LD转C示意

#### 触点逻辑

触点量，或者经过简单处理（取反操作）

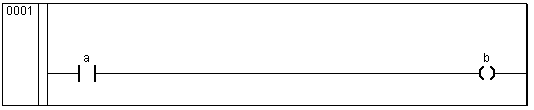


图 4‑6简单LD程序

对应触点元件，翻译为C语言：b = a

在中间语言的树结构中，触点逻辑对应的是一个叶子节点。触点逻辑生成该叶子节点的流程如下图所示：

图 4‑7 触点逻辑生成树节点流程

#### 线圈元件逻辑

线圈元件在LD组态中对应的是一个赋值逻辑，在转换成中间语言的树结构时，线圈逻辑对应的是一个赋值节点。在生成赋值节点的过程中，需要的信息来自由触点、功能块组成的输入逻辑量，在这里我们用InputLogic表示线圈中的输入节点信息。线圈逻辑转换成树节点的流程图如下：

图 4‑8线圈逻辑转换成中间树节点的流程

#### 块元件逻辑

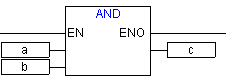


图 4‑9简单块元件相关逻辑

对应逻辑转换到C语言为：

if( eno\_var == TRUE)

{

and\_out = a & b;

c = and\_out;

}

说明：

eno\_var为为块元件AND关联的使能变量；

and\_out为块元件AND关联的中间变量。

块元件转换成中间树的过程：在块元件进行语法检查时，我们已经为功能块生成了两个中间变量，一个是使能引脚中间变量，另一个是完成块运算的中间变量。块元件转换时，首先将块元件的使能端的中间变量进行赋值，即生成一个赋值节点。完成了使能的赋值后，就遍历输入引脚，生成输入引脚所对应的节点，然后由输入引脚生成块元件所完成的运算节点，并将运算结果赋值给块元件的另一个中间变量，最后生成将这个中间变量的值赋值给输出引脚变量的节点。

#### 返回逻辑

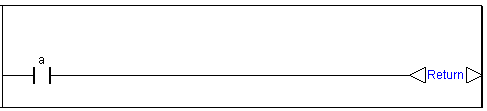


图 4‑10简单返回逻辑

从POU中直接返回。

对应C语言：return。

在上图中，相应的C语言逻辑为：

If(a==TRUE)

{

Return;

}

返回逻辑在转换成中间树的时候，将生成一个RETURN节点。

#### 跳转逻辑

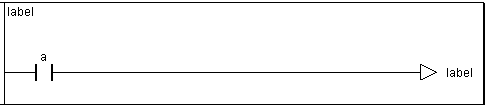


图 4‑11简单跳转逻辑

跳转到指定标号位置。

对应C语言的goto。

在上图中，相应的C语言逻辑为：

If(a == TRUE)

{

goto label;

}

跳转逻辑在转换成中间树的时候，将生成一个JUMP节点。因为跳转逻辑要有相应的跳转目标，所以也会生成一个LABEL节点来表示跳转到的位置。

#### And逻辑

And逻辑生成中间树的And节点，循环遍历And在逻辑存储树结构中的子节点，生成子节点相应的中间树节点，并自底向上生成And节点，示例如下：

QQ截图未命名

图 4‑12 And中间树节点生成示例

上图的组态逻辑将生成下面这样一个树节点：



图 4‑13 And逻辑生成的中间树节点示例

当And逻辑在逻辑存储树结构中存在块元件的子节点时，首先完成对块元件使能端中间变量的赋值，然后将此中间变量挂在And节点下。以下三个图是带有块元件的And逻辑生成的And中间树节点，其中envar是块元件生成的中间变量。

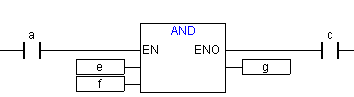


图 4‑14 带有块元件的And逻辑





图 4‑15带有块元件的And逻辑生成的And节点

#### Or逻辑

Or逻辑生成中间树的Or节点，循环遍历Or在逻辑存储树结构中的子节点，生成子节点相应的中间树节点，并自底向上生成Or节点，示例如下：

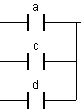


图 4‑16 Or中间树节点生成示例

上面的组态逻辑将生成下面这样一个树节点：



图 4‑17带有块元件的Or逻辑生成的Or节点

当Or逻辑在逻辑存储树结构中存在块元件的子节点时，首先完成对块元件使能端中间变量的赋值，然后将此中间变量挂在Or节点下。以下三个图是带有块元件的Or逻辑生成的Or中间树节点，其中envar是块元件生成的中间变量。

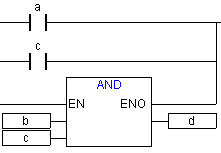


图 4‑18带有块元件的Or逻辑





图 4‑19带有块元件的Or逻辑生成的Or节点

#### Assign逻辑

逻辑存储树结构中的Assign逻辑在转换成中间树中的IF\_ELSE[[6](#中间代码结构)]节点。其转换为IF\_ELSE节点的流程如下图所示：

图 4‑20 Assign逻辑中间树节点生成流程

#### Branches逻辑

在逻辑存储树结构中，Branches逻辑下只包含Assign节点，因此对Branches逻辑生成中间树的过程就是遍历其所有Assign子节点，对Assign节点进行中间树的转换，并将生成的节点加入到树中即可。

### 组态逻辑转换中间树示例

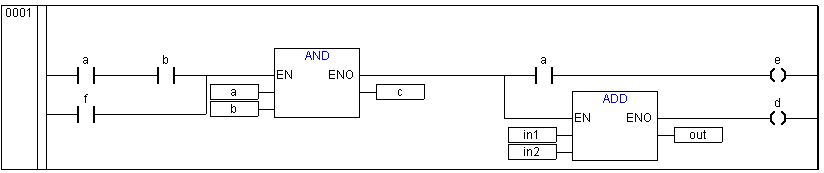


图 4‑21 LD组态示例

此处对上图中的LD程序逻辑进行分析，以C语言来介绍中间代码的生成过程：

EN\_Var1 = (a && b) || f;

If(EN\_Var1 == True)

{

TempValue1 = a && b;

C = TempValue1;

}

e = a && EN\_Var1;

EN\_Var2 = EN\_Var1;

If(EN\_Var2 == TRUE)

{

TempValue2 = in1 + in2;

Out = TempValue2;

}

d = EN\_Var2;

其中EN\_Var1和TempValue1、EN\_Var2、TempValue2是块生成的中间变量

在上面的组态逻辑中，对应的C语言逻辑标号都有对应的中间树节点，上面各表达式对应的中间树节点以及这个组态逻辑对应的生成树如下图所示。

图 4‑22语句的中间树



图 4‑23语句的中间树



图 4‑24语句的中间树



图 4‑25语句的中间树



图 4‑26语句的中间树



图 4‑27语句的中间树

把上面的六棵树按照从上之下的顺序将根节点连接就构成了图 4‑21 LD组态示例所对应的中间树结构。如下图所示：

图 4‑28 LD组态示例对应的中间树

在上图中，AT\_ASDF\_43\_EN、AT\_ASDF\_43为AND功能块生成的中间变量，AT\_ASDF\_47\_EN、AT\_ASDF\_47是ADD功能块生成的中间变量。