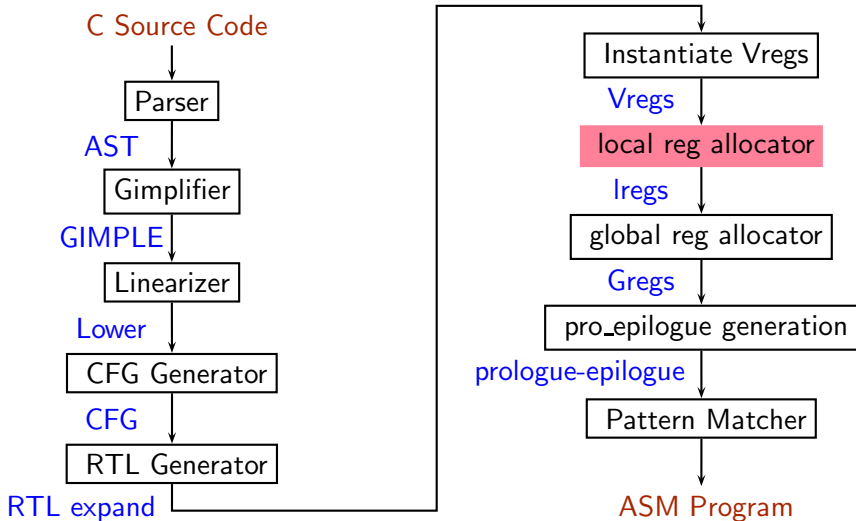
Vregs generation

Decisions that are already taken

- Virtual stack is assigned to a hard register.
- Each insn is committed to an instruction in machine description.
- This commitment helps in scheduling, peephole, and assembly code generation.



Important Phases of GCC

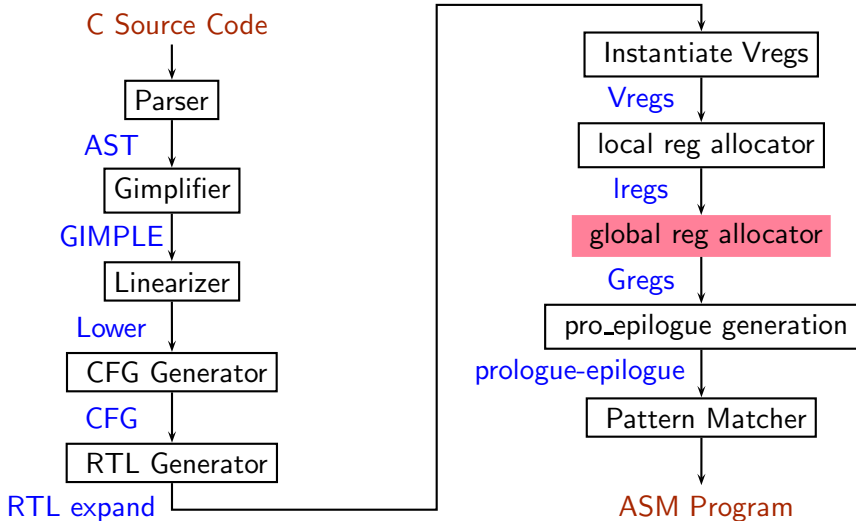


Vregs - Lregs

```
(insn 7 6 8 2 (set (reg:SI 39)
  (mem/c/i:SI (plus:SI (reg/f:SI 1 $at)
    (const_int -4 [...])) [...]))
  4 *IITB_move_from_mem (nil))
(insn 8 7 9 2 (set (reg:SI 40)
  (plus:SI (reg:SI 39)
    (const_int 1 [...])))) 12 addsi3
  (expr_list:REG DEAD (reg:SI 39)(nil))
(insn 9 8 12 2 (set
  (mem/c/i:SI (plus:SI (reg/f:SI 1 $at)
    (const_int -4 [...])) [...])
  (reg:SI 40)) 5 *IITB_move_to_mem
  (expr_list:REG DEAD(reg:SI 40) (nil)))
```



Important Phases of GCC



Lregs - Gregs

```
(insn 7 6 8 3 test.c:4 (set (reg:SI 2 $v0 [39])
  (mem/c/i:SI (plus:SI (reg/f:SI 1 $fp )
    (const_int -4 [...])) [...]))
  4 *IITB_move_from_mem (nil))
(insn 8 7 9 3 test.c:4 (set (reg:SI $v0 [40])
  (plus:SI (reg:SI $v0 [39])
    (const_int 1 [...])))) 12 addsi3 (nil))
(insn 9 8 18 3 test.c:4 (set
  (mem/c/i:SI (plus:SI (reg/f:SI 1 $fp )
    (const_int -4 [...])) [...])
  (reg:SI $v0 40)) 5 *IITB_move_to_mem (nil))
```



