Parul® University

COMPILER DESIGN

SUBJECT CODE: 303105349

Vishal Singh , Assistant Professor
Computer Science & Engineering

Parul® University

DIGITAL LEARNING CONTENT

CHAPTER-5

Intermediate code generation

Parul® University

Contents

- Different intermediate representations –
  - Quadruples
  - Triples
  - Trees
  - SSA forms
- Translation of expressions
- Assignment statements

Parul® University

Intermediate Representation

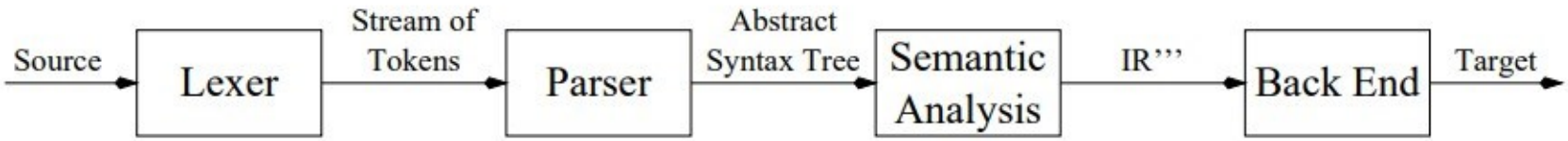


Fig 6.1: Intermediate Representation (IR)

Intermediate Representation (IR):

- An abstract machine language
- Expresses operations of target machine
- Not specific to any particular machine
- independent of source language.

Parul®
University

Intermediate Representation- Benefits

- Retargeting is facilitated
- Machine independent Code Optimization can be applied.

Parul® University

Intermediate representations

Intermediate codes can be represented in a variety of following ways and they have their own benefits.

- Three address code
  - Quadruples
  - Triples
- Trees
- SSA forms

Parul® University

Intermediate representations - Three address code

- A three-address code has at most three address locations to calculate the expression.
- A three-address code can be represented in two forms :
  - Quadruples
  - Triples

Parul® University

Intermediate representations - Three address code- Quadruples

Each instruction in quadruples presentation is divided into four fields: operator, arg1, arg2, and result.

•For example:

a = b + c * d;

is represented below in quadruples format:

Op	arg_1	arg_2	result
*	c	d	r1
+	b	r1	r2
+	r2	r1	r3
=	r3		a

Fig 6.2 Quadruples example

Parul® University

Intermediate representations - Three address code- Triples

- Each instruction in triples presentation has three fields : op, arg1, and arg2.
- The results of respective sub-expressions are denoted by the position of expression.
- Triples represent similarity with DAG and syntax tree. They are equivalent to DAG while representing expressions.

Op	arg_1	arg_2
*	c	d
+	b	(0)
+	(1)	(0)
=	(2)	

Fig 6.3 Triples example

- Triples face the problem of code immovability while optimization, as the results are positional and changing the order or position of an expression may cause problems.

Parul® University

Intermediate representations - Trees

- Syntax tree is a variant of the parse tree, where each leaf represents an operand and each interior node represent an operator.
- A sentence a * (b +d) would have the following syntax tree

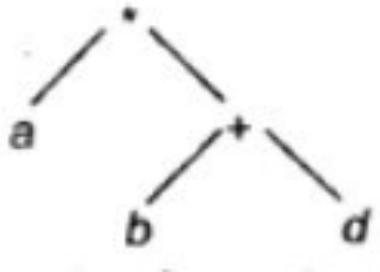


Fig 6.4 Example of syntax tree

Parul® University

Intermediate representations - SSA

•Static single-assignment form (SSA) is an intermediate representation that facilitates certain code optimizations.

•The first is that all assignments in SSA are to variables with distinct names; hence the term *static single-assignment.*

Intermediate program in three-address code and SSA

p = a + b	p1 = a + b
q = p - c	q1 = P1 - c
p = q * d	p2 = q1 *d
P = - P	P3 = -P2
q = p + q	q2= p3 + q1
(a) Three-address code.	(b) Static single-assignment form.

Parul® University

Translation of expressions and assignment statements

•In the syntax directed translation, assignment statement is mainly deals with expressions. The expression can be of type real, integer, array and records.

