Compilation Fourth Step: Intermediate Representation (IR)

Flattening the AST to a sequence of instructions

December 25, 2018

IR properties

- **Independent** of the source language

clang(C/CPP)	$\rightarrow$	
flang(Fortran)	$\rightarrow$	
ghc(Haskell)	$\rightarrow$	LLVM IR
llgo(Go)	$\rightarrow$	
...	$\rightarrow$	

- **Independent** of the target language

	$\rightarrow$	x86
	$\rightarrow$	ARM
LLVM IR	$\rightarrow$	WebAssembly
	$\rightarrow$	Mips
	$\rightarrow$	...

- Contains the **entire** information needed for final translation

IR of Industrial Compilers :: LLVM Bitcode
Global variables handled *similarly* in IR and ASM

oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...

File Edit View Search Terminal Help

```
$ cat example_01.c
int x;
int y;
int z;
int w;

int main()
{
        x = 5;
        y = 6;
        z = 7;
        w = 8;

        return x+y+z+w;
}
$ clang -c -emit-llvm example_01.c
$ opt -instnamer -o example_01.bc example_01.bc
$ llvm-dis example_01.bc
$ sed -n '5,27p;28q' example_01.ll
@x = common global i32 0, align 4
@y = common global i32 0, align 4
@z = common global i32 0, align 4
@w = common global i32 0, align 4

; Function Attrs: nounwind uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 5, i32* @x, align 4
  store i32 6, i32* @y, align 4
  store i32 7, i32* @z, align 4
  store i32 8, i32* @w, align 4
  %tmp = load i32, i32* @x, align 4
  %tmp1 = load i32, i32* @y, align 4
  %add = add nsw i32 %tmp, %tmp1
  %tmp2 = load i32, i32* @z, align 4
  %add1 = add nsw i32 %add, %tmp2
  %tmp3 = load i32, i32* @w, align 4
  %add2 = add nsw i32 %add1, %tmp3
  ret i32 %add2
}
```

- declarations (red)
  - default value 0
- stores (blue)
  - name based access
- loads (how many?)
  - name based access
- temps (how many?)
  - tmp,tmp1,tmp2,...
  - add,add1,add2,...
  - the more the merrier?

IR of Industrial Compilers :: LLVM Bitcode
Global variables handled *similarly* in IR and ASM

oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...

File Edit View Search Terminal Help

```
$ cat example_01.c
int x;
int y;
int z;
int w;

int main()
{
        x = 5;
        y = 6;
        z = 7;
        w = 8;

        return x+y+z+w;
}
$ clang -c -emit-llvm example_01.c
$ opt -instnamer -o example_01.bc example_01.bc
$ llvm-dis example_01.bc
$ sed -n '5,27p;28q' example_01.ll
@x = common global i32 0, align 4
@y = common global i32 0, align 4
@z = common global i32 0, align 4
@w = common global i32 0, align 4

; Function Attrs: nounwind uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 5, i32* @x, align 4
  store i32 6, i32* @y, align 4
  store i32 7, i32* @z, align 4
  store i32 8, i32* @w, align 4
  %tmp = load i32, i32* @x, align 4
  %tmp1 = load i32, i32* @y, align 4
  %add = add nsw i32 %tmp, %tmp1
  %tmp2 = load i32, i32* @z, align 4
  %add1 = add nsw i32 %add, %tmp2
  %tmp3 = load i32, i32* @w, align 4
  %add2 = add nsw i32 %add1, %tmp3
  ret i32 %add2
}
```

oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAMPLES

File Edit View Search Terminal Help

```
$ clang -g -O0 -o example_02 example_02.c
$ objdump -S example_01 | cat -n | sed -n '100,123p;124q'
   100  int main()
   101  {
   102    400460:       55                      push   %rbp
   103    400461:       48 89 e5                mov    %rsp,%rbp
   104    400464:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
   105          x = 5;
   106    40046b:       c7 04 25 30 10 60 00    movl   $0x5,0x601030
   107    400472:       05 00 00 00
   108          y = 6;
   109    400476:       c7 04 25 38 10 60 00    movl   $0x6,0x601038
   110    40047d:       06 00 00 00
   111          z = 7;
   112    400481:       c7 04 25 2c 10 60 00    movl   $0x7,0x60102c
   113    400488:       07 00 00 00
   114          w = 8;
   115    40048c:       c7 04 25 34 10 60 00    movl   $0x8,0x601034
   116    400493:       08 00 00 00
   117
   118          return x+y+z+w;
   119    400497:       8b 04 25 30 10 60 00    mov    0x601030,%eax
   120    40049e:       03 04 25 38 10 60 00    add    0x601038,%eax
   121    4004a5:       03 04 25 2c 10 60 00    add    0x60102c,%eax
   122    4004ac:       03 04 25 34 10 60 00    add    0x601034,%eax
   123    4004b3:       5d                      pop    %rbp
```

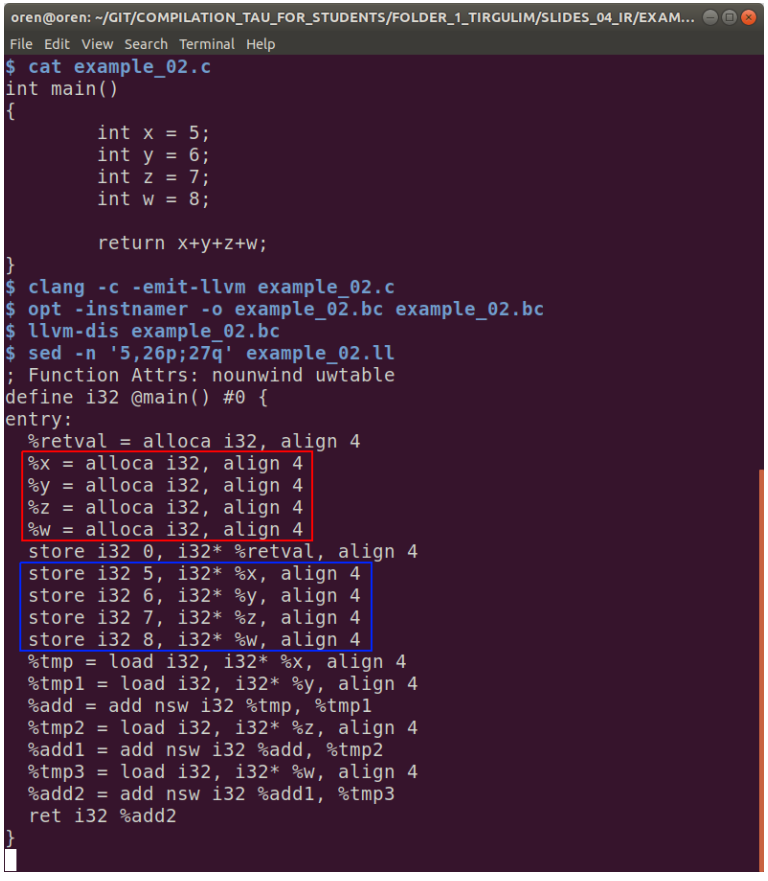
IR of Industrial Compilers :: LLVM Bitcode
Local variables handled *differently* in IR and ASM

