compiler design u.4

classmate
Date
Page

Intermediate Code generation.

Q.1 Three Address Code

$$\begin{tikzcd} \text{lexical Analysis} \arrow[r] & \text{Syntax Analysis} \arrow[r] & \text{Semantic Analysis} \arrow[d] \\ \text{Machine code Generation} & \text{Code optimization} \arrow[l] & \text{Intermediate Code Generation} \arrow[l] \end{tikzcd}$$

6marks

Various forms of Intermediate Code Genration

① Linear form
a. Prefix
b. Postfix
c. 3 address code.

② Tree form
a. Syntax Tree
b. DAG (Directly Acyclic) Graph

a. Prefix Notation

a+b ⇒ +ab
(a+b)*c ⇒ *(+ab) c
(a+b) * (c-d) ⇒ *(+ab) (-cd)

b. Postfix Notation

a+b ⇒ ab+
(a+b)*c ⇒ (ab+) c*
(a+b) * (c-d) ⇒ (ab+) (cd-) *

c. 3 address code :

x op z → (variable operator variable)
op z → operator variable

ex → x = a + b * c + d

1. T_1 = b * c
⇒ a + T_1 + d
2. T_2 = a + T_1
⇒ T_2 + d
3. T_3 = T_2 + d
4. x = T_3

ex → $a = (b * -c) + (b * -c)$

1. $T_1 = -c$ $\{ a = (b * T_1) + (b * -c)$
2. $T_2 = b * T_1$
3. $T_3 = -c$
4. $T_4 = b * T_3$
5. $T_5 = T_2 + T_4$
6. $x = T_5$

* Implentation of 3 address code
    ↳ form of intermediate code
    ↳ Repsntations are:
        ① Quadruples
        ② Triples
        ③ Indirect Triples.

① Quadruples:
    ↳ has 4 fields
    ↳ op, arg 1, arg 2 and resut.

② Triples:
    ↳ has 3 fields
    ↳ op, arg 1, arg 2

③ Indirect Triples:
    uses pointers for repsntation

example To find quadruples, triples, Indirect Triples.

S-1 Convert to Three Address code

$a = (b * -c) + (b * -c)$

1. $T_1 = -c$

   $a = (b * T_1) + (b * -c)$

2. $T_2 = b * T_1$

   $a = T_2 + (b * -c)$

3. $T_3 = -c$

   $a = T_2 + (b * T_3)$

4 - $T_4 = b * T_3.$

   $a = T_2 + T_4$

5. $T_5 = T_2 + T_4$

6. $\boxed{a = T_5.}$

Three address code

① operand opr opnd

② operator opnd

Quadruples

#	op	arg 1	arg 2	result ~~arg 3~~
(1)	-	c		T_1
(2)	*	b	T_1	T_2
(3)	-	c		T_3
(4)	*	b	T_3	T_4
(5)	+	T_2	T_4	T_5
(6)	=	T_5		a

↳ leave blank (—) or (=)

Triples

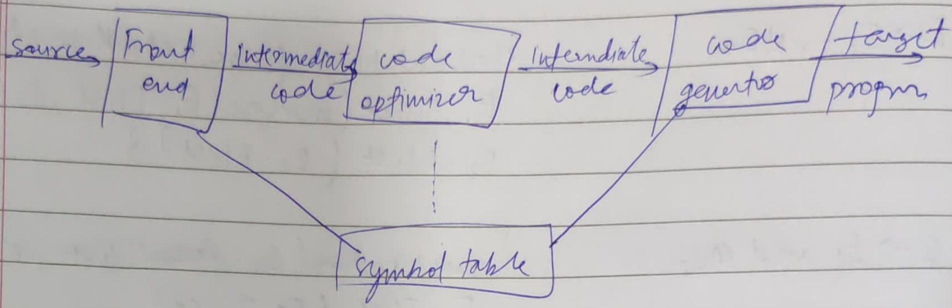
#	op	arg 1	arg 2
(1)	-	c	
(2)	*	b	(1)
(3)	-	c	
(4)	*	b	(3)
(5)	+	(2)	(4)
(6)	=	a	(5)

Indirect Triples

#	Reference Istmt
(1)	13
(2)	14
(3)	15
(4)	16
(5)	17
(6)	18

Code Generation

Properties: (1) high performance
(2) correctness
(3) efficient
(4) quick code generation



Issues: (1) Input target problems (Notations)
(2) code generation (
(3) Memory Management (Mapping values)
(4) Instruction selection (Assembly code)
(5) Register allocation (Use of registers)

example Q- $x = y + z$

```
MOV  y, RO
ADD  z, Ro
MOV  Ro, x
```

Q $a = b + c$
$d = a + e$

```
MOV  b, Ro
ADD  c, Ro      →      MOV  b, Ro
MOV  Ro, a             ADD  c, Ro
MOV  a, Ro             ADD  e, Ro
ADD  e, Ro             MOV  Ro, d
MOV  Ro, d
```

Code generation Algorithm:

Q $T = (a-b) + (a-c) + (a-c)$

convert to three address code (Intermediate Code.

