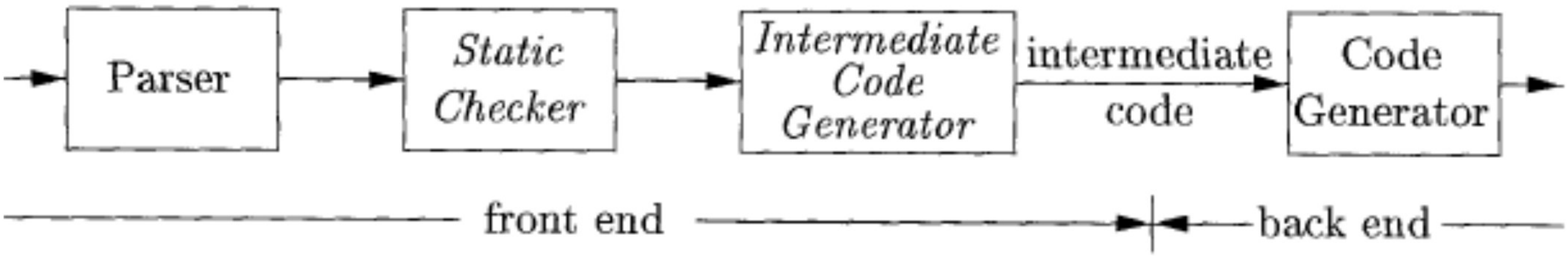
Intermediate-Code Representation

outline



Intermediate representations: syntax trees and three-address code.

High-level representations are close to the source language and

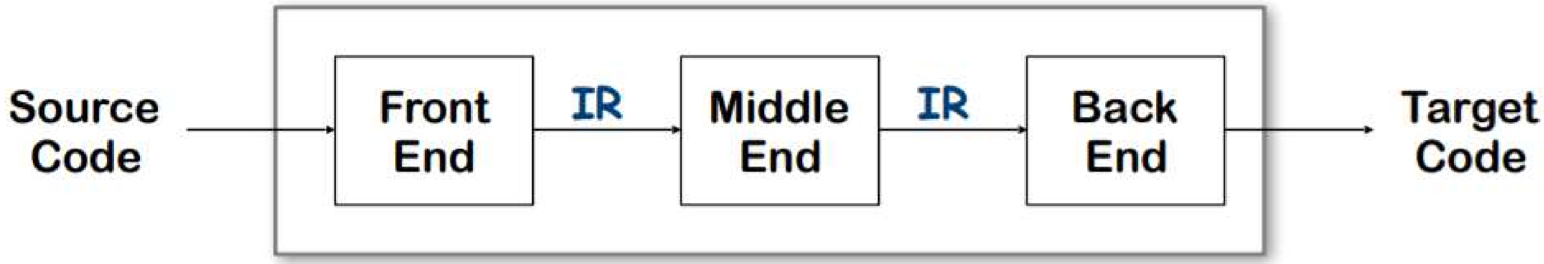
low-level representations are close to the target machine.

outline

Source Program $\longrightarrow$ High Level Intermediate Representation $\longrightarrow \cdots \longrightarrow$ Low Level Intermediate Representation $\longrightarrow$ Target Code

- The choice or design of an intermediate representation varies from compiler to compiler.
- C is a programming language, yet it is often used as an intermediate form because it is flexible.

Intermediate Representations



- Front end - produces an intermediate representation (IR)
- Middle end - transforms the IR into an equivalent IR that runs more efficiently
- Back end - transforms the IR into native code

- IR encodes the compiler's knowledge of the program
- Middle end usually consists of several passes

Intermediate Representations

- Decisions in IR design affect the speed and efficiency of the compiler
- Some important IR properties
  - Ease of generation
  - Ease of manipulation
  - Procedure size
  - Freedom of expression
  - Level of abstraction
- The importance of different properties

Types of Intermediate Representations

Three major categories

- Structural
  - Graphically oriented
  - Heavily used in source-to-source translators
  - Tend to be large

  Examples:
  Trees, DAGs

- Linear
  - Pseudo-code for an abstract machine
  - Level of abstraction varies
  - Simple, compact data structures
  - Easier to rearrange

