Compiler Construction

BITS Pilani
Pilani Campus

Vinti Agarwal
March 2021

innovate achieve lead

BITS Pilani
Pilani Campus

CS F363, Compiler Construction

Lecture topics: Three Address code generation

Statements in three-address code

- assignments
- jumps
- pointer and address assignments
- procedure call/returns
- miscellaneous

Procedure calls and returns

- A call to procedure p($x_{1,}$ $x_{2,....,}$ x_n) can be written as sequence of three address instructions:

param x_1

param x_2

param x_n

enter f

leave f

return

return x

Miscellaneous statements

- More statements may be needed depending upon the requirement of a language
- One example is “next, break, continue statements”

```
goto L
    -
    -
label L
```

Examples

op op1, op2, result

e.g. $\boxed{x = y + z}$ ⇒ ADD y, z, x

⇒ $x + (-y)$

$$\boxed{\begin{array}{l} \text{Uminus } y, , t_1 \\ \text{ADD } x, t_1, t_2 \end{array}}$$

$-y$

$x + t_1$

Examples

if y relop z goto L

JMPGT y, z, addr.of(L)

if x != 0 goto L

JMPNZ x, _, addr(L)

Examples

Examples

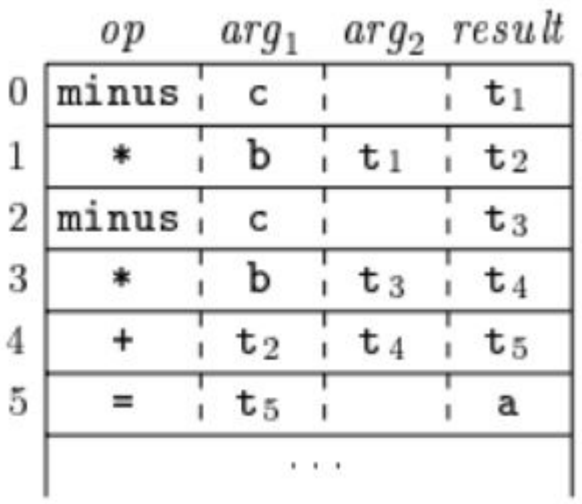
Examples

Implementation of Three address code

- Quadruples ✓
- triples ✓
- indirect triples ✓

Quadruples

- has four fields: op, arg_1, arg_2, result
- e.g. a = b*-c + b*-c



	op	arg_1	arg_2	$result$
0	minus	c		t_1
1	*	b	t_1	t_2
2	minus	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
	. . .			

Quadruples

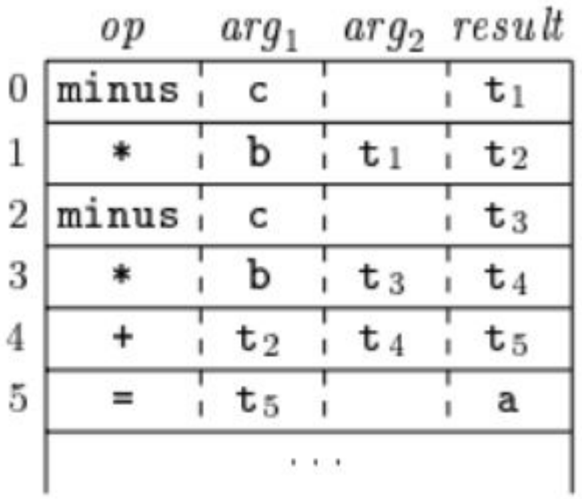
Examples

1. x = y + z
2. if t1 >= t2 goto L

op	ADD	JMPGE
arg1	y	t1
arg2	z	t2
result	x	dest(L)

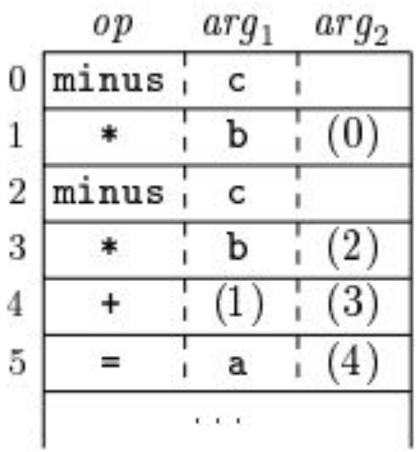
Triples

- has only three fields: op, arg_1, arg_2
- result of an operation is refer by its position



	op	arg_1	arg_2	$result$
0	minus	c		t_1
1	*	b	t_1	t_2
2	minus	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
		. . .		

