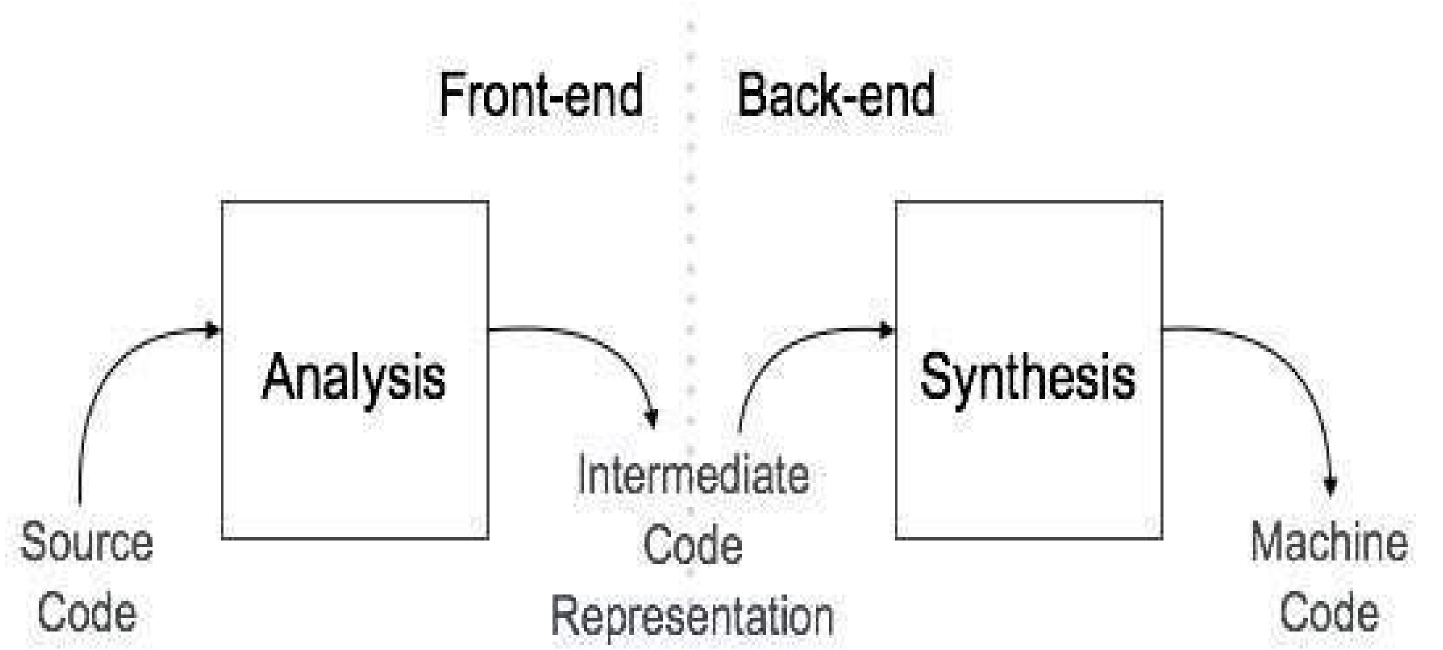
GLOBAL EDGE
Intelligence Of Things

Compilation(Synthesis)

Suvarna P Biliki

Contents

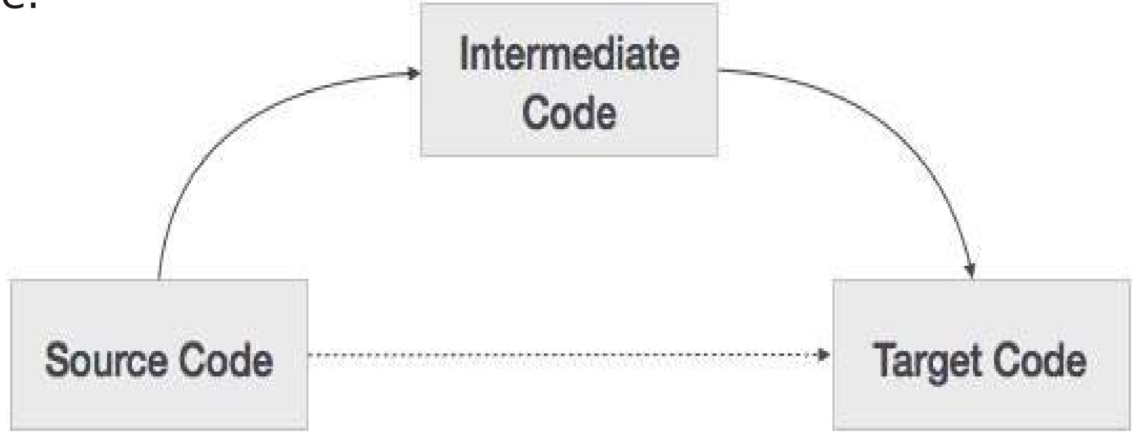
Compilation phases



Intermediate Code Generation

Intermediate code is in between the high-level language and the machine language.

This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.



Why intermediate code generation ?

- If a compiler translates the source language to its target machine language without having the option for generating intermediate code,then for each new machine,a full native compiler is required.
- Intermediate code eliminates the need of a new full compiler for every unique machine by keeping the analysis portion for all the compiler.
- It becomes easier to apply source code modification to improve code performance by applying code optimization technique on the intermediate code.

Intermediate Representation

- High Level IR

  High level IR is very close to source language.

  It can be easily Generated from the source code and we can easily apply code modification to enhance performance.

