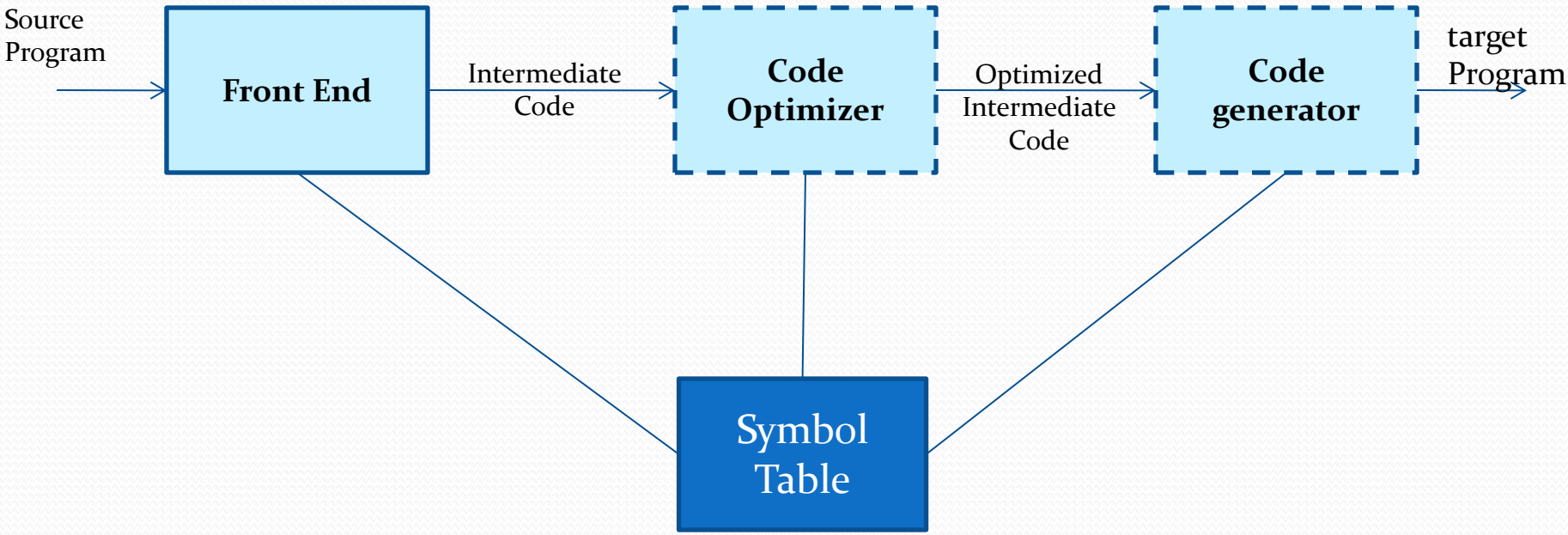
VII
Code Generation

Shyamalendu Kandar
Assistant Professor,
Department of Information Technology
IIEST, Shibpur

Introduction

- Final phase of compiler.
- The output code must be correct and of high quality(Make effective use of resources of the target machine.)



Factors Affecting Code Generation

A. Input
B. Target Code
C. Memory management
D. Instruction Selection
E. Register Allocation
F. Choices of Evaluation Order.

A.Input

- Intermediate Representation of the source program+ Information at Symbol table is used to determine the run time address of the data objects denoted by the names in the intermediate representation.
- Must be error free (especially semantic error), Type checked.

B. Target Code

- The target code may be in different forms a) absolute machine language b) relocatable machine language c) Assembly language

- absolute machine language program as output can be placed in a fixed location in memory and immediately executed.
- relocatable machine language program as output allows subprograms to be compiled seperately.
- Linker and loader plays its role when a set of relocatable object modules can be linked together and loaded for execution.
- Disadvantage is the cost but advantage is gaining the flexibility to compile subroutines separately and to call other previously compiled programs from an object module.
- Assembly language code as target makes the process easier.
- From symbolic instruction, macro facilities of assembler helps to generate code.
-

