Frances: A Tool For Understanding Code Generation

Tyler Sondag[†]   Kian L. Pokorny[♮]   Hridesh Rajan[†]

[†] Iowa State University, Dept. of Computer Science

[♮] McKendree University, Division of Computing

March 11, 2010

Overview

- Compiler and language courses are important
- These courses cover a wide range of difficult topics
- For example, translation to low level languages
  - Students often unfamiliar with low level languages
  - Students are comfortable with high level languages

- **Frances**
  - Leverage familiarity with a high level language
  - Help teach low level languages
  - Help teach language translation
  - Easy to use

Compiler design

- Compiler: translates one language to another
    - Typically: High level language → Low level language
    - Example: C++ → Assembly
- Compiler design in curriculum is common and important
    - Main topic in all Computing Curricula revisions
    - Difficult, but rewarding experience for students

Difficulties in Compiler design

- Compiler design and language courses
  - Wide range of topics to cover
  - Difficult topics
- For example: Language translation
  - Translation itself is difficult, also…
  - Thorough knowledge required of high level language
  - Thorough knowledge required of low level language

Difficulties in Compiler design

- Knowledge required of high level language
  - Most students are experienced with at least one
- Knowledge required of low level (assembly) language
  - Most students have never used such a language
- **On top of learning language translation, most students must learn a new type of language in little time.**
  - What can we do to ease this process?

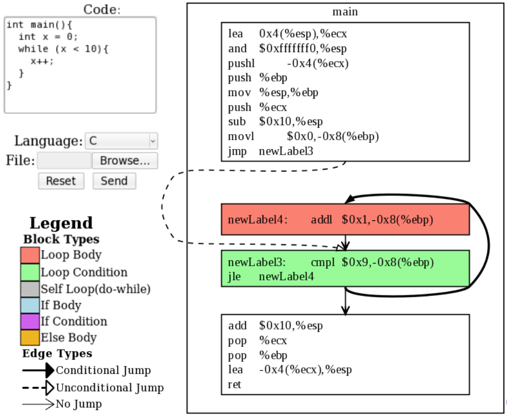
Ideas behind Frances

How to ease learning low level language and translation?

- Students are familiar with a high level language
- Compare user written high level code to compiler generated low level (assembly) code
  - Takes advantage of existing knowledge
  - Ability to compare language features in isolation
  - Try combinations of language constructs
  - Graphical an hands on
  - Easy to use

Frances overview

- Gives a comparison of high level and low level code
  - Graphical side by side representation
  - Color code types of code (ex: loop body vs condition)
  - Distinguish different program path types
  - Maintains ordering
- Easy to use
  - Machine independent
  - No adoption hurdles
  - Simple interface



Code:

```
int main(){
  int x = 0;
  while (x < 10){
    x++;
  }
}
```

Language: C
File: Browse...
Reset Send

Legend
Block Types

🟥	Loop Body
🟩	Loop Condition
⬜	Self Loop(do-while)
🟦	If Body
🟪	If Condition
🟨	Else Body

Edge Types

➔	Conditional Jump
- - ▷	Unconditional Jump
→	No Jump

main

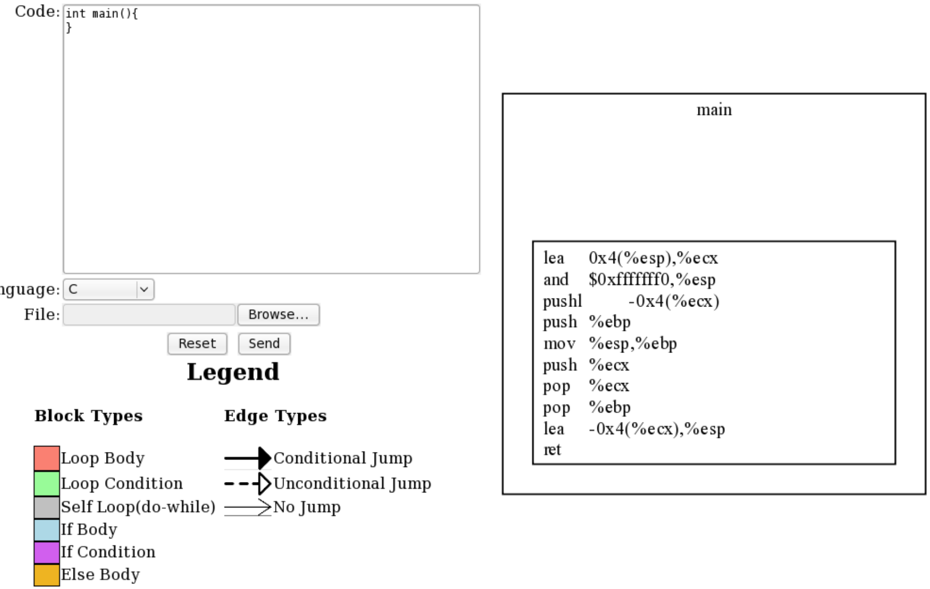
```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl         -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %ecx
sub    $0x10,%esp
movl          $0x0,-0x8(%ebp)
jmp    newLabel3
```

```
newLabel4:      addl   $0x1,-0x8(%ebp)
```

```
newLabel3:      cmpl   $0x9,-0x8(%ebp)
jle    newLabel4
```

```
add    $0x10,%esp
pop    %ecx
pop    %ebp
lea    -0x4(%ecx),%esp
ret
```

