Compiler Design

Preet Kanwal
Department of Computer Science & Engineering

Teaching Assistant : Kavya P K

Compiler Design

Unit 3: Intermediate Code Generation

Preet Kanwal

Department of Computer Science & Engineering

Lecture Overview

In this lecture, you will learn about -

- What is Three-Address Code?
- Format of TAC instructions
- Recap - Address Calculation for 1-D and 2-D arrays
- Example Questions

What is Three Address Code?

- Three-Address Code(TAC) is a Linearized representation of syntax tree or DAG.
- It has at most one operator on RHS of an instruction.
- Each instruction can have up to three addresses.
- The Address can either be a
    - Name (identifier)
    - Constant (number)
    - Temporary (holds an intermediate result)

Format of TAC instructions

The following table represents statements and their corresponding TAC format -

Statement	TAC Format
Assignment Statement	x = y op z (op : Binary operator) x = op y (op : Unary operator)
Copy statement	x = y
Unconditional jumps	goto L
Conditional Jumps	if x goto L ifFalse goto L
Compare and jump	if x relop y goto L ifFalse x relop y goto L

Format of TAC instructions

Statement	TAC Format
Address or Pointers	x = &y z = * x *x = a
Indexed Copy	x[i] = y y = x[i]
Procedure call : foo(a, b, …)	param a param b … call (foo, n) where, n is the number of arguments in function foo().
return statement	return y

Exercise 1

Generate Three-Address Code for the following statements -

1) a + b * c – d / b * c
2) x = *p + &y
3) x = f(y+1) + 2
4) x = foo (2 * x + 3, y + 10, g(i), h(3, j))
5) x = f(g(i), h(3, j))
6) alpha = (65 <=c && c<=90) || (97 <= c && c<=122)

Exercise 1.1 - Solution

Given Statements	Three Address Code
a + b * c – d / b * c	t1 = b * c t2 = a + t1 t3 = d / b t4 = t3 * c t5 = t2 - t4

Exercise 1.2 - Solution

Given Statements	Three Address Code
x = *p + &y	t1 = *p t2 = &y t3 = t1 + t2 x = t3

Exercise 1.3 - Solution

Given Statements	Three Address Code
x = f(y+1) + 2	t1 = y + 1 param t1 t2 = call f, 1 t3 = t2 + 2 x = t3

Exercise 1.4 - Solution

Given Statements	Three Address Code	
x = foo (2 * x + 3, y + 10, g(i), h(3, j))	t1 = 2 * x t2 = t1 + 3 param t2 t3 = y + 10 param t3 param i t4 = call g, 1 param t4 param 3 param j	t5 = call h, 2 param t5 t6 = call foo, 4 x = t6

Exercise 1.5 - Solution

Given Statements	Three Address Code
x = f(g(i), h(3, j))	param i t1 = call g, 1 param t1 param 3 param j t2 = call h,2 param t2 t3 = call f, 2 x = t3

Exercise 1.6 - Solution

Given Statements	Three Address Code
alpha = (65 <=c && c<=90) || (97 <= c && c<=122)	t1 = 65 <= c iffalse t1 goto L1 t2 = c <=90 iffalse t2 goto L1 L0 : alpha = true goto next L1 : t3 = 97<=c iffalse t3 goto L3 t4 = c <=122 iffalse t4 goto L3 goto L0 L3 : alpha = false next :

Exercise 2

Generate Three-Address Code for the following function -

```
void main() {
    int x, y;
    int m2 = x * x + y * y;
    while (m2 > 5)
    {
        m2 = m2 – x;
    }
}
```

Exercise 2 - Solution

Given Statements	Three Address Code	

void main() {
int x, y;
int m2 = x * x + y * y;
while (m2 > 5)
{
m2 = m2 – x;
}
}
	void main( )
	{
	int x;
	int y;
	int m2;
	t1 = x * x
	t2 = y * y
	t3 = t1 + t2
	m2 = t3
		L1: t4 = m2 > 5
		ifFalse t4 goto L2
		t5 = m2 - x
		m2 = t5
		goto L1
		L2:
		}

Exercise 3

Generate Three-Address Code for the following code snippet -

```
x = i + 10;
switch(x)
{
    case 1 : x = x * i;
    break;
    case 2 : x = 5;
    case 3  : x = i;
    default: x = 0;
}
```

Exercise 3 - Solution

Given Statements	Three Address Code	
	t1 = i + 10	L4 : if x ==3 goto L5
x = i + 10;	x = t1	goto L6
switch(x)	if x == 1 goto L1	L5 : x = i
{	goto L2	L6 : x = 0
case 1 : x = x * i;	L1 : t2 = x * i	
break;	x = t2	next :
case 2 : x = 5;	goto next	
case 3 : x = i;	L2 : if x ==2 goto L3	
default: x = 0;	goto L4	
}	L3 : x = 5	
	goto L5	

Recap - Address Calculation for 1-D Arrays

Array of an element of an array say A[i] is calculated using the following formula -

Address of A [i] = $A + W * (i - L_B)$

where,

A = Name of the array denotes the Base address

W = Storage Size of one element stored in the array (in bytes)

i = Subscript of element whose address is to be found

L_B = Lower limit of subscript, if not specified assume 0

Exercise 4

Generate Three-Address Code for the following code snippets -

1) a = b[i]