Memory Management

- Labels of TAC are converted to address of instruction

D. Instruction Selection

- Nature of instruction set of the target machine determines the difficulty of instruction selection.
- increment instruction (INC) can reduce code for a=a+1 to only one statement

MOV a, R_0

ADD #1, R_0

MOV R_0 , a

E. Register Allocation

- Instruction involving register operands are shorter and faster than those involving operands in memory.
- register allocation: selecting set of variables that will reside in registers at a point in the program.
- register assignment: picking the specific register that a variable will reside in.
- Assigning variables optimally to CPU registers is a difficult task.[NP complete problem]
- Certain machines requires register-pairs[Example IBM System 370 requires register pair for integer multiplication and division].

D x, y — The 64 bit dividend occupies an even/odd register pair whose even register is x; y represents the divisor.
After division even register holds the remainder and odd register the quotient.

M x, y — multiplicand x is the even register, multiplier 'y' is a single register.
Product occupies the entire even-odd pair.

t=a+b	L	R_0, a
t=t+c	A	R_0, b
t=t/d	A	R_0, c
	SRDA	R_0, 32
	D	R_0, d
	ST	R_1, t

SRDA R_0, 32 shifts the dividend into R_1, and clears R_0,

F. Choices of Evaluation Order

- The orders in which computations are performed can affect the efficiency of the target code.
- Some order require fewer registers, some may ensure higher degree of accuracy for floating point operations.
- Finding best evaluation order is also an NP complete problem.

Basic Blocks and Flow Graph

- Flow Graph: Graph representation of three address statements.
- Useful for understanding code generation algorithm.
- Nodes represents computations and edges represents flow of control.

Basic Block: Sequence of consecutive statements in which flow of control enters at the beginning and leaves at the ends without halt or possibility of branching expect at the end.

Algorithm for partition into basic block

Input: A sequence of three address statements

Output: A list of basic blocks with each three address statement in exactly one block.

Process:

1. Determine the set of leaders, the first statement of basic blocks.
    a) First statement is a leader.
    b) Any statement that is the target of a condition or unconditional goto is a leader.
    c) Any statement that immediately follows a goto or conditional goto statement is a leader.
2. For each leader, its basic block consists of the leader and all statements up to but not including the next leader or the end of the program.

Example

The code is to set a 10 X 10 matrix into identity matrix.

```
for i=1 to 10 do
    for j=1 to 10 do
        a[i,j] =0.0;
for i=1 to 10 do
        a[i,i] =1.0
```

TAC

```
 1)   i = 1
 2)   j = 1
 3)   t1 = 10 * i
 4)   t2 = t1 + j
 5)   t3 = 8 * t2
 6)   t4 = t3 - 88
 7)   a[t4] = 0.0
 8)   j = j + 1
 9)   if j <= 10 goto (3)
10)   i = i + 1
11)   if i <= 10 goto (2)
12)   i = 1
13)   t5 = i - 1
14)   t6 = 88 * t5
15)   a[t6] = 1.0
16)   i = i + 1
17)   if i <= 10 goto (13)
```

Example Cont…

Statement 1 i=1 is a leader.

Statement 2 j=1 is a leader because of goto in statement 11.

Statement 3 t_1=10*i is a leader because of goto in statement 9.

Statement 10 i=i+10 is a leader because it immediately follow the goto statement.

Statement 12 i=1 is a leader because it immediately follow the goto statement.

Statement 18 [does not exist] is a leader because it immediately follow the goto statement.

Example Cont…

Statement 17 to upwards statement 13 forms a block. [Not including the next leader]