- Initial state



Code:
```
int main(){
}
```

nguage: C

File: Browse...

Reset Send

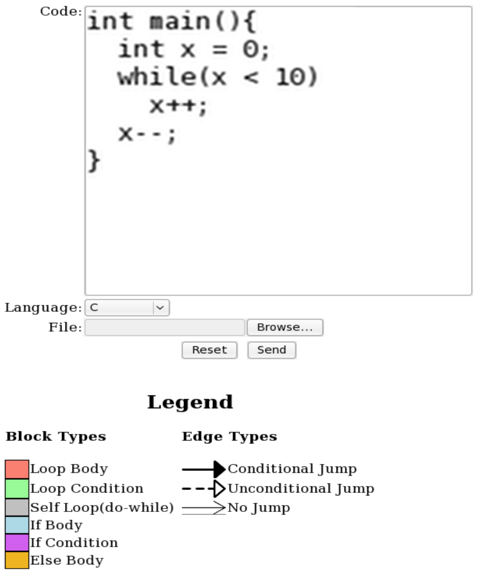
Legend

Block Types	**Edge Types**
🟥 Loop Body	➞ Conditional Jump
🟩 Loop Condition	⇢ Unconditional Jump
⬜ Self Loop(do-while)	→ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	

main

```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl       -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %ecx
pop    %ecx
pop    %ebp
lea    -0x4(%ecx),%esp
ret
```

- Syntax and statement ordering can be confusing
- Ex: ordering of loop condition and body may be swapped



Code:

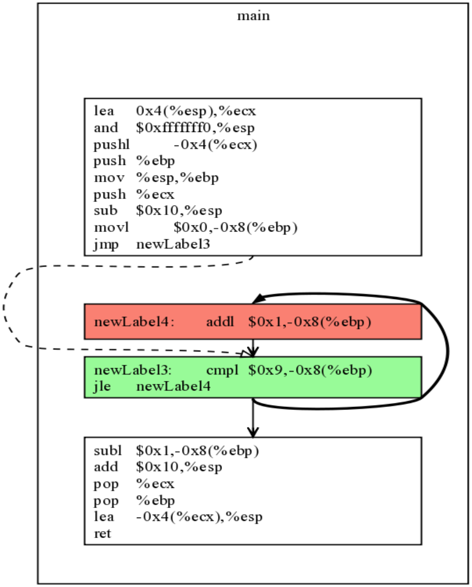
```
int main(){
   int x = 0;
   while(x < 10)
      x++;
   x--;
}
```

Language: C

File:

Legend

Block Types	Edge Types
🟥 Loop Body	➔ Conditional Jump
🟩 Loop Condition	⇢ Unconditional Jump
⬜ Self Loop(do-while)	⟹ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	