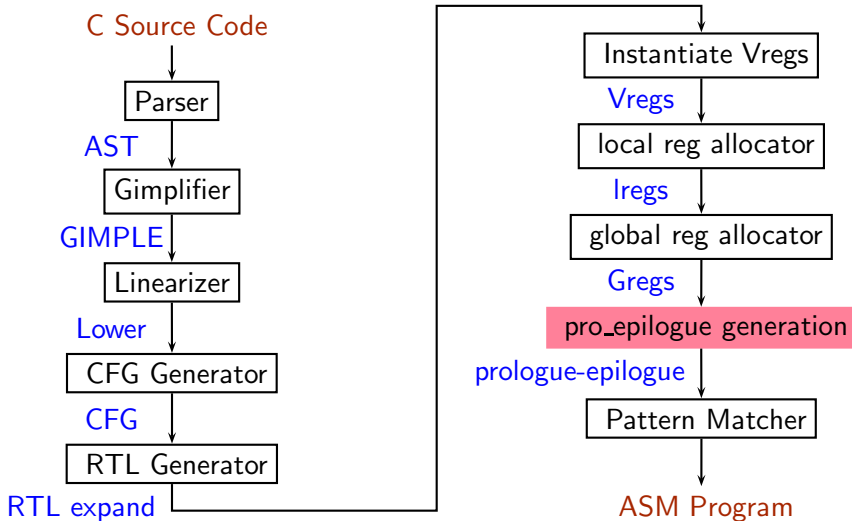
After Greg generation

Decisions that are already taken

- Local and global register are being allocated.
- Register usage has been annotated by REG_DEAD etc.



Important Phases of GCC



RTL for Function Calls in Spim

Calling function	Called function
<ul style="list-style-type: none">• Allocate memory for activation records (push)• Copy arguments into stack arguments• 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



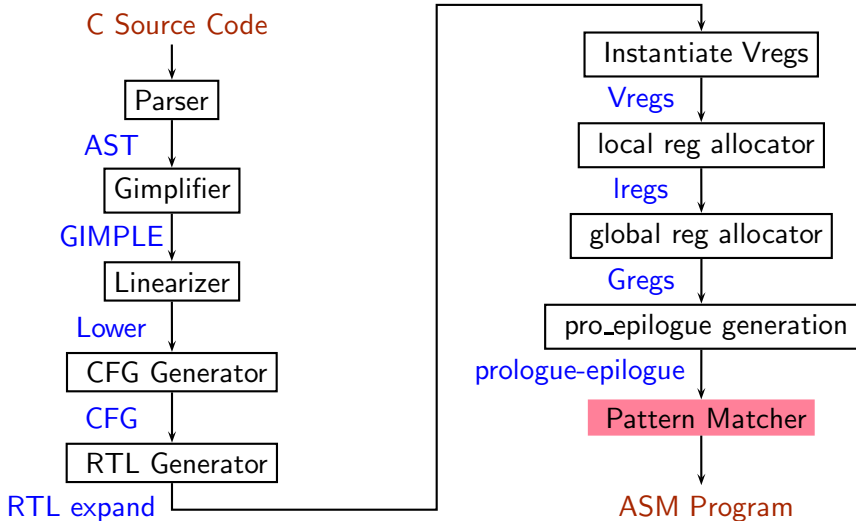
Prologue and Epilogue

```
(insn 17 3 18 2 test.c:2
  (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8])
        (reg:SI 31 $ra)) -1 (nil))
(insn 18 17 19 2 test.c:2
  (set (mem:SI (plus:SI (reg/f:SI 29 $sp)
                        (const_int -4 [...])) [...])
        (reg/f:SI 29 $sp)) -1 (nil))
(insn 19 18 20 2 test.c:2 (set
  (mem:SI (plus:SI (reg/f:SI 29 $sp)
                  (const_int -8 [...])) [...])
  (reg/f:SI 30 $fp)) -1 (nil))
(insn 20 19 21 2 test.c:2 (set
  (reg/f:SI 30 $fp)
  (reg/f:SI 29 $sp)) -1 (nil))
(insn 21 20 22 2 test.c:2 (set
  (reg/f:SI 29 $sp)
  (plus:SI (reg/f:SI 30 $fp)
           (const_int -32 [...]))) -1 (nil))
```

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



Important Phases of GCC



Assembly

Assembly Code for `a = a + 1;`

For spim

```
lw $v0, -8($fp)
addi $v0, $v0, 1
sw $v0, -8($fp)
```

For i386

```
addl $1, -8(%ebp)
```



Conclusion

- Source code is transformed into assembly by moving it step by step close to machine architecture.
- This transformation can be understood to a large extent by observing their inputs and output.
- Fortunately in gcc, output of all the passes can be seen using fdump.
- Complete list of dumps can be seen by doing man gcc.

