

## 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)	→	
flang(Fortran)	→	
ghc(Haskell)	→	LLVM IR
llgo(Go)	→	
...	→	

- **Independent** of the target language

	→	x86
	→	ARM
LLVM IR	→	WebAssembly
	→	Mips
	→	...

- 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 -instname -o example_01.bc example_01.bc  
$ llvmdisasm 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

```
aren@oren: ~/GIT/COMPIATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAMPLES
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 -instname -o example_01.bc example_01.bc
$ llvmdisasm example_01.bc
$ sed -n '5,27p;28q' example_01.ll
@x = common global i32 @, align 4
@y = common global i32 @, align 4
@z = common global i32 @, align 4
@w = common global i32 @, align 4

; Function Attrs: nounwind uwtable
define i32 @main() #0 {
entry:
    %retval = alloca i32, align 4
    store i32 @, 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
}
;
;
```

```
aren@oren: ~/GIT/COMPIATION_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
106     x = 5;
107     40046b:    c7 04 25 30 10 60 00 movl    $0x5,0x601030
108     400472:    05 00 00 00
109
110     y = 6;
111     400476:    c7 04 25 38 10 60 00 movl    $0x6,0x601038
112     40047d:    06 00 00 00
113
114     z = 7;
115     400481:    c7 04 25 2c 10 60 00 movl    $0x7,0x60102c
116     400488:    07 00 00 00
117
118     w = 8;
119     40048c:    c7 04 25 34 10 60 00 movl    $0x8,0x601034
120     400493:    08 00 00 00
121
122     return x+y+z+w;
123     400497:    8b 04 25 30 10 60 00 mov     0x601030,%eax
124     40049e:    03 04 25 38 10 60 00 add     0x601038,%eax
125     4004a5:    03 04 25 2c 10 60 00 add     0x60102c,%eax
126     4004ac:    03 04 25 34 10 60 00 add     0x601034,%eax
127     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 -instname -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

```
oren@oren: ~/GIT/COMPIATION_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 -instname -o example_02.bc example_02.bc  
$ llvm-dis example_02.bc  
; 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  
}
```

```
oren@oren: ~/GIT/COMPIATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAM...  
File Edit View Search Terminal Help  
$ clang -g -O0 -o example_02 example_02.c  
$ objdump -S example_02 | cat -n | sed -n '96,115p;116q'  
96 int main()  
97 {  
98     400460:    55                push    %rbp  
99     400461:    48 89 e5          mov     %rsp,%rbp  
100    400464:    c7 45 fc 00 00 00 movl    $0x0,-0x4(%rbp)  
101  
102    40046b:    c7 45 f8 05 00 00 movl    $0x5,-0x8(%rbp)  
103    int y = 6;  
104    400472:    c7 45 f4 06 00 00 movl    $0x6,-0xc(%rbp)  
105    int z = 7;  
106    400479:    c7 45 f0 07 00 00 movl    $0x7,-0x10(%rbp)  
107    int w = 8;  
108    400480:    c7 45 ec 08 00 00 movl    $0x8,-0x14(%rbp)  
109  
110    return x+y+z+w;  
111    400487:    8b 45 f8          mov     -0x8(%rbp),%eax  
112    40048a:    03 45 f4          add     -0xc(%rbp),%eax  
113    400490:    03 45 f0          add     -0x10(%rbp),%eax  
114    400490:    03 45 ec          add     -0x14(%rbp),%eax  
115    400493:    5d                pop     %rbp
```

# IR of Industrial Compilers :: LLVM Bitcode

## How to generate IR for (field) vars?

```
oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAMPLES
File Edit View Search Terminal Help
$ cat example_03.c
typedef struct P {
    int ID;
    int age;
    int height;
    int weight;
} P;

int main()
{
    P *p;
    p->weight = 78;
    return p->age;
}

$ clang -c -emit-llvm example_03.c
$ opt -instname -o example_03.bc example_03.bc
$ llvm-dis example_03.bc
$ sed -n '8,20p;21q' example_03.ll
define i32 @main() #0 {
entry:
    %retval = alloca i32, align 4
    %p = alloca %struct.P*, align 8
    store i32 0, i32* %retval, align 4
    %tmp = load %struct.P*, %struct.P** %p, align 8
    %weight = getelementptr inbounds %struct.P, %struct.P* %tmp, i32 0, i32 3
    store i32 78, i32* %weight, align 4
    %tmp1 = load %struct.P*, %struct.P** %p, align 8
    %age = getelementptr inbounds %struct.P, %struct.P* %tmp1, i32 0, i32 1
    %tmp2 = load i32, i32* %age, align 4
    ret i32 %tmp2
}
```

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

# IR of Industrial Compilers :: LLVM Bitcode

## How to generate IR for (field) vars?