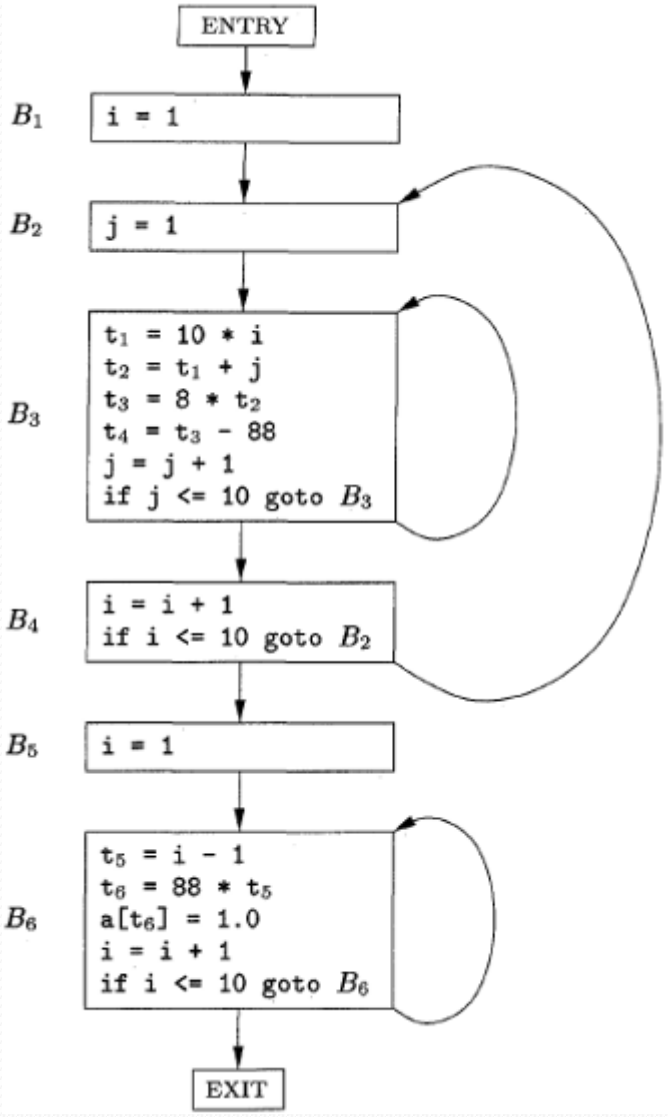
Statement 12 forms a block.

Statement 11 to 10 forms a block.

Statement 9 to 3 forms a block.

Statement 2 forms a block and statement 1 forms a block.

Total 6 blocks.



Assignment

- Construct basic block for the following TAC

1. location = -1
2. i=0
3. if i<100 goto 5
4. goto 13
5. t_1 =4*i
6. t_2=A[t_1]
7. if t_2 =x goto 9
8. goto 10
9. location =i
10. t_3= i+1
11. i=t_3
12. goto 3
13. .....

Transformations on Basic Blocks

- Two basic blocks are said to be equivalent if they compute the same set of expressions.
- Transformation can be applied on basic block for improving quality of code.
- Two important classes of local transformations that can be applied to basic blocks are

A. Structure preserving transformation

B. algebraic transformation

Transformation will not change the expressions computed by the block.

A. Structure preserving transformation

- The primary Structure-Preserving Transformations on basic blocks are

a) Common sub expression elimination
b) dead-code elimination
c) renaming temporary variables
d) interchange of two independent adjacent statements.

a. Common sub expression elimination

a=b+c

b=a-d

c=b+c

d=a-d

Here the second and fourth statement compute the same expression b+c-d. This can be transformed to

a=b+c

b=a-d

c=b+c //c computes b+c-d+c Thus not equivalent with first statement.

d=b

Another Example:

```
double x = d * (lim / max) * sx;
double y = d * (lim / max) * sy;
```

```
double depth = d * (lim / max);
double x = depth * sx;
double y = depth * sy;
```

b. dead-code elimination

- Dead Code is code that is either never executed or, if it is executed, its result is never used by the program.
- to remove code which does not affect the program results

```
1.    a = y + 2          1. a = y + 2
2.    z = x + w          2. x = a      //common sub-expression elimination
3.    x = y + 2          3. z = b + c
4.    z = b + c          4. b = a
5.    b = y + 2
```

benefits:

it shrinks program size.

it allows the running program to avoid executing irrelevant operations, which reduces its running time.

Further reading: Partially dead code

c. renaming temporary variables
d. Interchange of statements

- If a temporary variable is renamed (let from t to u) and all use of t is replaced by u then the value of the basic block is not changed.

