Оглавление

Полное описание доступно по ссылке http://llvm.org/docs/LangRef.html.

ИДЕНТИФИКАТОРЫ

- бывают глобальные и локальные. Имена глобальных (функции, глобальные переменные) начинаются с символа ‘@’. Локальные (имена регистров, типы) начинаются с ‘%’.

Пример:

%foo, @DivisionByZero, %a.really.long.identifier.

Комментарии начинаются с ‘;’ и идут до конца строки

```
%0 = add i32 %X, %X           ; yields i32:%0
%1 = add i32 %0, %0           ; yields i32:%1
%result = add i32 %1, %1
```

СТРУКТУРА МОДУЛЯ

LLVM программы состоят из модулей, которые соответствуют единицам трансляции. Каждый модуль состоит из функций, глобальных переменных и таблицы символов. Модули могут быть связаны между собой с помощью компоновщика.

Пример:

```
; Declare the string constant as a global constant.
@.str = private unnamed_addr constant [13 x i8] c"hello world\0A\00"
; External declaration of the puts function
declare i32 @puts(i8* nocapture) nounwind
; Definition of main function
define i32 @main() {   ; i32()*
  ; Convert [13 x i8]* to i8  *...
  %cast210 = getelementptr [13 x i8]* @.str, i64 0, i64 0
  ; Call puts function to write out the string to stdout.
  call i32 @puts(i8* %cast210)
  ret i32 0
}
; Named metadata
!0 = metadata !{i32 42, null, metadata !"string"}
!foo = !{!0}
```

В приведенном примере переменная “.str” объявлена как глобальная, функция “puts” внешняя, определены “main” и метаданные “foo”. В общем случае глобальные переменные представлены как указатель, в

нашем примере как указатель на массив символов и указатель на функцию.

Типы данных

Void - аналог типа void в C/C++

Синтаксис:
void
Function Type

Функциональный тип соответствует сигнатуре функции и состоит из типа возвращаемого значения, и списка типов формальных параметров.

Синтаксис:
<тип возвращаемого значения> (<список параметров>)

Пример:
i32 (i32) функция принимающая i32, возвращающая i32
float (i16, i32 *) * Указатель на функцию, принимающую i16 и указатель на i32, возвращающую float.

First Class Types

Инструкции могут вернуть только значения этих типов.

Single Value Types

Типы, которым будут соответствовать типам регистров целевой машины, при генерации кода.

Integer Type

Целочисленные типы, простой тип который указывает количество бит отводимое для хранение значения. Диапазон от 1 бита до 2^{23}-1.

Синтаксис:
iN
N – число бит.

Examples:

i1 1 битное целое.
i32 32 битное целое.
i1942652 1942652 битное целое.

Floating Point Types

half 16-битное число с плавающей точкой
float 32- битное число с плавающей точкой
double 64- битное число с плавающей точкой
fp128 128- битное число с плавающей точкой (112-битная мантиса)
x86_fp80 80- битное число с плавающей точкой (X87)

Pointer Type

LLVM не поддерживает указатель на void (void*) , вместо этого используется i8*.

Синтаксис:
<type> *
Пример:
[4 x i32]* Указатель на массив из четырех i32.
i32 (i32*) * Указатель на функцию, принимающую i32* и возвращающую i32.

Aggregate Types

Array Type

Служит для описания массивов
Синтаксис:
[<# количество элементов> x <тип элементов>]
Примеры:
[40 x i32] Массив из 40 32-битных целых.
[41 x i32] Массив из 41 32-битного целого.
[4 x i8] Массив из 4 8-битных целых.
[3 x [4 x i32]] массив 3x4 содержащий 32- битных целых.

Structure Type

Доступ к элементам структур в памяти осуществляется с использованием инструкций ‘load‘ and ‘store‘ по указателю на поле полученному с помощью инструкции ‘getelementptr‘. Доступ к структурам в регистрах осуществляется с использованием инструкций ‘extractvalue‘ и ‘insertvalue‘.

Синтаксис:

%T1 = type { <список типов> } ; описывает обычную структуру
%T2 = type <{ <список типов > }> ; описывает «packed» структуру – элементы следуют непосредственно друг за другом без выравнивания элементов в памяти, ее размер равен сумме размеров членов структуры.
Пример:
{ i32, i32, i32 } Три i32 значения.
{ float, i32 (i32) * } «Пара» (кортеж из двух элементов) где первый элемент float и второй указатель на функцию, принимающую i32, и возвращающую i32.
<{ i8, i32 }> ‘packed’ структура, с размером в 5 байт

Инструкции

terminator instructions:

Каждый базовый блок должен заканчиваться инструкцией класса “Terminator” , показывающей, что должно быть выполнено после завершения базового блока. Таких инструкций немного:

‘ret‘, ‘br‘, ‘switch‘, ‘indirectbr‘, ‘invoke‘, ‘resume‘, and ‘unreachable‘.

ret – аналог оператора return в C/C++

Пример:

ret i32 5

ret void

br – инструкция передачи управления

br i1 <условие>, label <условие истинно>, label <условие ложно>

br label <dest> ; Unconditional branch

пример:

```
Test:
  %cond = icmp eq i32 %a, %b
  br i1 %cond, label %IfEqual, label %IfUnequal
IfEqual:
  ret i32 1
IfUnequal:
  ret i32 0
```

binary instructions

Принимают два аргумента и возвращают значение, примерами могут служить add, sub, mul, div.