main

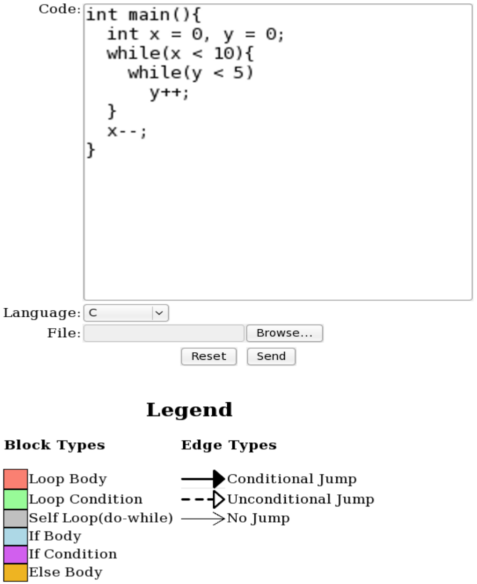
```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl          -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %ecx
sub     $0x10,%esp
movl           $0x0,-0x8(%ebp)
jmp     newLabel3
```

```
newLabel4:       addl   $0x1,-0x8(%ebp)
```

```
newLabel3:       cmpl   $0x9,-0x8(%ebp)
jle     newLabel4
```

```
subl    $0x1,-0x8(%ebp)
add     $0x10,%esp
pop     %ecx
pop     %ebp
lea     -0x4(%ecx),%esp
ret
```

- Nesting



Code:

```
int main(){
  int x = 0, y = 0;
  while(x < 10){
    while(y < 5)
      y++;
  }
  x--;
}
```

Language: C

File: Browse...

Reset Send

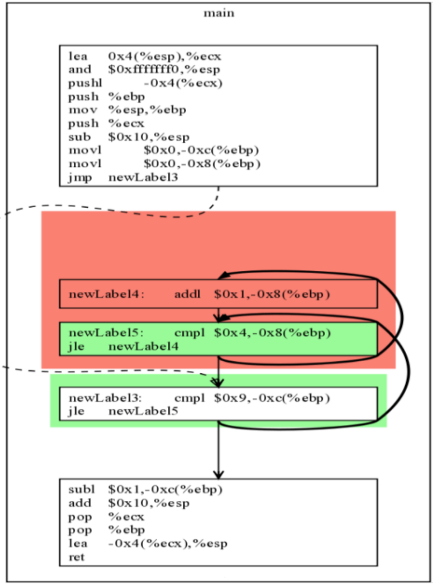
Legend

Block Types

- 🟥 Loop Body
- 🟩 Loop Condition
- ⬜ Self Loop(do-while)
- 🟦 If Body
- 🟪 If Condition
- 🟨 Else Body

Edge Types

- ➔ Conditional Jump
- - - ▷ Unconditional Jump
- → No Jump



```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl           -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %ecx
sub     $0x10,%esp
movl            $0x0,-0xc(%ebp)
movl            $0x0,-0x8(%ebp)
jmp     newLabel3
```

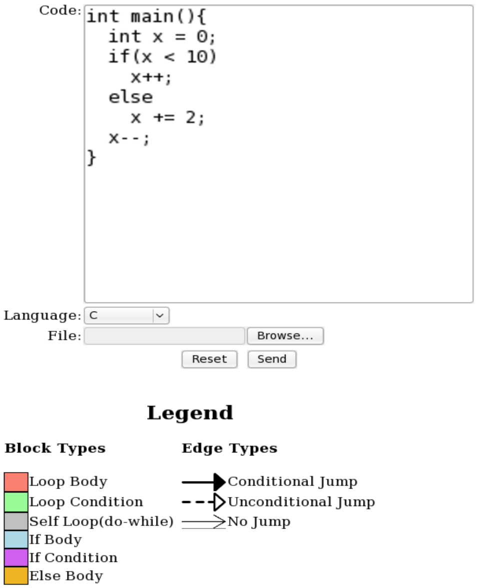
```
newLabel4:       addl   $0x1,-0x8(%ebp)
```

```
newLabel5:       cmpl   $0x4,-0x8(%ebp)
jle     newLabel4
```

```
newLabel3:       cmpl   $0x9,-0xc(%ebp)
jle     newLabel5
```

```
subl    $0x1,-0xc(%ebp)
add     $0x10,%esp
pop     %ecx
pop     %ebp
lea     -0x4(%ecx),%esp
ret
```

- If/else


Code:

