**指令选择**

**指令选择的学习**

指令选择包括：

pattern matching

pattern selection

指令特征

单输出指令

最简单的机器指令是一组单输出指令。它们只产生一个可观察的输出，从某种意义上说，“可观察”意味着程序可以读取和访问的值。这包括所有实现单个操作的机器指令（例如加法、乘法和位操作，如 OR 和 NOT），但它也包括更复杂的指令，这些指令实现了一些操作，例如刚刚讨论的具有复杂寻址模式的内存操作（例如加载到register rd 基址寄存器 rx 中指定的内存位置的值加上寄存器 ry 中指定的偏移量加上立即数）；只要可观察到的输出构成单个值，单输出指令就可以任意复杂。

多输出指令

正如预期的那样，多输出指令从同一输入产生多个可观察输出。示例包括计算两个输入值的商和余数的 divrem 指令，以及除了计算结果之外还设置状态标志的算术指令。状态标志（也称为条件标志或条件代码）是表示有关结果的附加信息的位，例如如果发生进位溢出或结果为 0。因此，这些指令通常被称为具有副作用，但实际上这些位只不过是指令产生的附加输出值，因此将被称为多输出指令。访问内存中的值然后递增地址指针的加载和存储指令也被认为是多输出指令。

不相交输出指令

从许多不同的输入值产生许多可观察输出值的机器指令称为不相交输出指令。这些类似于多输出指令，只是后者中的所有输出值都源自相同的输入值。另一种说法是，如果一个人形成了对应于每个输出的表达式图，那么所有这些图都将彼此不相交；我们将在下一章解释这些图是如何形成的。不相交输出指令通常包括 SIMD（单指令、多数据）和向量指令，它们对许多不同的输入值同时执行相同的操作。

内部循环指令

内部循环指令表示其行为表现出某种形式的内部循环的机器指令。例如，TI TMS320C55x [247] 处理器提供 RPT k 指令，该指令将紧随其后的指令重复 k 次，其中 k 是作为机器指令的一部分给出的立即值。这些机器指令是最难利用的指令之一，现在可以通过定制的优化例程或编译器的特殊支持来处理。

相互依赖的指令

最后一类是相互依赖的机器指令集。这包括带有附加约束的指令，这些约束在指令以某种方式组合时出现。一个示例包括一条 ADD 指令，同样来自 TMS320C55x 指令集，如果特定寻址模式用于 ADD，则该指令不能与 RPT k 指令组合。正如我们将在本报告中看到的，这是大多数指令选择器难以处理的另一类机器指令，因为它们通常超出底层技术所做的假设集。此外，它们仅在某些情况下出现。

算了。这综述我是看不下去了。

**LLVM的指令选择**

* **Lowering**

selectionDAG：

先说DAG的节点。

按照那个图打印出来的情况，分别包含指向操作数的节点指针，节点类型枚举, 节点编号与节点的数据类型。

有些只有三个部分，我的看法是，这是叶子结点的缘故。另外，一般来说，叶子结点都是存在寄存器中的操作数。（换句话说，树里面的节点都是ins或者类似的。除了操作数之外，也有dag的入口节点entrytoken，感觉没啥意义，还有load store会多个undef的节点，不知道为什么。。。巴拉巴拉再说吧）

另外，还有ch和glue，前者表示不能换顺序，后者表示不能拆分。

DAG的构建

从IR转换到DAG就是lowering的过程，注意这里面phi不进行lowering的操作。还有就是DAG的建立是根据BB进行的，而不是根据函数。

* **Combine**
* **Legalize**
* **Select**