```
oren@oren: ~/GIT/COMPILATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAMPLES
File Edit View Search Terminal Help
$ cat example_03.c
typedef struct P {
    int ID;
    int age;
    int height;
    int weight;
} P;

int main()
{
    P *p;
    p->weight = 78;
    return p->age;
}

$ clang -c -emit-llvm example_03.c
$ opt -instname -o example_03.bc example_03.bc
$ llvm-dis example_03.bc
$ sed -n '8,20p;21q' example_03.ll
define i32 @main() #0 {
entry:
    %retval = alloca i32, align 4
    %p = alloca %struct.P*, align 8
    store i32 0, i32* %retval, align 4
    %tmp = load %struct.P*, %struct.P** %p, align 8
    %weight = getelementptr inbounds %struct.P, %struct.P* %tmp, i32 0, i32 3
    store i32 78, i32* %weight, align 4
    %tmp1 = load %struct.P*, %struct.P** %p, align 8
    %age = getelementptr inbounds %struct.P, %struct.P* %tmp1, i32 0, i32 1
    %tmp2 = load i32, i32* %age, align 4
    ret i32 %tmp2
}
```

- ▶ look at the last command in the **blue** rectangle: `tmp2 := [ age ]`
- ▶ was it 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 -instname -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 -instname -o example_05.bc example_05.c
$ 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 @foo(i32 %add, i32 %tmp2, i32 %sub)
    ret i32 %call
}
```

- ▶ 3 parameters are sent for foo:
  - ▶  $\text{add} = i + j$  (red)
  - ▶  $\text{tmp2} = x$  (blue)
  - ▶  $\text{sub} = i - j$  (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?

```
File Edit View Search Terminal Help
oren@oren: ~/GIT/COMPIATION_TAU_FOR_STUDENTS/FOLDER_1_TIRGULIM/SLIDES_04_IR/EXAMPLES
$ cat example_06.cpp
class A { public:
    virtual int WALK(int x,int y){return x-y;}
    virtual int RUN( int x,int y){return x*y;}
    virtual int SWIM(int x,int y,int z){return x+y+z;}
};
class B : public A { public:
    virtual int SWIM(int x,int y,int z){return 222;}
    virtual int RUN( int x,int y){return x+1;}
};
class C : public B { public:
    virtual int STUDY(){ return 333; }
    virtual int SWIM(int x,int y,int z){return 9;}
};
int x=44;
int main()
{
    A *p;
    return p->SWIM(3,x,7);
}
$ clang -c -emit-llvm example_06.cpp
$ opt -instname -o example_06.bc example_06.bc
$ llvm-dis example_06.bc
$ sed -i -e 's/\{3,\}; preds/ ; preds/g' example_06.ll
$ sed -n '10,23p;24q' example_06.ll
define i32 @main() #0 {
entry:
    %retval = alloca i32, align 4
    %p = alloca %class.A*, align 8
    store i32 0, i32* %retval, align 4
    %tmp = load %class.A*, %class.A** %p, align 8
    %tmp1 = bitcast %class.A* %tmp to i32 (%class.A*, i32, i32, i32)***
    %vtable = load i32 (%class.A*, i32, i32, i32)***, i32 (%class.A*, i32, i32, i32)*** %tmp1, align 8
    %vfn = getelementptr inbounds i32 (%class.A*, i32, i32, i32)*, i32 (%class.A*, i32, i32, i32)** %vtable, 164 2
    %tmp2 = load i32 (%class.A*, i32, i32, i32)*, i32 (%class.A*, i32, i32, i32)** %vfn, align 8
    %tmp3 = load i32, i32* @x, align 4
    %call = call i32 @tmp2(%class.A* %tmp, i32 3, i32 %tmp3, i32 7)
    ret i32 %call
}
```

- ▶ 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