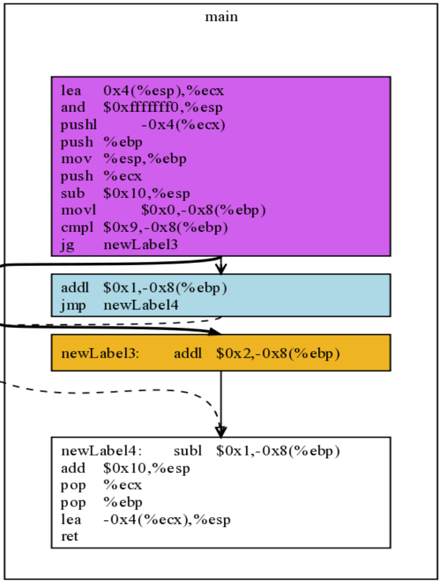
```c
int main(){
   int x = 0;
   if(x < 10)
      x++;
   else
      x += 2;
   x--;
}
```

Language: C

File:

Legend

Block Types	**Edge Types**
🟥 Loop Body	➞ Conditional Jump
🟩 Loop Condition	⇢ Unconditional Jump
⬜ Self Loop(do-while)	→ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	



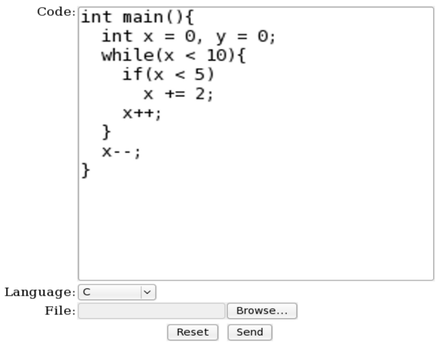
```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl         -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %ecx
sub     $0x10,%esp
movl          $0x0,-0x8(%ebp)
cmpl    $0x9,-0x8(%ebp)
jg      newLabel3
```

```
addl    $0x1,-0x8(%ebp)
jmp     newLabel4
```

```
newLabel3:      addl    $0x2,-0x8(%ebp)
```

```
newLabel4:      subl    $0x1,-0x8(%ebp)
add     $0x10,%esp
pop     %ecx
pop     %ebp
lea     -0x4(%ecx),%esp
ret
```

▶ More nesting

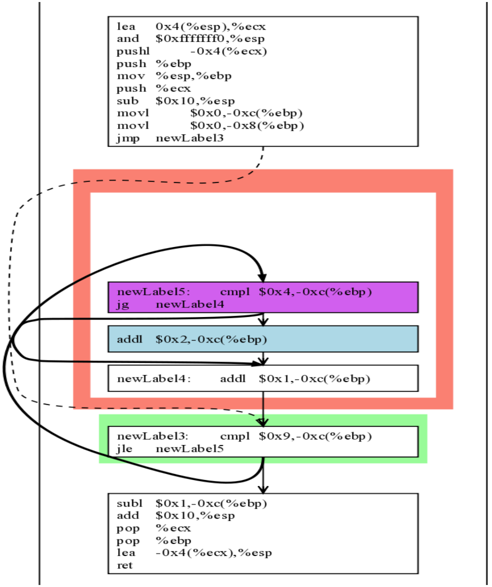


Code:

```
int main(){
  int x = 0, y = 0;
  while(x < 10){
    if(x < 5)
      x += 2;
    x++;
  }
  x--;
}
```

Legend

Block Types	**Edge Types**
🟥 Loop Body	➔ Conditional Jump
🟩 Loop Condition	⇢ Unconditional Jump
⬜ Self Loop(do-while)	⟶ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	



```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl       -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %ecx
sub    $0x10,%esp
movl        $0x0,-0xc(%ebp)
movl        $0x0,-0x8(%ebp)
jmp    newLabel3
```

```
newLabel5:      cmpl  $0x4,-0xc(%ebp)
jg     newLabel4
```

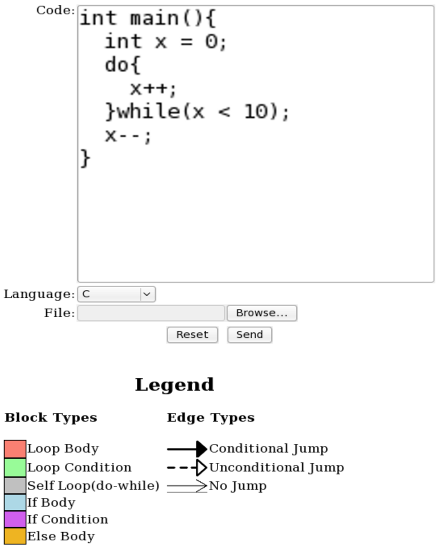
```
addl   $0x2,-0xc(%ebp)
```

```
newLabel4:      addl  $0x1,-0xc(%ebp)
```

```
newLabel3:      cmpl  $0x9,-0xc(%ebp)
jle    newLabel5
```

```
subl   $0x1,-0xc(%ebp)
add    $0x10,%esp
pop    %ecx
pop    %ebp
lea    -0x4(%ecx),%esp
ret
```