  Examples
  3 address code
  Stack machine code

- Hybrid
  - Combination of graphs and linear

  Example: Control-flow graph

Three-Address Code

In three-address code, there is at most one operator on the right side of an instruction.

$$\boxed{x+y*z} \longrightarrow \boxed{\begin{array}{l} t_1 = y * z \\ t_2 = x + t_1 \end{array}}$$

Three-address code is a linearized representation

Three-address code is built from two concepts: addresses and instructions.Addresses:

A name

A constant

A compiler-generated temporary

- Advantages:
  - Resembles many real machines
  - Introduces a new set of names
  - Compact form

Three address code instructions

Assignment Statement

Assignment instructions of the form

x = y op z

x = op y, where op is a unary operation..

x = y, copy statement

c = a + -b

t1 = minus b

t2 = a + t1

c = t2

Example

x= a + b * c + d

Three address code instructions will be

t1 = b *c

t2 = a + t~~3~~;

t4 = t2 + d

x= t4

Three address code instructions

- Jump Statements
  - An unconditional jump *goto* L.
  - Conditional jumps of the form

    if x *goto* L

    if False x *goto* L.
- Conditional jumps such as

  if x relop y *goto* L.

```
if (a < b + c)
 a = a - c;
c = b * c;
```

```
    t1 = b + c;
    t2 = a < t1;
    If False t2 goto L0;
        t3 = a - c;
        a = t3;
L0: t4 = b * c;
        c = t4;
```

Three address code instructions

Procedure calls and returns

A procedure call like P(a1, a2,..., an) uses the following instructions: param x for parameters; call p , n and y = call p,n and return y.

param a1

param a2

-

-

param an

call P , n

y = sum(a, b)

TAC

param a

param b

y = call sum, 2

Three address code instructions

Indexed copy instructions of the form x = y [i] and x [i] = y.

Address and pointer assignments of the form x = & y, x = * y, and * x = y.

Example : Consider the statement

do i = i+l; while (a[i] < v) ;

```
L:   t1 = i + 1
     i = t1
     t2 = i * 8
     t3 = a [ t2 ]
     if t3 < v goto L
```

(a) Symbolic labels.

```
100:   t1 = i + 1
101:   i = t1
102:   t2 = i * 8
103:   t3 = a [ t2 ]
104:   if t3 < v goto 100
```

(b) Position numbers.

Representation of Three Address code

- Quadruples
  - Has four fields op, agr1, arg2 and result.
- Triplets
  - Result is not used instead of references of instructions are made.
- Indirect Triplets
  - In addition to triplets we use a list of pointers to triplets.

Quadruples

- A quadruple (or just "quad') has four fields, which we call *op*, *arg1*, *arg2* , and *result*.
- The *op* field contains an internal code for the operator.

$\boxed{x = y + z}$

placing **+** in op, y in arg1, z in arg2, and x in result.

Op	arg1	arg2	result
+	y	z	x

1. Instructions with unary operators like x = minus y or x = y do not use arg2.
2. For a copy statement like x = y, op is =.
3. Operators like param use neither arg2 nor result.
4. Conditional and unconditional jumps put the target label in result.

Quadruples

- **Example** : Three-address code for the assignment
  a = b * - c + b * - c

t_1 = minus c
t_2 = b * t_1
t_3 = minus c
t_4 = b * t_3
t_5 = t_2 + t_4
a = t_5

(a) Three-address code

	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
	...			

(b) Quadruples

Naïve representation of three address code

- Table of k * 4 small integers
- Simple record structure
- Easy to reorder
- Explicit names

The original FORTRAN compiler used "quads"

Triples

- A triple has only three fields, which we call op, arg1, and arg2

- we refer to the result of an operation x op y by its position, rather than by an explicit temporary name.

Triples

Example : The triples in Figure correspond to the three-address code

a = b * - c + b * - c

t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
a = t5

	op	*arg*$_1$	*arg*$_2$
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
	...		

(address)

invali-

(b) Triples

- Index used as implicit name
- 25% less space consumed than quads
- Much harder to reorder

Indirect Triples

- Indirect triples consist of a listing of pointers to triples, rather than a listing of triples themselves.
- With indirect triples, an optimizing compiler can move an instruction by reordering the instruction list, without affecting the triples themselves.

	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)
	...		

Figure: Indirect triples representation of three-address code