Quadruples



	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
		. . .	

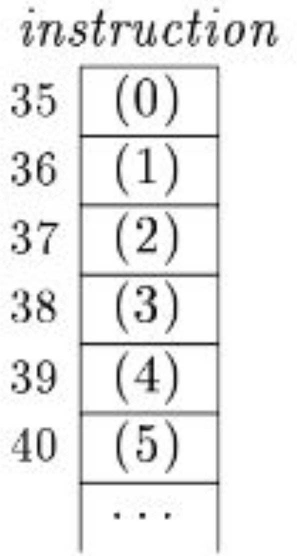
Triples

Quadruples vs Triples

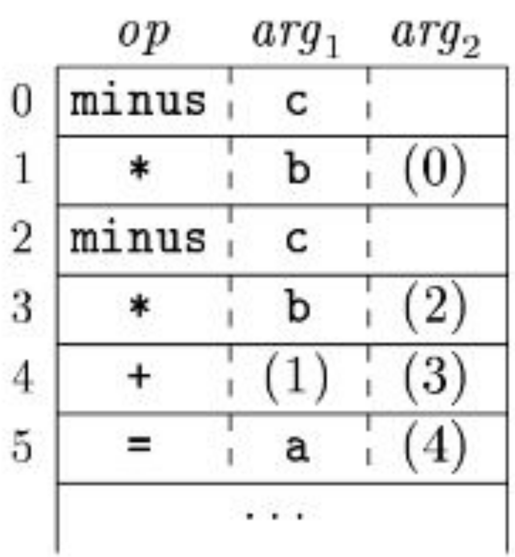
- optimization is easy in quadruples than triples
- if an instruction that computes a temp t moves, required no change in other instructions that use t.

Indirect Triples

- list pointers to triples, rather than listing triples only.



	instruction
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)
	. . .



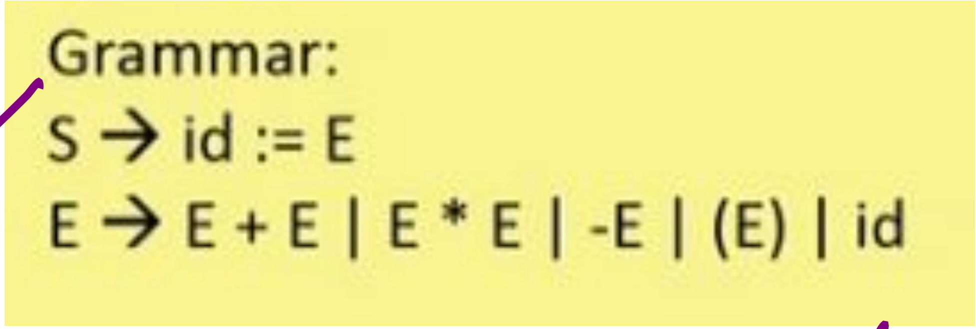
	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
		. . .	

SDD

Three address code generation

- Using syntax directed translation of productions
- Parse Tree

  Semantic Rules



Grammar:

S → id := E

E → E + E | E * E | -E | (E) | id

root

- Attributes for non terminal S: S.code
- Attributes for non terminal E: E.place, E.code

leaf

- Attributes for terminal id: id.place

Three address code generation

Grammer Rule	Semantic Actions.
1 $S \rightarrow id := E$	$S.code := E.code \parallel gen(id.place = E.place)$
2 $E \rightarrow E_1 + E_2$	$E.place := newTemp(\ )$ $E.code := E_1.code \parallel E_2.code \parallel gen(E.place := E_1.place + E_2.place)$
3 $E \rightarrow E * E_2$ unary	
4 $E \rightarrow -E_1$	$E.place := newTemp(\ )$ $E.code := E_1.code \parallel gen(E.place := uminus\ E_1.place)$
5 $E \rightarrow (E_1)$	$E.place := E_1.place$ $E.code := E_1.code$
6 $E \rightarrow id$	$E.place := id.place$ $E.code :=$

Examples

$x := (y + z) * (-w + v)$

Parse Tree.

