

LLVM BITCODE REFERENCE

TYPED ARGUMENTS

A typed argument is a pair `<t> <k>` where `<t>` is a data type and `<k>` is the name of a memory location or register. For example, `i32 %reg1`. Note that literals do not need an explicit type. Hence `1` is a valid typed argument that will conform to the type expected by its operator.

INSTRUCTION SUBSET

- `define <t0> @<X>([t1] ... [tn]) {...}` Define a function with name `<X>`, return type `<t0>`, and, optionally, arguments `[t1]` through `[tn]`.
- `%k = alloca <t>, align <n>` - Allocate enough memory to fit an instance of type `<t>` with alignment `<n>` and return its virtual location to register `k`. Example: `%reg = alloca i32, align 4`
- `%k = load <t0>, <p>, align <n>` Retrieve a value of type `t0` from the location pointed to by the (pointer) typed-argument `<p>` with alignment `<n>`. The result is stored in register `k`. Example: `%reg = load i32, i32* %ptr, align 4`.
- `store <t0> <p0>, <t1> <p1>, align <n>` Store the value Retrieve a value of type `t0` from the location pointed to by the `<t1>`-typed pointer `<p>` with alignment `<n>`. The result is stored in register `k`. Example: `%reg = load i32, i32* %ptr, align 4`.
- `ret <A>` return the typed-argument `A`. Example: `ret i32 %reg`
- `icmp <X>, <a0> <a1>` perform an `<X>`-comparison of the values of typed arguments `<a0>` and `<a1>`. `X` may be one of
 - `eq`: equal
 - `ne`: not equal
 - `ugt`: unsigned greater than
 - `uge`: unsigned greater or equal
 - `ult`: unsigned less than
 - `ule`: unsigned less or equal
 - `sgt`: signed greater than
 - `sge`: signed greater or equal
 - `slt`: signed less than
 - `sle`: signed less or equal

Example - `%pred = icmp eq %argc, 1 br <a>, label L1, label L2` If typed argument `<a>` evaluates to true, jump to label `L1`, otherwise, jump to label `L2`. Example: `br i1 %pred, label %lblt, label %lblf`

- `%<r> = <X> <a0>, <a1>` Perform math operation `<X>` on typed argument `<a0>`. `<X>` may be one of:
 - `add`: addition
 - `mul`: multiplication
 - `sub`: subtraction
 - `div`: division