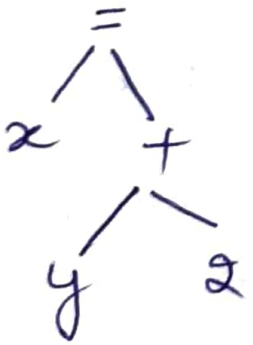
Intermediate Code Generation

→ Machine Independent
→ Abstract Syntax tree
→ Direct Acyclic Graph
→ Postfix
→ 3- address code

$x = y + 2$



Abstract Syntax Tree

Postfix

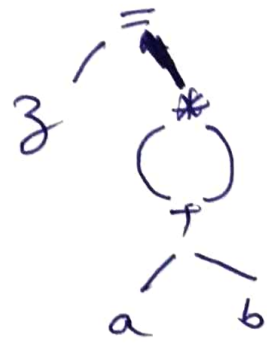
$x * y + 3$

$y 3 + x *$

Direct Acyclic Graph

DAG

$z = (a+b) * (a+b)$



Can contain cycle

Representation of 3-addr code

$-(a \times b) + (c \times d + e)$

$t_1 = a \times b$
$t_2 = -t_1$
$t_3 = c \times d$
$t_4 = t_3 + e$
$t_5 = t_2 + t_4$

① Quadruples

	operator	op1	op2	result
0	×	a	b	t_1
1	−	t_1		t_2
2	×	c	d	t_3
3	+	t_3	e	t_4
4	+	t_2	t_4	t_5

Adv.
→ Statements can be moved in Quadruples
2, 0, 1

Disadv.
⇒ consume more space Quadruple

② Triples

	operator	op1	op2
0	×	a	b
1	−	(0)	
2	×	c	d
3	+	(2)	e
4	+	(1)	(3)

Adv.
1) Less space consumption

Disadv.
1) Can't move statements

③ Indirect Triples

100	(0)
101	(1)
102	(2)
103	(3)
104	(4)

1) Adv
Stmt can be moved
2, 0, 1

2) Disadv.
2 memory references

Adv. Indirect triples
Can save space if same temp. value is used more than once.

Code optimization

Platform Dependent Tech

- → Peephole optimization
- → Instruction level Parallelism (pipeline)
- → Data level parallelism (Distributed Databases)
- → Cache optimization (Creating levels of cache)
- → Redundant Resources (Make use of more registers to enhance speed (execution))

Platform Independent Techniques

- → Loop optimization
- → constant folding
- → Constant Propagation
- → Common Subexpression Elimination

Peephole optimization

① Remove Redundant load and store

```
a = b + c
d = a + e
MOV b, R0
ADD c, R0
MOV R0, a      } Redundant
MOV a, R0      }
Add e, R0
MOV R0, d
```

② Strength Reduction

$x^2 \Rightarrow x * x$ | $x \times 2 \Rightarrow$ left shift x
$x/2 \Rightarrow$ right shift x

③ Simplify Algebraic Expressi

```
a = a + 0
a = a * 1
a = a / 1
a = a - 0
```

All these expressions can be eliminate

Peephole optimization continued.....

④ Replace slower instructions with faster.

Add #1, R0 ⇒ INC R0

Sub #1, R0 ⇒ DEC R

In Java:

```
a load x
a load x   } ⇒   a load x
MUL              DUP
                 MUL
```

⑤ Dead Code Elimination

y=3 } ⇒ Dead code
x=1

```
if (x=1)
  {
  Printf("Hello")

elseif (x=0)
  Print ('Hi")
else
  Print ('Never")
```

Dead code