- Low Level IR

  Low Level IR is close to target machine which makes it suitable for register and memory allocation,instruction set selection.

  It is Good for machine dependent Optimization.

Three Address Code

A Three address code has at most three address locations to calculate the expression,it can be represented in two forms.

- Quadruples
- Triples

Quadruples

- Each instructions in Quadruples presentation is divided into 4 fields operator,arg1,arg2,and Result.

  Ex:a = b + c * d;

- Generator will divide this expression into sub expression and generate corresponding code.

```
t1 = c * d;
t2 = b + t1;
t3 = t2 + t1;
a =t3;
```

where t1, t2, t3 being used as temporary variables.

Quadruples Representation

operator	argument1	argument2	result
*	c	d	t1
+	b	t1	t2
+	t2	t1	t3
=	t3		a

Triples Representation

- Each instruction in Triples presentation has three fields:Operator, argument1, argument2.

- The Result of respective sub-expression are denoted by the position Of expression.

Triples Representation

Operator	argument1	argument2	Position
*	c	d	(0)
+	b	(0)	(1)
+	(1)	(0)	(2)
=	(2)		

Code Optimization

Optimization is a program transformation technique, which tries to improve the code by making it consume less resources (ie., CPU, Memory) and deliver high speed.

There are two types of Code Optimization

- Machine Independent Optimization.
- Machine Dependent Optimization.

Cont....

- Machine Independent Code Optimization
  - Global Common sub – expression Elimination
  - Copy Propagation
  - Loop Invariant Code Motion
  - Function In lining
- Machine Dependent Code Optimization
  - Dead Code Elimination
  - Flow Control

Machine Independent Optimizations

- In this optimization, the compiler takes the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations.
  Ex:

```
do
{
item = 10;
value = value +
item;
}
while(value<100);
```

```
item = 10;
do
{
value = value + item;
}while(value<100);
```

GLOBAL EDGE

Peep hole optimization

Definition

It removes unnecessary code lines, and arranges the sequence statements in order to speed up the program execution without wasting resources (CPU, memory).

Redundant instruction elimination

```
int add_ten(int x)

{
     int y, z;
     y = 10;
     z = x + y;
     return z;
}
```

```
int add_ten(int x)
{
     int y;
     y = 10;
     y = x + y;
     return y;
 }
```

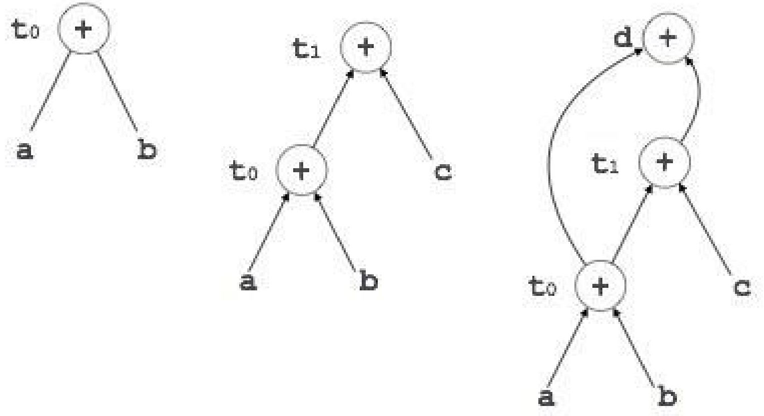
```
int add_ten(int x)
 {
      int y = 10;
      return x+y;
 }
```

```
int add_ten(int x)
 {
      return x+10;
 }
```

Code Generation

- Basic blocks comprises of sequence of three address instructions. Code generator takes these instructions as input.
- It should consider following things to generate the code:
    - Target Language
    - Selection of Instruction
    - Register Allocation
- Optimized intermediate code is converted to assembly language.

- t0 =a + b;
  t1 =t0 + c;
  t2 =t1 + t0;
  d = t2;



Machine dependent optimizations

Types

- Control Flow.
- Dead Code Elimination.

Flow of control optimization

```
MOV R1, R2
 GOTO L1
---------
---------
---------
---------
L1 : GOTO L2
L2 : INC R1
```

After:-

```
MOV R1, R2
GOTO L2
------------
------------
------------

 L2 : INC R1
```

Dead code elimination

- Instructions that compute a value never used is called a dead code which is eliminated.
- Ex:

```
LD R0, a
LD R1,b
LD R2, #6
add R0, R1
ST R0, c
```

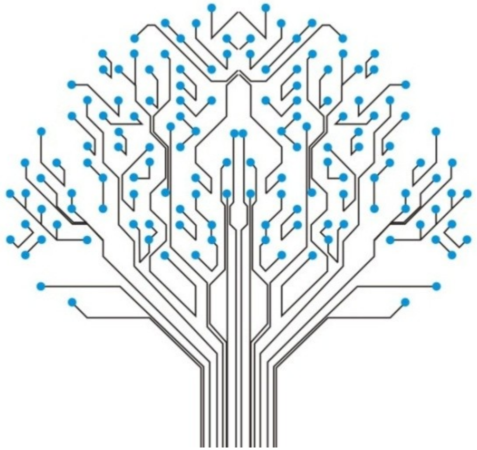
Large enough to Deliver, Small enough to Care

Global Village	South Main Street	Raheja Mindspace
IT SEZ	Milpitas	IT Park
Bangalore	California	Hyderabad

www.globaledgesoft.com

GLOBAL EDGE
Intelligence Of Things

Thank you



Fairness

Learning

Responsibility

Innovation

Respect