Синтаксис:

<result> = add <ty> <op1>, <op2>
<result> = add nuw <ty> <op1>, <op2>
<result> = add nsw <ty> <op1>, <op2>
<result> = add nuw nsw <ty> <op1>, <op2>
<result> = add i32 4, %var ; i32:result = 4 + %var

Если операция вызвала беззнаковое переполнение, возвращаемое значение будет равно 2^n, где n ширина результата в битах.
nuw и nsw сокращения от “No Unsigned Wrap” и “No Signed Wrap”. Если эти ключевые слова присутствуют, то при переполнении в качестве результирующего значения будет использовано специальное значение “poison value” (см. http://llvm.org/docs/LangRef.html#poisonvalues)

bitwise binary instructions

например: and,or, xor;

memory instructions

alloca

инструкция ‘alloca‘ выделяет память на стеке текущей функции, <результат> = alloca [inalloca] <тип_выделяемой_памяти> [, <тип> <число элементов>] [, align <выравнивание>]

Инструкция ‘alloca‘ выделяет sizeof(<тип>)*КоличествоЭлементов байт на стеке текущей функции, возвращает указатель на выделенную память. Память будет освобождена при возврате управления из этой функции.

Пример:

%ptr = alloca i32
%ptr = alloca i32, i32 4
%ptr = alloca i32, i32 4, align 1024
%ptr = alloca i32, align 1024

load

< результат > = load [volatile] <тип>* <указатель>[, align <выравнивание>]

Инструкция ‘load‘ используется для чтения из памяти.

store

store [volatile] <тип> <значение>, <тип>* <указатель>[, align <выравнивание >]
Инструкция ‘store’ используется для записи в память.

Пример:

```
%ptr = alloca i32                ;  выделить на стеке i32
store i32 3, i32* %ptr           ;  записать 3 по указателю %ptr
%val = load i32* %ptr            ; считать значение по указателю ptr; val =3
```

getelementptr

Используется для получения адреса элемента составного типа данных. Инструкция только вычисляет адрес и не осуществляет доступ к памяти.

<результат> = getelementptr <Тип указателя>* <переменная_указывающая_на_составной_тип>{, <тип> <индекс>}*

<результат> = getelementptr inbounds < Тип указателя >* < переменная_указывающая_на_составной_тип >{, <тип> <индекс>}*

Первый аргумент всегда указатель, являющийся базой для вычислений. Остальные аргументы являются индексами для обращения к элементам составного типа. Интерпретация каждого индекса зависит от индексируемого типа. Первый индекс относится к указателю, переданному первым аргументом, второй индекс к значению на которое указывает указатель (не обязательно, чтобы значение адресовалось напрямую, поскольку первый индекс может быть не нулевым). Первый индексируемый тип должен быть указателем, последующие могут быть: array, vector, struct. Примечание: последующие индексируемые типы не могут быть указателями, поскольку тогда потребовалось бы загружать указатель, перед продолжением вычислений.

Рассмотрим пример:

```
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};
```

```
int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```

Код LLVM:

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }

define i32* @foo(%struct.ST* %s) nounwind uwtable readnone optsize ssp {
entry:
  %arrayidx = getelementptr inbounds %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13
  ret i32* %arrayidx
}
```

В примере выше первый индекс относится к типу ‘%struct.ST*‘, являющейся указателем на структуру ‘%struct.ST‘ = ‘{ i32, double, %struct.RT }‘. Второй индекс относится к третьему элементу структуры - ‘%struct.RT‘ = ‘{ i8 , [10 x [20 x i32]], i8 }‘, другой структуре. Третий индекс относится ко второму элементу структуры - ‘[10 x [20 x i32]]‘, массиву. Двумерный массив состоит из элементов ‘i32‘ . Инструкция The ‘getelementptr‘ возвращает указатель на этот элемент, поэтому возвращаемое значение имеет тип ‘i32*‘.
Тоже самое можно записать в несколько инструкций:

```
define i32* @foo(%struct.ST* %s) {
  %t1 = getelementptr %struct.ST* %s, i32 1                ; %struct.ST*:%t1
  %t2 = getelementptr %struct.ST* %t1, i32 0, i32 2        ; %struct.RT*:%t2
  %t3 = getelementptr %struct.RT* %t2, i32 0, i32 1        ; [10 x [20 x i32]]*:%t3
  %t4 = getelementptr [10 x [20 x i32]]* %t3, i32 0, i32 5  ; [20 x i32]*:%t4
  %t5 = getelementptr [20 x i32]* %t4, i32 0, i32 13       ; i32*:%t5
  ret i32* %t5
}
```

other instructions

‘icmp‘

Инструкция возвращает булево значение, являющееся результатом сравнения.

<результат > = icmp <условие> <тип> <операнд1>, <операнд2>

Первый операнд является условием сравниения:

eq: равно
ne: не равно
ugt: беззнаковое больше
uge: беззнаковое больше или равно
ult: беззнаковое меньше
ule: беззнаковое меньше или равно
sgt: знаковое больше
sge: знаковое больше или равно
slt: знаковое меньше
sle: знаковое меньше или равно

Пример:
<result> = icmp eq i32 4, 5 ; result=false

‘phi‘

Инструкция ‘phi‘ используется для реализации φ-узла в представлении SSA.

<результат> = phi <тип> [<значение0>, <метка0>], ...

Пример:

```
Loop:
  %indvar = phi i32 [ 0, %LoopHeader ], [ %nextindvar, %Loop ]
  %nextindvar = add i32 %indvar, 1
  br label %Loop
```

'call'

Инструкция 'call' представляет собой обычный вызов функции.

conversion operations

'sext'

Инструкция 'sext' расширяет значение до типа ty2.

<результат> = sext <тип1> <значение> to <тип2>

Пример:
%x = sext i8 -1 to i16