$T_1 \Rightarrow (a-b)$

$T = T_1 + (a-c) + (a-c)$

$T_2 \rightarrow (a-c)$

$T = T_1 + T_2 + (a-c)$

$T_3 \rightarrow (a-c)$

$T = T_1 + T_2 + T_3$

$T_4 \Rightarrow T_1 + T_2$

$T = T_4 + T_3$

$T_5 \rightarrow T_4 + T_3$

$T \rightarrow T_5$

$T_1 \rightarrow (a-b)$

$T_2 \rightarrow (a-c)$

$T_3 \rightarrow T_1 + T_2$

$T_4 \rightarrow T_3 + T_2$

$T \rightarrow T_4$

Assembly code

```
MOV a, R0
SUB b, R0
MOV a, R1
SUB c, R1
ADD R1, R0
ADD R1 R0
MOV R0, T
```

Stat	Code Generated	Register Descriptor	Addr. Descriptor
(1) $T_1 = a-b$	MOV a, R0	R0 has T_1	T_1 in R0
	SUB b, R0		
(2) $T_2 = a-c$	MOV a, R1	R0 has T_1	T_1 in R0
	SUB c, R1	R1 has T_2	T_2 in R1
(3) $T_3 = T_1 + T_4$	ADD R1, R0	R0 has T_3	T_3 in R0
		R1 has T_2	T_2 in R1
(4) $T_4 = T_3 + T_2$	ADD R1 R0	R0 has T_4	T_4 in R0
		R1 has T_2	T_2 in R1
(5) $T = T_4$	MOV R0, T		

classmate
Date
Page

④ **Backpatching**

↳ A process which is used to fulfill the infos. used during code generation
↳

Rules:

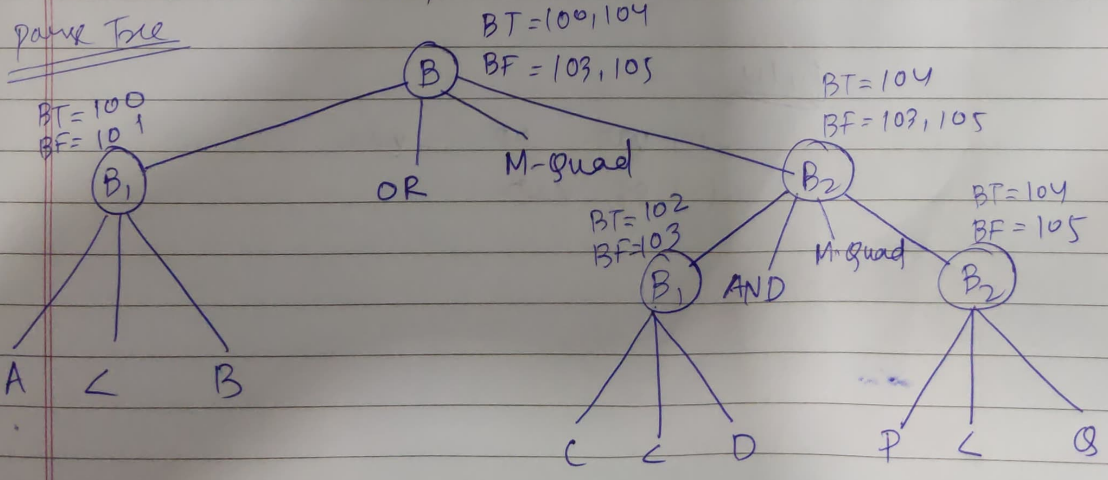
Production Rule	Semantic Action
B → B_1 || B_2	{ backpatching(E_1 · flist, M · instr) E_1 · Tlist(merge(E_1.Tlist, E_2-Tlist)) E_2 · Flist(E_2 · Flist) }
B → B_1 && B_2	{ backpatching(E_1 · Flist, M · instr) E_1 · Tlist(E_2 · Tlist) E_2 · Flist(merge(E_1 · Flist, E_2 · Flist)) }

Three Address Code Using Backpatching

A < B || C < D && P < Q

```
100 : if A < B go to 106
101 : else go to 102
102 : if C<D go to 104
103 : else go to 107
104 : if P<Q go to 106
105 : else go to 107
106 : True statement
107 : False stmt
```

Parse Tree



Ans

① $x = a * - (b + c)$

① $T_1 = b + c$
$x = a * - T_1$

② $T_2 = - T_1$
$x = a * T_2$

③ $T_3 = a * T_2$

④ $x = T_3$

For Triply

#	op	arg 1	arg 2
(1)	+	b	c
(2)	–	(1)	
(3)	*	a	(2)

② $x = * y$ $a = \& x$

① $T_1 = * y$
② $x = T_1$

① $T_1 = \& x$
② $a = T_1$

Rules

(1) var op. var
(2) op var