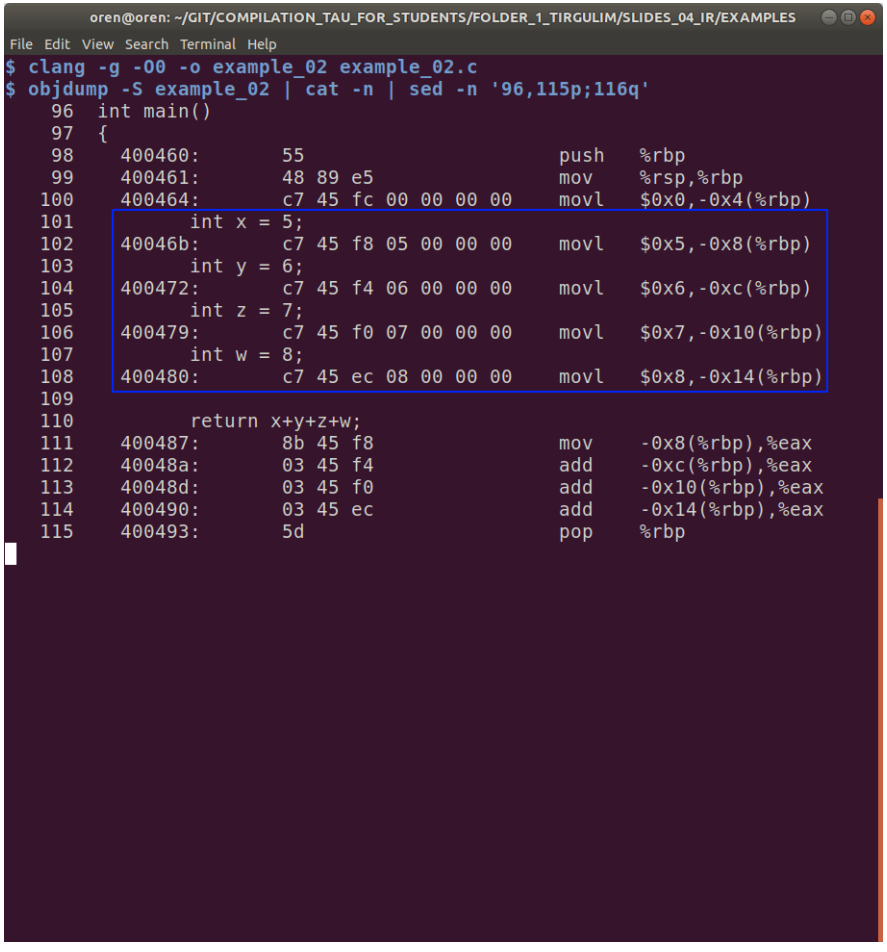
```
oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...
File  Edit  View  Search  Terminal  Help
$ cat example_02.c
int main()
{
        int x = 5;
        int y = 6;
        int z = 7;
        int w = 8;

        return x+y+z+w;
}
$ clang -c -emit-llvm example_02.c
$ opt -instnamer -o example_02.bc example_02.bc
$ llvm-dis example_02.bc
$ sed -n '5,26p;27q' example_02.ll
; Function Attrs: nounwind uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %z = alloca i32, align 4
  %w = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 5, i32* %x, align 4
  store i32 6, i32* %y, align 4
  store i32 7, i32* %z, align 4
  store i32 8, i32* %w, align 4
  %tmp = load i32, i32* %x, align 4
  %tmp1 = load i32, i32* %y, align 4
  %add = add nsw i32 %tmp, %tmp1
  %tmp2 = load i32, i32* %z, align 4
  %add1 = add nsw i32 %add, %tmp2
  %tmp3 = load i32, i32* %w, align 4
  %add2 = add nsw i32 %add1, %tmp3
  ret i32 %add2
}
```