▸ Self loop


Code:

```
int main(){
  int x = 0;
  do{
    x++;
  }while(x < 10);
  x--;
}
```

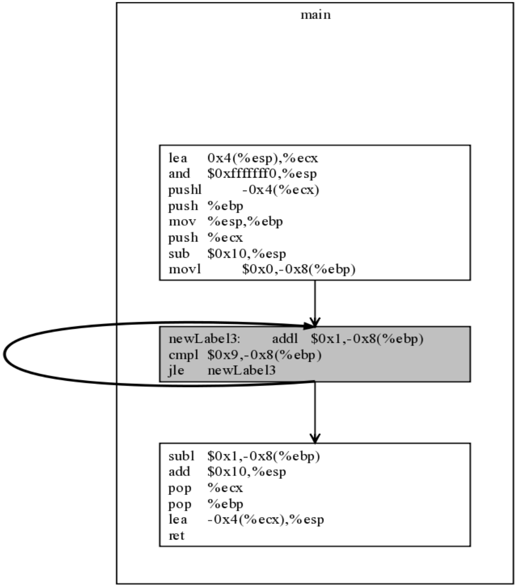
Language: C

File: Browse...

Reset Send

Legend

Block Types	**Edge Types**
🟥 Loop Body	➔ Conditional Jump
🟩 Loop Condition	⇢ Unconditional Jump
⬜ Self Loop(do-while)	→ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	

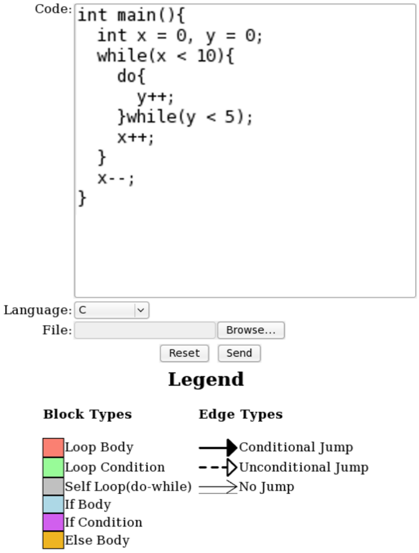

main

```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl           -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %ecx
sub     $0x10,%esp
movl            $0x0,-0x8(%ebp)
```

```
newLabel3:       addl   $0x1,-0x8(%ebp)
cmpl    $0x9,-0x8(%ebp)
jle     newLabel3
```

```
subl    $0x1,-0x8(%ebp)
add     $0x10,%esp
pop     %ecx
pop     %ebp
lea     -0x4(%ecx),%esp
ret
```

▶ More nesting



Code:

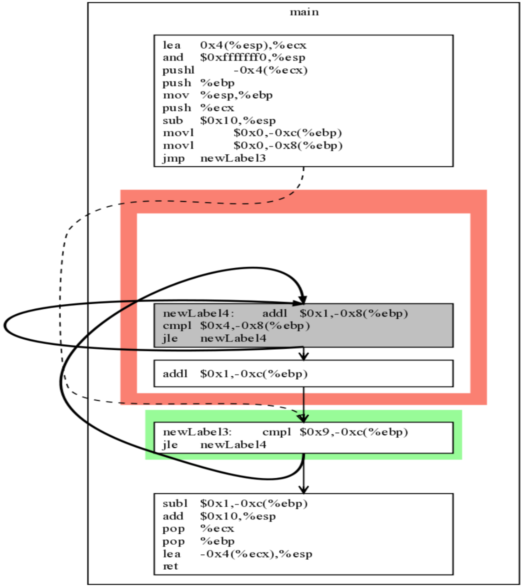
```
int main(){
  int x = 0, y = 0;
  while(x < 10){
    do{
      y++;
    }while(y < 5);
    x++;
  }
  x--;
}
```

Language: C

File:

