***Compilers***

***Principle ,Techniques, and Tools***

***Anbar University – Computer College***

**Intermediate Code Generation ( IR)**

IR is an internal form of a program created by the compiler while translating the program from a *H.L.L* to *L.L.L.*(*assembly* or *machine code*),from IR the back end ofcompiler generates *target code*.

Although a source program can be translated directly into the target language,some benefits of using a machine independent IR are:

* 1. A compiler for different machine can be created by attaching a back end for a new machine into an existing front end.
	2. Certain optimization strategies can be more easily performed on IR than on either original program or L.L.L.
	3. An IR represents a more attractive form of target code.

**Intermediate Languages:-**

1. Syntax Tree and Postfix Notation are tow kinds of intermediate representations, for example **a=b\*-c+b\*-c**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | = |  |  |  |  |  | = |  |  |  |  |  |  |  |  |
| a |  |  | + |  |  |  | a |  | + |  |  |  |  |  |  |
|  |  | \* |  |  |  | \* |  |  |  | \* |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| b | - |  | b | - | b | - |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | c |  |  | c |  |  | c |
|  |  |  |  |  |  |  |  |  |  |
|  | **Syntax Tree** |  |  |  |  | **DAG** |



* A *DAG* give the same information in syntax tree but in compact way because common subexpressions are identified.
* *Postfix notation* is a linearized representation of a syntaxtree, for example: **a b c - \* b c - \* + =**
* Two representation of above syntax tree are:

***With My Best Wishes*** *44* ***Esam***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ***Compilers*** |  |  |  |  |  |  | ***Anbar University – Computer College*** |  |
|  | ***Principle ,Techniques, and Tools*** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** | **id** | **b** |  |
|  |  |  |  | = | • |  | • |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **1** | **id** | **c** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **id** |  | **a** |  |  |  |  |  |  | **2** | **-** | **1** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **3** | **\*** | **0** | **2** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **+** | **•** | **•** |  |  |  |  | **4** | **id** | **b** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **5** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **id** | **c** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **6** | **-** | **5** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **7** | **\*** | **4** | **6** |
| **\*** | **•** |  |  |  | **•** |  |  |  |  | **\*** | **•** | **•** |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **8** | **+** | **3** | **7** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **9** | **id** | **a** |  |
| **id** |  | **b** |  |  |  |  |  | **id** |  | **b** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **10** | **=** | **9** | **8** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **….** | **…..** | **…..** |
| **-** | **•** |  |  |  |  |  |  |  |  | **-** | **•** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | **…..** | **…..** | **…..** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **id** |  |  | **c** |  |  |  |  |  | **id** |  | **c** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**1** **2**

1. Three-Address Code is a sequence of statements of the general form :

*X=Y op Z* // op is binary arithmetic

operation

For example : *x + y \* z*

**

*t1 = y \* z*

*t2 = x + t1*

where t1 ,t2 are compiler generated temporary.

***With My Best Wishes*** *45* ***Esam***

***Compilers***

***Principle ,Techniques, and Tools***

***Anbar University – Computer College***

**Types of three address code statement:-**

1. Assignment statements of the form *X=Y op Z* ( where op is a binary arithmetic or logical operator).
2. Assignment instructions of the form *X= op Y* ( op is a unary operator).
3. Copy statements of the form *X=Y* .
4. Unconditional jump ( *Goto L* ).
5. Conditional jump ( *if X relop Y goto L*).
6. *Param X* & *Call P,N* for procedure call and andreturn *Y* , for example :

Param x1 Param x2

……..

Param xn

Call P,n

1. Index assignments of the form X=Y[i] & X[i]=Y.
2. Address & Pointer Assignments

*X= &Y*

*X= \* Y*

*\*X= Y*

Example : a= b \* -c + b \* -c

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **t1** | **= - c** |  | **t1** | **= - c** |
| **t2** | **= b \* t1** |  | **t2** | **= b \* t1** |
| **t3** | **= - c** |  | **t5** | **= t2 + t2** |
| **t4** | **= b \* t3** |  | **a = t5** |
| **t5** | **= t2 + t4** |  |  |  |  |  |
| **a = t5** |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | **Three address code** |  |  |  |  | **Three address code** |  |
|  |  | **For syntax tree** |  |  |  |  | **For DAG** |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

***With My Best Wishes*** *46* ***Esam***

***Compilers***

***Principle ,Techniques, and Tools***

***Anbar University – Computer College***

Note: Three-address statements are a kin to assembly code statements can have symbolic labels and there are statements for flow of control.

**Implementation of Three Address Code :-**

In compiler , three-address code can be implement as records, with fields for operator and operands.

1. **Quadruples :-** It is a record structure with fourfields:
	* **OP //** operator
	* **arg1 , arg2 //** operands
	* **result**
2. **Triples :-** To avoid entering temporary into*ST*, wemight refer to a temporary value by position of the statement that compute it . So three address can be represent by record with only three fields:
	* **OP //** operator
	* **arg1 , arg2 //** operands

***With My Best Wishes*** *47* ***Esam***

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Compilers*** |  | ***Anbar University – Computer College*** |
|  | ***Principle ,Techniques, and Tools*** |  |  |  |  |
|  | **Example: a = b \* -c + b \* -c** |  |  |  |  |
|  |  | **i. By Quadruples** |  |  |  |
|  |  |  |  |  |  |  |  |
|  | **Position** | **OP** |  | **arg1** |  | **arg2** | **result** |
|  |  |  |  |  |  |  |  |
|  | **0** | **-** |  | **c** |  |  | **t1** |
|  | **1** | **\*** |  | **b** |  | **t1** | **t2** |
|  | **2** | **-** |  | **c** |  |  | **t3** |
|  | **3** | **\*** |  | **b** |  | **t3** | **t4** |
|  | **4** | **+** |  | **t2** |  | **t4** | **t5** |
|  | **5** | **=** |  | **t5** |  |  | **a** |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ii.** | **By Triples** |  |  |
|  |  |  |  |  |  |  |
|  | **Position** |  |  | **OP** | **arg1** | **arg2** |
|  |  |  |  |  |  |  |
|  | **0** |  | **-** | **c** |  |
|  | **1** |  |  | **\*** | **b** | **(0)** |
|  | **2** |  |  | **-** | **c** |  |
|  | **3** |  |  | **\*** | **b** | **(2)** |
|  | **4** |  |  | **+** | **(1)** | **(3)** |
|  | **5** |  |  | **=** | **a** | **(4)** |
|  |  |  |  |  |  |  |

***With My Best Wishes*** *48* ***Esam***