① E.place = y

② E.place = z

③ E.place = t_1 →

E.code = E.place := E_1.place + E_2.place

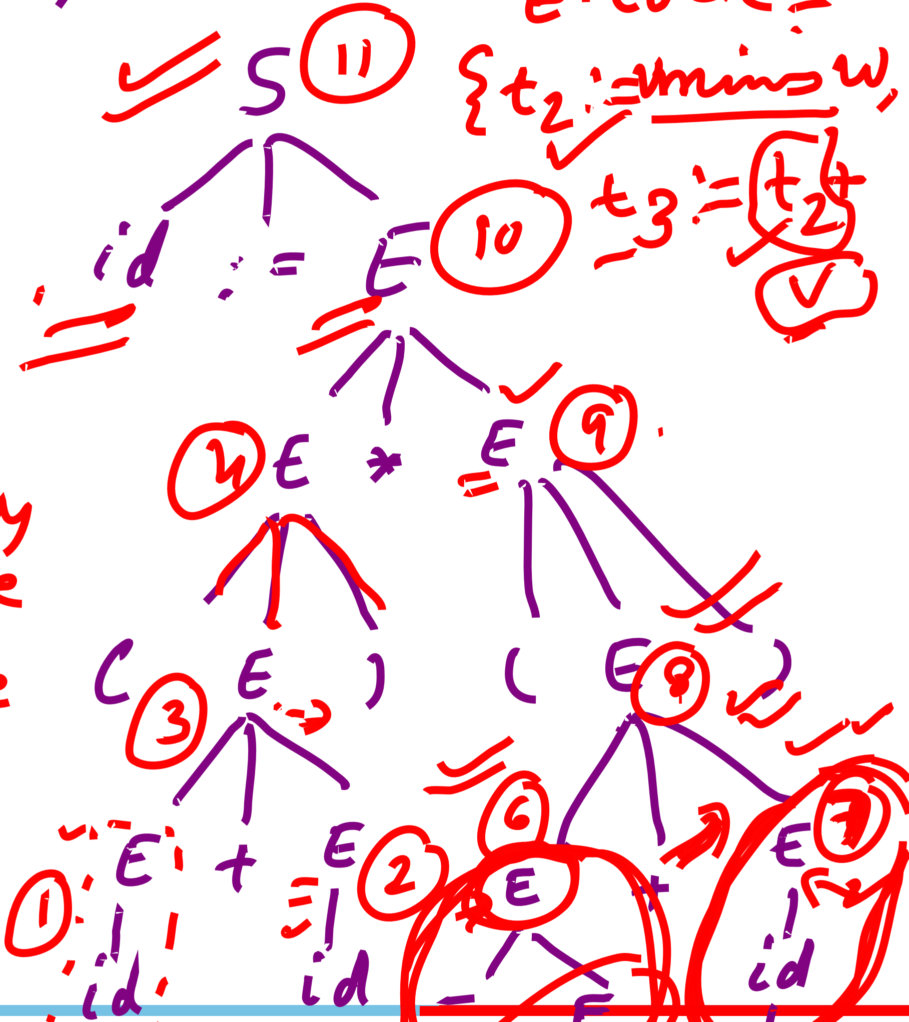
$= \boxed{t_1 := y + z}$

④ E.place = t_1

E.code = { t_1 := y+z }

E → id

E.code = ''



⑦ E.place = v

⑧ E.place = t_3

E.code = { t_2 := uminus w, t_3 := t_2 + v

Three address code generation

5 E.place = w

E.place : t_2 , E.code : t_2 := uminus w

new Temp() : — returns a unique new temporary var.

gen: produces a 3-addr code for exp.

|| : concatenates two 3-addr. codes.

S.code := { t_1 := y+z, t_2 := uminus w, t_3 := t_2+v, t_4 := t_1 * t_3, x := t_4 }

Three address code for expression

x:=(y+z)*(-w+v)

e.g. (a = b + -c)

→ Three add code using above semantic rules.

Home Exercise.

Three address code for expression

Three address code for expression

Three address code for expression

Thank You!