Interchange of Statements:

Two adjacent independent statements can be interchanged without affecting the value of the block.

$t_1 = b+c$

$t_2 = x+y$

The statements for t_1 and t_2 can be interchanged.

B. Algebraic Transformations

- x= x+0 or x=x*1 can be eliminated
- x=y^2 [in some language y**2] can be transformed to x=y*y

DAG representation of Basic Blocks

- DAG gives a pictorial view of how the value computed by each statement in a basic block is used in subsequent statements of the block.
- identify the common computational parts that may be present at multiple points within the block.

The nodes of DAG correspond to the operations in the block.

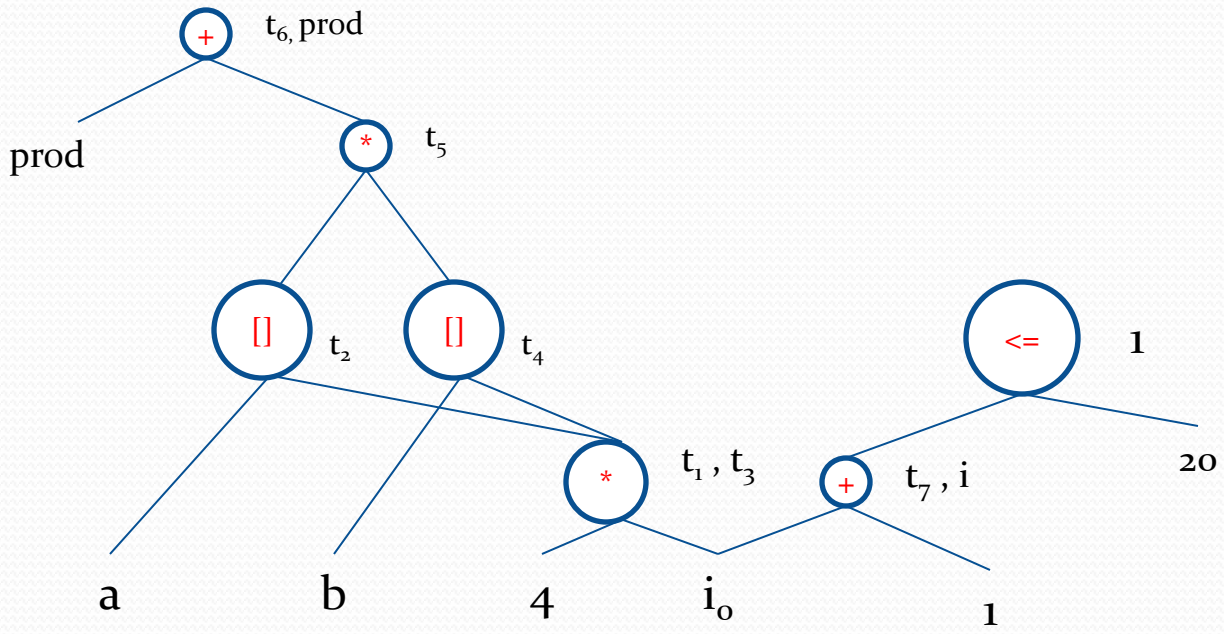
The labels are assigned as per the following rule.

a) Leaves are labeled by unique identifier[Either variable name or constants]

b) Interior nodes are labeled by operator symbol, according to the operation carried out.

Example

1. $t_1 = 4*i$
2. $t_2 = a[t_1]$
3. $t_3 = 4*i$
4. $t_4 = b[t_3]$
5. $t_5 = t_2 * t_4$
6. $t_6 = prod + t_5$
7. $prod = t_6$
8. $t_7 = i+1$
9. $i = t_7$
10. if i<20 goto 1.



Peephole Optimization

A simple but effective technique for locally improving the target code.

A method for trying to improve the performance of the target program by examining a short sequence of target instruction (Known as peephole)

References

- Guo,Yao "Introduction to Optimizations" Advanced Compiler Techniques (Fall 2011), School of EECS, Peking University.