888: LLVM

Week 4 - LLVM-IR II

Tobias Grosser

Last week problem - Sum

```
int sum(int n) {
  int sum = 0;

  for (int i = 0; i <= n; i++)
    sum += i;

  return sum;
}
```

LLVM-IR - Type classes

- **Primitive**
  integer, floating point, label, metadata, void, x86mmx,
- **Derived**
  array, function, pointer, structure, packed structure, vector, opaque

First class types - Non first class types

LLVM-IR - Array type

- Set of elements arranges sequential in memory.
- Takes a type and a constant size
- Only fixed sized multi dimensional arrays.
- No indexing restrictions by type system.

```
[20 x i1]              ; Array of 20 boolean elements
[100 x float]          ; Array of 100 float elements
[20 x [100 x i32]]     ; Array of 20 arrays of
                       ; 100 i32 elements
[0 x float]            ; Zero elemement array. Can be used
                       ; to implement variable sized arrays
```

LLVM-IR - Struct Type

- Collection of data elements in memory.
- Packaging matches the ABI of the underlying processor.
- Use a packed structure to remove padding.

```
{float, i64}
{float, {double, i3}}
{float, [2 x i3]}
<{float, [2 x i3]}>      ; Packed structure.
                         ; Removes padding
```

LLVM-IR - Vector type

- Vector of elements
- Used to apply a single instruction on various elements
- Arbitrary width

```
<4 x float>
<2 x double>
<123 x i3>     ; Probably generates inefficient code
```

LLVM-IR - Pointer type

- Gives a location in memory
- void pointer or pointer to labels not permitted. Use i8*.
- Optional address space qualifier

```
float*                ; Pointer to a float
[5 x float]*          ; Pointer to an array
<2 x float>*          ; Pointer to a vector
float addrspace(5)*   ; Pointer to a float in
                      ; address space 5
```

LLVM-IR - Named Type

- Types can be named
- Names are aliases for types
- Names are not part of the types

```
%intv4 = type <4 x i32>
%intv8 = type <8 x i32>
%floatptr = type float*
%mytype = type { %mytype*, i32 }
```

LLVM-IR - Constants

```
[i1 true, i1 false]          ; Constant array
<i3 5, i3 10>                ; Constant vector
{i1 true, float 15}          ; Constant structure

<2 x i1> zeroinitializer     ; Zero vector
```

LLVM-IR - Instructions

- Computational instructions
- Vector/structure management
- Type conversion
- Memory management
- Control flow instructions

LLVM-IR - Computational instructions

- Are applied elemente wise on vector types

```
%sum = add <2 x i32> %a, %b
%product = fmul <4 x float> %a, %b
%equal = icmp eq <2 x i32> %a, %b
%not_equal = float ne <3 x i5> %c, %d
```

LLVM-IR - Vector management

- Get and set an element
- Shuffle elements by a constant shuffle mask

```
extractelement <4 x float> %vec, i32 0
; yields float

insertelement <4 x float> %vec, float 1, i32 0
; yields <4 x float>

shufflevector <4 x float> %v1, <4 x float> %v2,
              <4 x i32> <i32 0, i32 4, i32 1, i32 5>
; yields <4 x float>
```

LLVM-IR - Array/Structure management

- Extract an element from a structure/array
- Indexes need to be in bounds
- Indexes are constants

```
extractvalue {i32, float} %agg, 0
; yields i32

extractvalue {i32, {float, double}} %agg, 0, 1
; yields double

extractvalue [2 x i32] %array, 0
; yields i32
```

LLVM-IR - Array/Structure management II

- Insert an element into a structure/array.

```
%agg1 = insertvalue {i32, float} undef, i32 1, 0
; yields {i32 1, float undef}
%agg2 = insertvalue {i32, float} %agg1, float %val, 1
; yields {i32 1, float %val}

%aggA = insertvalue {i32, float} zeroinitializer,
                                 i32 1, 0
; yields {i32 1, float 0}
```

LLVM-IR - Allocate memory

- alloca - Allocate memory on the stack
- malloc - Use C stdlib memory allocator

```
%ptr = alloca i32
%ptr = alloca i32, i32 4
%ptr = alloca i32, i32 4, align 1024
%ptr = alloca i32, align 1024
; All yield i32*

%mallocP  = call i8* @malloc(i32 %objectsize)
; yields i8* (void pointer)
```

LLVM-IR - Load/Store memory

- The only operations that can access memory

```
%ptr = alloca i32
store i32 3, i32* %ptr
%val = load i32* %ptr
```

LLVM-IR - Select operation

- Select one value depending on a condition
- a = condition ? valueOne : valueTwo
- No branch (mis) prediction necessary

```
%X = select i1 true, i8 17, i8 42
; yields i8:17
```

LLVM-IR - Type conversion

- Size conversion int $\leftrightarrow$ int
- Size conversion float $\leftrightarrow$ float
- float $\leftrightarrow$ int
- int $\leftrightarrow$ ptr
- Bitcast - Do not change bit representation

```
trunc i32 257 to i8          ; yields i8:1
zext i32 257 to i64          ; yields i64:257
sext i8  -1 to i16           ; yields i16:65535
bitcast <2 x i32> %V to i64;; yields i64: %V
```

Exercise

Will be sent out tonight.