

2.6 - Formas Internas do Programa-Fonte

- ♦ O processo usual de compilação produz, a partir do código fonte, o programa objeto para ser executado:



- ♦ Entretanto, pode ser mais conveniente (por várias razões) dividir o processo de tradução em duas fases, gerando um "programa intermediário", antes da geração do programa objeto:



- ♦ Algumas razões:
 - intérprete
 - limitação de espaço
 - complicação da linguagem
 - complicação de projeto (código otimizado, por exemplo)
- ♦ O programa intermediário é escrito numa linguagem que, em termos de complexidade, é intermediária entre a linguagem de programação (alto nível de abstração) e o código de máquina. Os códigos intermediários usados mais frequentemente são: a notação polonesa e os códigos de 3 endereços.

Formas Internas do Programa-Fonte

- ♦ A característica comum a todos os códigos intermediários é a de que os operadores aparecem na ordem em que devem ser executados, ao contrário do que ocorre nas linguagens de alto-nível.

Notação polonesa (devida a Lucasiewicz)

- ♦ Na notação polonesa o operador é colocado após os operandos (notação "postfix").
- ♦ Se e_1 e e_2 são expressões "postfix" e θ é um operador binário, o resultado de aplicar θ às expressões e_1 e e_2 é indicado pela notação $e_1 e_2 \theta$.

- ♦ Exemplos:

Notação usual ("infix")	Notação polonesa
$(a + b) * c$	$a b + c *$
$a * (b + c)$	$a b c + *$
$(a + b) * (c + d)$	$a b + c d + *$

- ♦ Na notação polonesa, portanto, os parênteses não são necessários.
- ♦ A notação polonesa pode ser generalizada para operadores k-ários ($k \geq 1$): se θ é um operador k-ário aplicado às expressões (polonesas) e_1, \dots, e_k então o resultado é escrito como $e_1 e_2 \dots e_k \theta$.
- ♦ Exemplo: Seja a expressão condicional: IF e THEN x ELSE y, que tem o valor x se $e \neq 0$ e o valor y se $e = 0$. Seja # um operador ternário. Podemos representar essa expressão como: $e x y \#$ (será uma boa representação?).

Formas Internas do Programa-Fonte

Avaliação de expressões polonesas

- ♦ Usar uma pilha:
 - encontrou um operando: empilha;
 - encontrou um operador k-ário: aplicar esse operador aos primeiros k elementos a partir do topo da pilha, retirar esses elementos da pilha, e empilhar o resultado.

- ♦ Exemplo: $1\ 3\ +\ 5\ *$

passo	entrada	pilha
1	$1\ 3\ +\ 5\ *$	
2	$3\ +\ 5\ *$	[1]
3	$+ 5\ *$	[1][3]
4	$5\ *$	[4]
5	$*$	[4][5]
6		[20]

- ♦ Observe que para o operador # (if-then-else), o segundo e o terceiro argumentos serão sempre avaliados, embora apenas um deles vá ser usado. Portanto, se os operandos puderem ser indefinidos ou ter "efeitos colaterais" essa implementação para o if-then-else não será apenas ineficiente, pois poderá ser catastrófica.

Formas Internas do Programa-Fonte

- ♦ Vamos estender a notação polonesa para outros operadores:
d JUMP causa um desvio para d
 $e_1 e_2$ d JLT causa um desvio para d se $e_1 < e_2$
e d JEQZ causa um desvio para d se $e = 0$
- ♦ Vamos supor que o código em notação polonesa é armazenado em um vetor onde cada elemento contém um operador ou um operando. Então, com os operadores de desvio, a expressão condicional IF e THEN x ELSE y pode ser expressa por:

	13	14	15	16	17	18	19	20	
...	e	19	JEQZ	x	20	JUMP	y		...