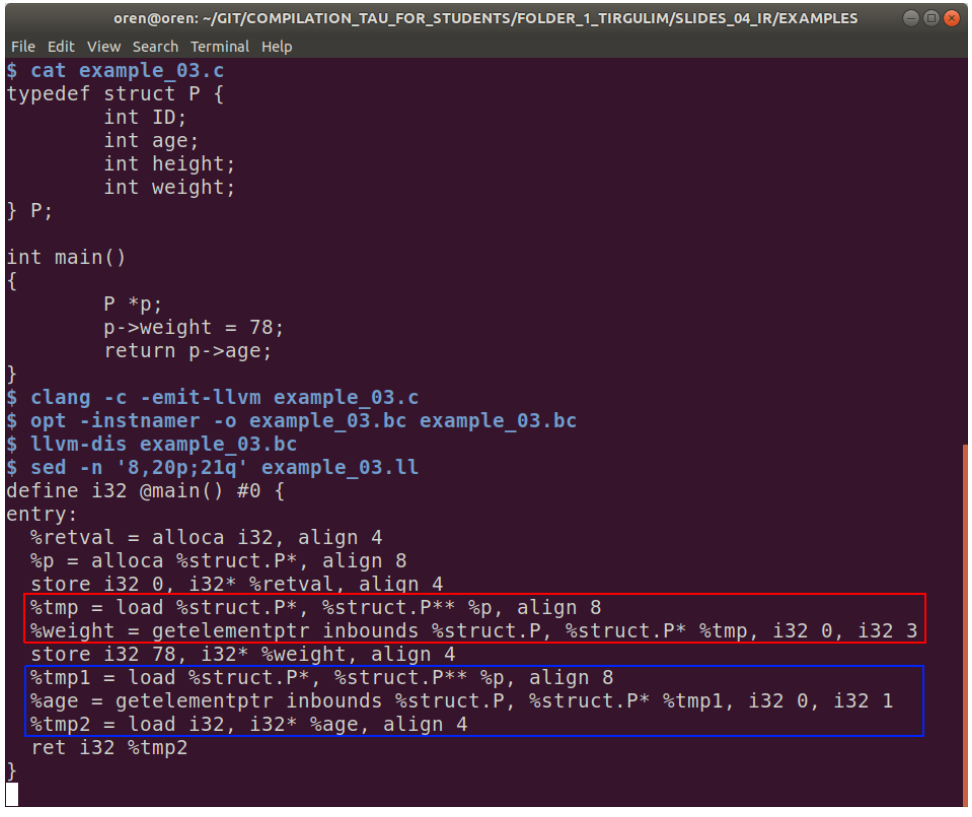
- allocations (red)
  - on main's stack frame
  - no default value
- stores (blue)
  - name based access
  - should eventually be translated to some relative offset from the frame pointer
- loads (how many?)
  - name based access
  - should eventually be translated to some relative offset from the frame pointer

IR of Industrial Compilers :: LLVM Bitcode
Local variables handled *differently* in IR and ASM



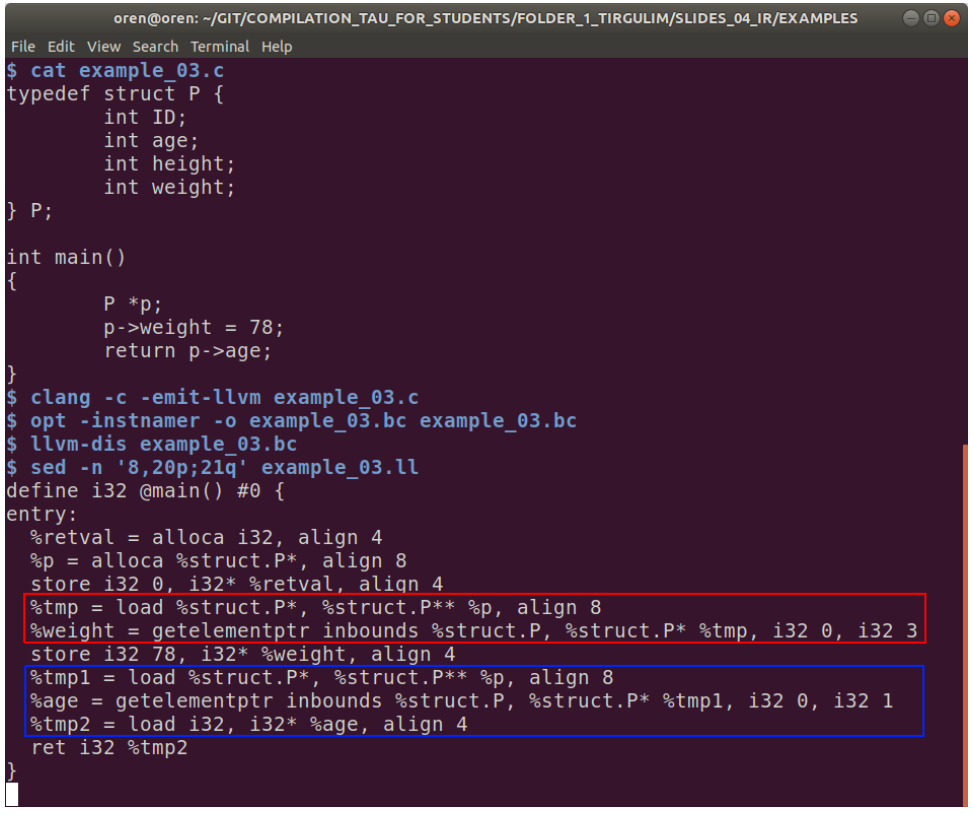


IR of Industrial Compilers :: LLVM Bitcode
How to generate IR for (field) vars?