Consider the grammar

S → id := E
E → E1 + E2
E → - E1
E → (E1)
E → id

•The translation scheme of above grammar is given below:

Parul® University

DIGITAL LEARNING CONTENT

Translation of expressions and assignment statements

PRODUCTION	SEMANTIC RULES
$S \rightarrow \textbf{id} = E\ ;$	$S.code = E.code \mid\mid gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \textbf{new}\ Temp()$ $E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ - E_1$	$E.addr = \textbf{new}\ Temp()$ $E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \textbf{id}$	$E.addr = top.get(\textbf{id}.lexeme)$ $E.code = ''$

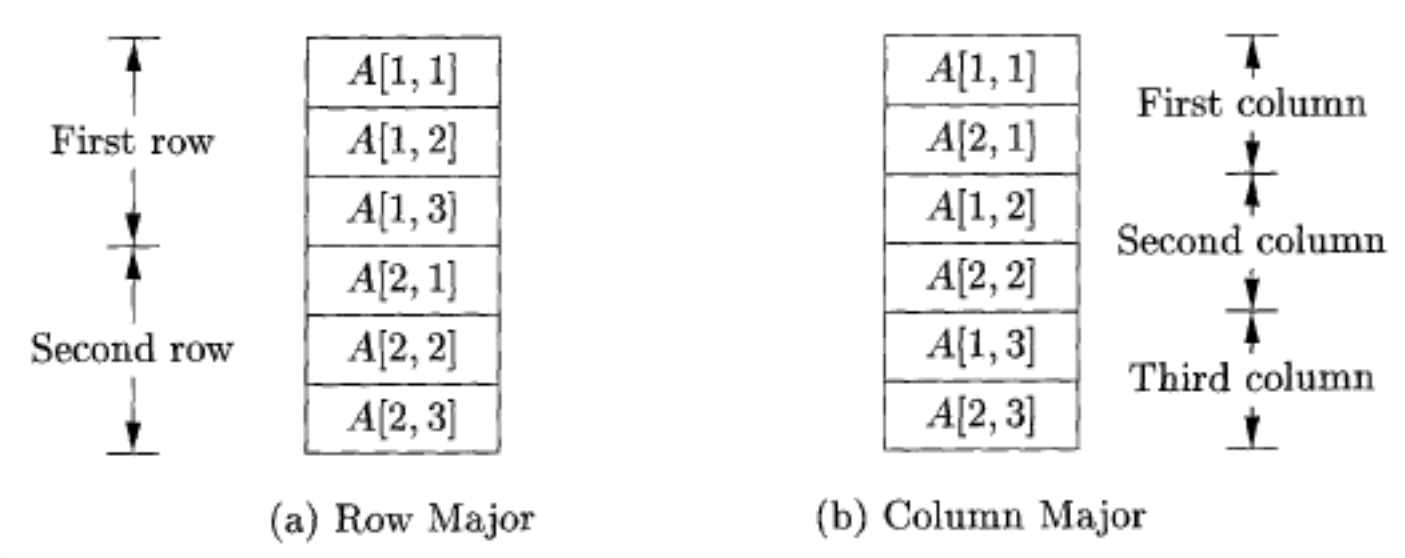
Fig 6.5 Three-address code for expressions

Parul® University

DIGITAL LEARNING CONTENT

Addressing Array Elements

Layouts for a two-dimensional array:



Parul® University

Semantic actions for array reference

S	$\rightarrow$	**id** = E ;	{ $gen($ $top.get($**id**$.lexeme)$ $'='$ $E.addr)$; }
	|	L = E ;	{ $gen(L.addr.base$ $'['$ $L.addr$ $']'$ $'='$ $E.addr)$; }
E	$\rightarrow$	E_1 + E_2	{ $E.addr =$ **new** $Temp\,()$; $gen(E.addr$ $'='$ $E_1.addr$ $'+'$ $E_2.addr)$; }
	|	**id**	{ $E.addr = top.get($**id**$.lexeme)$; }
	|	L	{ $E.addr =$ **new** $Temp\,()$; $gen(E.addr$ $'='$ $L.array.base$ $'['$ $L.addr$ $']')$; }