2)
```
do
    i = i + 1;
while(a[i] < v)
```

3)
```
Product = 0;
i = 1;
do
    Product = Product + A[i] * B[i];
    i = i + 1;
while( i < 20)
```

Exercise 4 - Solutions

Given Statements	Three Address Code
a = b[i]	t1 = 4 * i t2 = b + t1 or t2 = b[t1] a = t2

do
i = i + 1;
while(a[i] < v)
	L1: t1 = i + 1
	i = t1
	t2 = 4 * i
	t3 = a[t2]
	if t3 < v goto L1

Exercise 4 - Solutions

Given Statements	Three Address Code

Product = 0;
i = 1;
do
Product = Product + A[i] * B[i];
i = i + 1;
while(i < 20)
	Product =0
	i = 1
	L1 : t1 = 4 * i t2
	= A[t1]
	t3 = 4 * i t4
	= B[t3]
	t5 = t2 * t4
	t6 = product + t5
	product = t6
	t7 = i + 1 i
	= t7
	if i < 20 goto L1
	goto L2
	L2 :

Recap - Address Calculation for 2-D Arrays

- While storing the elements of 2-D array in memory, elements are allocated a contiguous memory locations.
- A 2-D array must be linearized so as to enable their storage.
- There are two ways to achieve linearization -
  - Row-major
  - Column-major

Recap - Address Calculation for 2-D Arrays - Row Major

The address of a location in Row Major System is calculated using the following formula:

$$\text{Address of } A[\,i\,][\,j\,] = A + W * [\,N * (\,i - L_r\,) + (\,j - L_c\,)\,]$$

where,

N = Number of columns of the given matrix

L_r = Lower limit of row/start row index of matrix, if not given assume 0

L_c = Lower limit of column/start column index of matrix, if not given assume 0

Recap - Address Calculation for 2-D Arrays - Column Major

The address of a location in Row Major System is calculated using the following formula:

$$\text{Address of } A[\,i\,][\,j\,] = A + W * [\,(i - L_r) + M * (j - L_c)\,]$$

where,

- **N** = Number of columns of the given matrix
- L_r = Lower limit of row/start row index of matrix, if not given assume 0
- L_c = Lower limit of column/start column index of matrix, if not given assume 0

TAC for 2-D Arrays -Assumptions

- **Assume all 2-D arrays follow row-major method.**
- **If the size of array is not mentioned assume it to be m x n array.**
- **Assume array type as integer and width of an array element as 4 bytes.**

Exercise 5

Generate Three-Address Code for the following code snippets -

1)
```
for(i = 0; i < n; i ++)
    for(j = 0; j <n ; j++)
        c[i][j] = 0;
```

where c is a 5x5 array

2)
```
for (i=1; i<=10 ; i++)
    for(j = 1; j <= 10; j++)
        C[i][ j]=     A[i][j] + B[i] [j];
```

where A and B are 10x10 arrays of type float, assume the arrays are 1-indexed.

Exercise 5.1 - Solution

1) for(i = 0; i < n; i
++) for(j = 0; j <n ;
j++)
c[i][j] = 0;

where c is a 5x5
array

Address calculation for c[i][j]

$$c[i][j] = B + W * [\, N * (i - L_r) + (j - L_c) \,]$$

$$= c + 4 * [\, n * (i - 0) + (j - 0) \,]$$

$$= c + 4 * (5 * i + j)$$

Exercise 5.1 - Solution

Given Statements	Three Address Code	
	i = 0	c[t5] = 0
for(i = 0; i < n; i ++)	L0: t1 = i < n	t6 = j + 1
for(j = 0; j <n ; j++)	if t1 goto L1	j = t6
c[i][j] = 0;	goto next	goto L4
	L1 : j=0	L3 : t7 = i + 1
	L4 : t2 = j < n	i = t7
	if t2 goto L2	goto L0
	goto L3	
	L2 : t3 = 5 * i	
	t4 = t3 + j	
	t5 = 4 * t4	

Exercise 5.1 - Solution

Given Statements	Three Address Code		

for (i=1; i<=10 ; i++)
for(j = 1; j <= 10; j++)
C[i][j]= A[i][j] + B[i] [j];
 	
where A, B, C are 10x10 arrays of type float
	i = 1
	L5 : t1 = i <=10
	if t1 goto L1
	goto next
	L1 : j = 1
	L 4 : t2 = j <=10
	if t2 goto L2
	goto L3 //inc i
	
	L2 : t1 = i - 1
	t2 = 10 * t1
	t3 = j - 1
	t4 = t2 + t3
	t5 = 8 * t4
	t6 = A[t5]
		t7 = i - 1
		t8 = 10 * t7
		t9 = j - 1
		t10 = t8 + t9
		t11 = 8 * t10
		t12 = B[t11]
		t13 = t6 + t12
		t14 = i - 1
		t15 = 10 * t14
		t16 = j - 1
		t17 = t15 + t16
		t18 = 8 * t17
		c[t18] = t13
			t19 = j + 1
			j = t19
			goto L4
			
			L3 : t20 = i + 1
			i = t20
			goto L5
			
			next :

THANK YOU

Preet Kanwal
Department of Computer Science & Engineering
preetkanwal@pes.edu