- how to handle p→weight? (red)
- similar to p→age? (blue)
- handling (field) vars in a uniform way clearly has its advantages.
- However, return p→age; needs the *value*, and p→weight := 78; needs the *address*
- any way to reconcile this?

IR of Industrial Compilers :: LLVM Bitcode
How to generate IR for (field) vars?



- look at the last command in the blue rectangle: tmp2 := [age]
- was in synthesized from p→age? or from its father *return* node?
- To achieve a uniform handling of (field) vars, they need to *always return their address*.
- father nodes (except one, which?) will add an extra indirection to extract the content.

IR of Industrial Compilers :: LLVM Bitcode
How to generate IR for while statements?

oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...

File Edit View Search Terminal Help

```
$ cat example_04.c
int main()
{
        int i;
        int sum=0;
        while (i<19)
        {
                sum = sum + i;
                i++;
        }
        i = 37;
        return sum;
}
$ clang -c -emit-llvm example_04.c
$ opt -instnamer -o example_04.bc example_04.bc
$ llvm-dis example_04.bc
$ sed -i -e 's/\s\{3,\}; preds/    ; preds/g' example_04.ll
$ sed -n '14,34p;35q' example_04.ll

while.cond:     ; preds = %while.body, %entry
  %tmp = load i32, i32* %i, align 4
  %cmp = icmp slt i32 %tmp, 19
  br i1 %cmp, label %while.body, label %while.end

while.body:     ; preds = %while.cond
  %tmp1 = load i32, i32* %sum, align 4
  %tmp2 = load i32, i32* %i, align 4
  %add = add nsw i32 %tmp1, %tmp2
  store i32 %add, i32* %sum, align 4
  %tmp3 = load i32, i32* %i, align 4
  %inc = add nsw i32 %tmp3, 1
  store i32 %inc, i32* %i, align 4
  br label %while.cond

while.end:      ; preds = %while.cond
  store i32 37, i32* %i, align 4
  %tmp4 = load i32, i32* %sum, align 4
  ret i32 %tmp4
}
```

- three labels are created:
  - while.cond (red)
  - while.body (blue)
  - while.end (green)
- how to make sure these labels are unique?
- one *conditional* branch is issued from *while.cond* to either *while.body* or *while.end*
- the other (unconditional) branch is from *while.body* to *while.cond*
- What would be the only difference when generating IR for if statements?

IR of Industrial Compilers :: LLVM Bitcode
How to generate IR for function calls?

oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...