Parul® University

DIGITAL LEARNING CONTENT

Semantic actions for array reference

$$
\begin{aligned}
L \rightarrow \ & \mathbf{id}\ [\ E\ ] \quad \{\ L.array = top.get(\mathbf{id}.lexeme); \\
& \quad L.type = L.array.type.elem; \\
& \quad L.addr = \mathbf{new}\ Temp\,(); \\
& \quad gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \} \\
\mid \ & L_1\ [\ E\ ] \quad \{\ L.array = L_1.array; \\
& \quad L.type = L_1.type.elem; \\
& \quad t = \mathbf{new}\ Temp\,(); \\
& \quad L.addr = \mathbf{new}\ Temp\,(); \\
& \quad gen(t\ '='\ E.addr\ '*'\ L.type.width);\ \} \\
& \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{aligned}
$$

Fig 6.6 Semantic actions for array references

Parul® University

Semantic actions for array reference

Nonterminal *L* has three synthesized attributes:

- *L.addr*
- L.array
- L.type

Parul® University

DIGITAL LEARNING CONTENT

CHAPTER-6

Intermediate code generation

Parul® University

Flow of control statements

There are 4 types of control statements: Each contains at least one Boolean condition, based on the condition next statement will decide. So we may need to do an unconditional jump from one statement to another to perform these control statements.

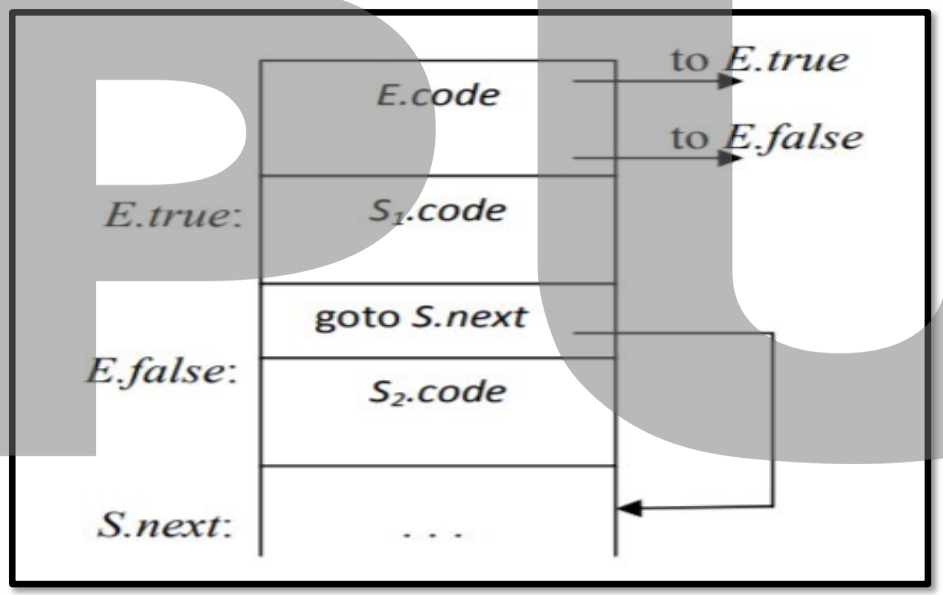
1. If..then..else...
2. If.. then...
3. While..do…
4. Case statement

Each control statement has one or more Boolean expressions. So let name that Boolean expression as E. so if E is true then control transfer to the E.True label. And if E is false then control transfer to the E.False label.