Legend

Block Types	**Edge Types**
🟥 Loop Body	➔ Conditional Jump
🟩 Loop Condition	- - ▷ Unconditional Jump
▨ Self Loop(do-while)	⟶ No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	



```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl       -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %ecx
sub     $0x10,%esp
movl        $0x0,-0xc(%ebp)
movl        $0x0,-0x8(%ebp)
jmp     newLabel3
```

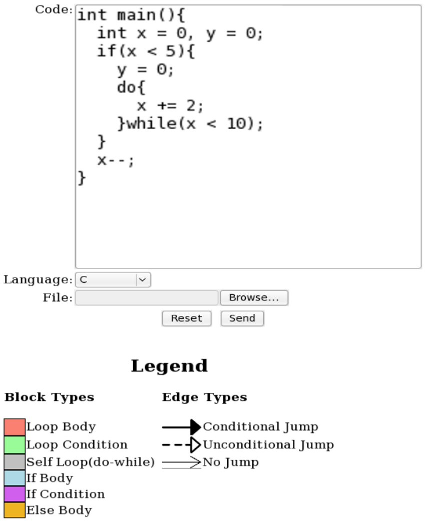
```
newLabel4:      addl   $0x1,-0x8(%ebp)
cmpl   $0x4,-0x8(%ebp)
jle     newLabel4
```

```
addl   $0x1,-0xc(%ebp)
```

```
newLabel3:      cmpl   $0x9,-0xc(%ebp)
jle     newLabel4
```

```
subl    $0x1,-0xc(%ebp)
add     $0x10,%esp
pop     %ecx
pop     %ebp
lea     -0x4(%ecx),%esp
ret
```

▶ More nesting



Code:

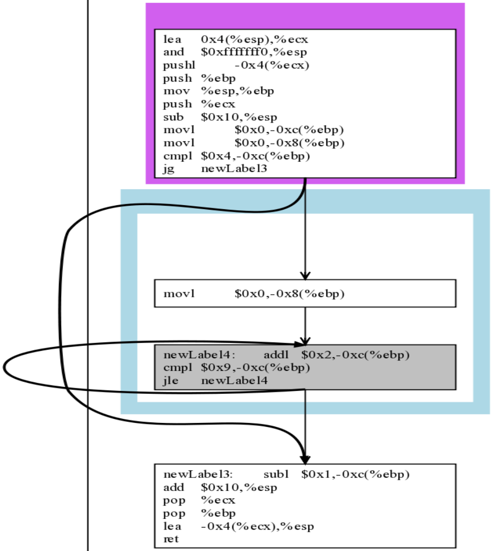
```
int main(){
  int x = 0, y = 0;
  if(x < 5){
    y = 0;
    do{
      x += 2;
    }while(x < 10);
  }
  x--;
}
```

Language: C

File:

Legend

Block Types	Edge Types
🟥 Loop Body	Conditional Jump
🟩 Loop Condition	Unconditional Jump
⬜ Self Loop(do-while)	No Jump
🟦 If Body	
🟪 If Condition	
🟧 Else Body	



```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl       -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %ecx
sub    $0x10,%esp
movl        $0x0,-0xc(%ebp)
movl        $0x0,-0x8(%ebp)
cmpl   $0x4,-0xc(%ebp)
jg     newLabel3
```

```
movl        $0x0,-0x8(%ebp)
```

```
newLabel4:     addl   $0x2,-0xc(%ebp)
cmpl   $0x9,-0xc(%ebp)
jle    newLabel4
```

```
newLabel3:     subl   $0x1,-0xc(%ebp)
add    $0x10,%esp
pop    %ecx
pop    %ebp
lea    -0x4(%ecx),%esp
ret
```

Uses/experiences in a course

- Speeds up teaching/learning code generation/assembly
- Useful for students while implementing code generation
- More time for other/additional material

Course materials available

Future work

- Enhance interface to illustrate program execution
  - Show how instructions impact machine state
  - Show how program paths are taken
- Integrate Frances into additional courses
  - Organization / Architecture
  - CS1 / CS2

Conclusion

- For students, learning code generation and/or low level languages is often difficult.
  - Little experience with low level languages
  - Extensive use of high level languages
- **Frances** takes advantage of a students familiarity with a high level language to help teach how language constructs appear in low level languages.

Questions

Questions?

http://www.cs.iastate.edu/∼sapha/tools/frances/