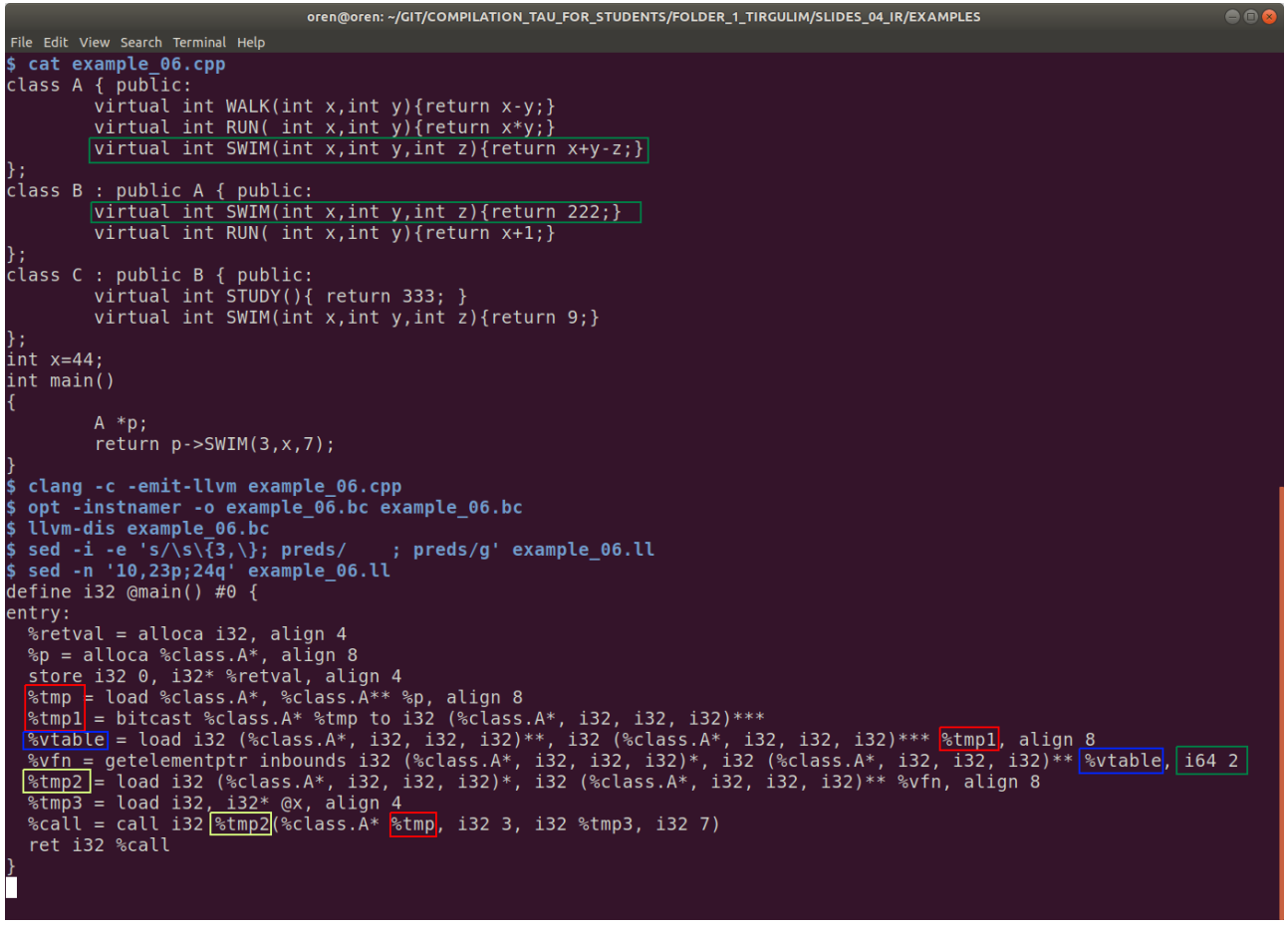
File Edit View Search Terminal Help

```
$ cat example_05.c
int foo(int a,int b,int c)
{
        return a+b*c;
}
int x=46;
int main()
{
        int i=8;
        int j=17;

        return foo(i+j,x,i-j);
}
$ clang -c -emit-llvm example_05.c
$ opt -instnamer -o example_05.bc example_05.bc
$ llvm-dis example_05.bc
$ sed -i -e 's/\s\{3,\}; preds/    ; preds/g' example_05.ll
$ sed -n '25,42p;43q' example_05.ll
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %i = alloca i32, align 4
  %j = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 8, i32* %i, align 4
  store i32 17, i32* %j, align 4
  %tmp = load i32, i32* %i, align 4
  %tmp1 = load i32, i32* %j, align 4
  %add = add nsw i32 %tmp, %tmp1
  %tmp2 = load i32, i32* @x, align 4
  %tmp3 = load i32, i32* %i, align 4
  %tmp4 = load i32, i32* %j, align 4
  %sub = sub nsw i32 %tmp3, %tmp4
  %call = call i32 @foo(i32 %add, i32 %tmp2, i32 %sub)
  ret i32 %call
}
```

- 3 parameters are sent for foo:
  - add = i + j (red)
  - tmp2 = x (blue)
  - sub = i − j (green)
- does evaluation order matter?
- how should we handle the return value?
- which calling convention is implied here?
- which calling convention should we adopt in *our* project?

IR of Industrial Compilers :: LLVM Bitcode
How to generate IR for method calls?



- load *this* to tmp
- load the virtual function table to vtable
- use offset $= 2$ within vtable for method swim load it to tmp2
- pass *this* as the first parameter to the method