Parul® University

DIGITAL LEARNING CONTENT

1. IF E then S1 else S2



Parul® University

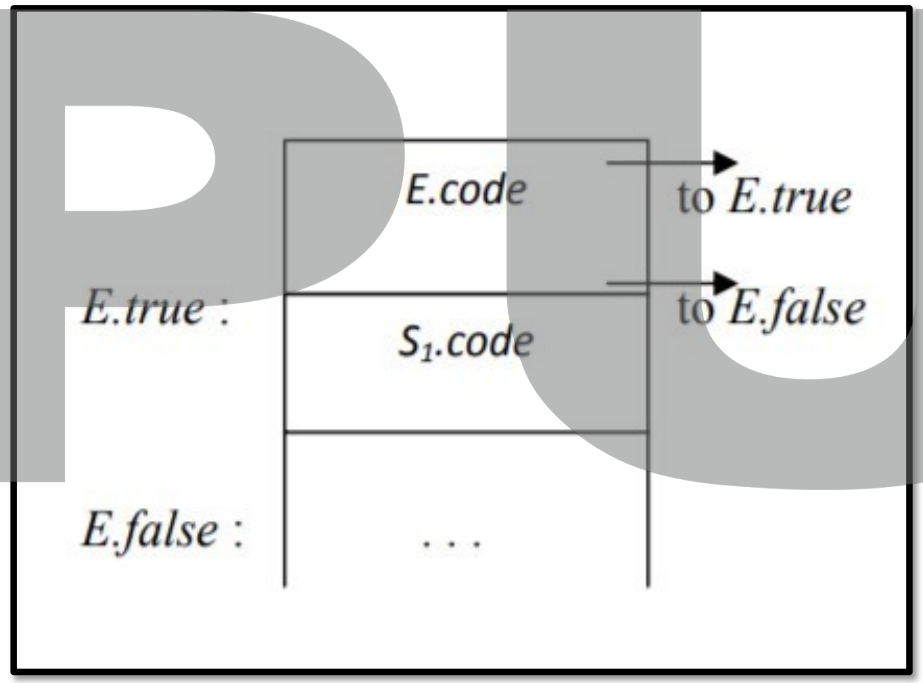
DIGITAL LEARNING CONTENT

1. IF E then S1 else S2

Convert this code into Three address code

If(a<b)
x=y+z
else:
p=q+r
	1. If a<b go to (3)
	2. Go to (6)
	3. t1 = y+z
	4. x=t1
	5. go to 9
	6. t2=q+r
	7. p=t2
	8. go to 9
	9. next statement

2. IF E then S1

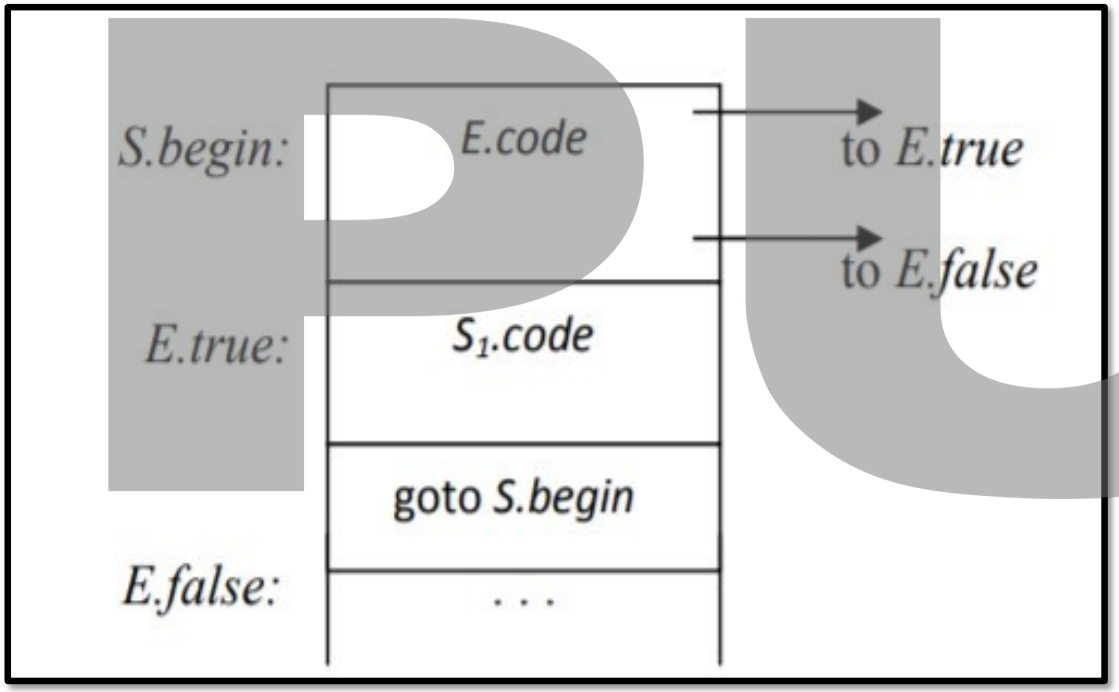


2. IF E then S1

Convert this code into Three address code

If(a<b) x=y+z	1. If a<b go to (3) 2. Go to (5) 3. t1 = y+z 4. x=t1 5. next statement

3. While E do S1



Parul® University

DIGITAL LEARNING CONTENT

3. While E do S1

Convert this code into Three address code

Int i=0
While (i<=10)
printf(i=%d,i);
i++;
 	
…
	1. i=0
	2. if i<=10 go to 4
	3. go to 7
	4. print i
	5. i=i+1
	6. go to 2
	7. next statement

4. Case Statement

```
switch expression
    begin
        case value :   statement
        case value :   statement
                ...
        case value :   statement
        default :      statement
    end
```

```
                code to evaluate E into t
                goto test
L1 :            code for S1
                goto next
L2 :            code for S2
                goto next
                . . .
Ln-1 :          code for Sn-1
                goto next
Ln :            code for Sn
                goto next
test :          if t = V1 goto L1
                if t = V2 goto L2
                . . .
                if t = Vn-1 goto Ln-1
                goto Ln
next :
```

Parul® University

4. Case Statement

When you see keyword switch then we have two new labels test label and next label and also t temporary variable is generated. Based on the evaluation of E, the t value is generated. That t value helps to get the best-suited case for the t value. Test label has all the possible t value and tests whether t value matches with any case. After completion of all statements all control transfer to the next label which has the next statements for evaluation

Parul® University

DIGITAL LEARNING CONTENT

4. Case Statement

Convert this code into Three address code

Switch(i+j)
{
Case 1: x=y+z
Case 2: p=q+r
Default: u=v+w
}
	1. t1=i+j
	2. go to 9
	3. t2=y+z
	4. x=t2
	5. t3=q+r
	6. p=t3
	7. t4=v+w
	8. u=t4
	9. if t1=1 go to 3
	10. if t1=2 go to 5
	11. next statement

Parul® University

DIGITAL LEARNING CONTENT

Short-Circuit Code

When We place multiple Boolean operator like “and,” “or” and “not” in one statement and that can also translate a boolean expression into three-address code without generating code for any of the boolean operators and without having the code necessarily evaluate the entire expression. This style of evaluation is sometimes called “short-circuit” or “jumping” code.

For example,

If(a>b or c<d and e>=f)

If three address code generate for just a>b and go to next statement without considering entire statement

Parul® University

DIGITAL LEARNING CONTENT

Boolean Expressions

Boolean expressions have two primary purposes. They are used to compute logical values, but more often they are used as conditional expressions in statements that alter the flow of control, such as if-then-else, or while-do statements.

Boolean expressions are composed of the boolean operators (and, or, and not) applied to elements that are boolean variables or relational expressions. Relational expressions are of the form E1 relop E2, where E1 and E2 are arithmetic expressions.

Here we consider boolean expressions generated by the following grammar :

E →E or E | E and E | not E | (E) | id relop id | true | false

Parul® University

Boolean Expressions

Methods of Translating Boolean Expressions: There are two principal methods of representing the value of a boolean expression. They are :

- To encode true and false numerically and to evaluate a boolean expression analogously to an arithmetic expression. Often, 1 is used to denote true and 0 to denote false.
- To implement boolean expressions by flow of control, that is, representing the value of a boolean expression by a position reached in a program. This method is particularly convenient in implementing the boolean expressions in flow-of-control statements, such as the if-then and while-do statements.

Parul® University

Backpatching

The easiest way to implement the syntax-directed definitions for Boolean expressions is to use two passes. First, just construct a syntax tree for the input, and then just walk the tree in depth-first order and compute the translations.

The main problem for generating code for Boolean expressions and flow of control statements in one pass or single pass is that in the process of singles pass we may not know the labels for go to or jump statements that control must go to a particular statement if there is a jump statement.

Therefore, a series of branching statements with the targets of the jumps left unspecified is generated. Each statement will be put on a list of goto statements whose labels will be filled in when the proper label can be determined. We call this subsequent filling in of labels backpatching.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in