- ♦ Os operadores de desvio (condicionais ou incondicionais) fazem com que seus operandos sejam retirados da pilha quando avaliados sem que nenhum valor seja empilhado.
- ♦ Outros operadores:
 - a) atribuição: $v e :=$ Nesse caso, como resultado da operação não se tem um valor para empilhar.
 - b) declaração de variáveis indexadas: array $A[i_1:s_1, \dots, i_n:s_n]$ pode ser representada por: $i_1 s_1 \dots i_n s_n A ADEC$, onde ADEC é um operador com número variável de operandos (essa informação vai estar disponível no registro correspondente a A na tabela de símbolos).

Formas Internas do Programa-Fonte

- c) referência a variáveis subscritas: $A[e_1, \dots, e_n]$ pode ser representada por:
 $e_1 \dots e_n$ A SUBS.
- d) início e fim de blocos: BLOCK e BLOCKEND (operadores de zero argumentos).

- ♦ Exemplo: Seja o programa:

```
begin
  integer k;
  array A[1:i-j];
  k := 0;
L:  if (i > j) then k := k + A[i-j]*6
    else begin
          i := i + 1;
          i := i + 1;
          goto L;
        end;
end;
```

Nesse caso, o código intermediário em notação polonesa será:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
BLOCK	1	i	j	-	A	ADEC	k	0	:=	i	j	-	29	JLEQ	k	k	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
i	j	-	A	SUBS	6	*	+	:=	41	JUMP	i	i	1	+	:=	i	i
36	37	38	39	40	41	42											
1	+	:=	11	JUMP	BLOCKEND	...											

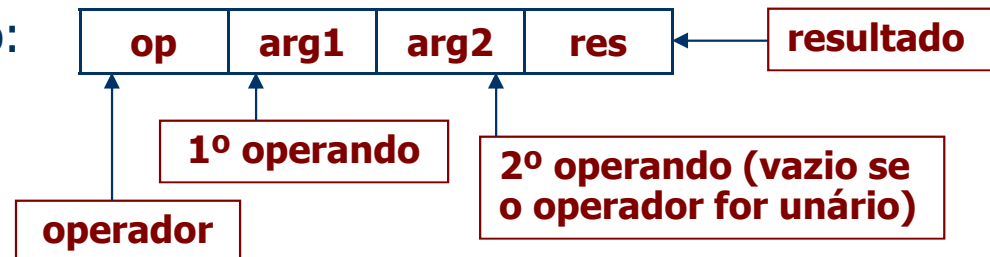
Formas Internas do Programa-Fonte

- Apenas para ilustrar a diferença entre a linguagem intermediária e o código objeto, consideremos a entrada de bloco. No caso de código objeto, em vez de BLOCK teremos todas as instruções necessárias para a abertura de um bloco.

Código de 3 endereços

- Código de 3 endereços é uma sequência de comandos (tipicamente) da forma $A := B \text{ op } C$ onde A , B e C são identificadores, constantes ou nomes temporários criados pelo próprio compilador e op é um único operador.
- Exemplo: $X + Y * Z$ será escrito como:
 $T1 := Y * Z$
 $T2 := X + T1$
onde $T1$ e $T2$ são nomes criados pelo compilador.
- Um outro comando de 3 endereços típico é: $\text{IF } x \text{ relop } y \text{ GOTO } z$, onde relop é um operador relacional ($<$, $=$, $<=$, $>$, ...).
- A implementação do código de 3 endereços nos compiladores é feita usando, principalmente, quádruplas ou triplas.

Quádrupla: usa uma estrutura como:



Formas Internas do Programa-Fonte

- ♦ Exemplos:

a) $-B * (C + D)$ será representado por:

(-	,	B	,		,	T1)
(+	,	C	,	D	,	T2)
(*	,	T1	,	T2	,	T3)

b) $IF\ A < B\ GOTO\ L$ será representado por: (BL, L, A, B)

- ♦ Outras quádruplas importantes são:

(BR, N, ,)	desvio para a quádrupla número N
(:=, A, , B)	atribuição ($B := A$)
(CVRI, A, , B)	converte de real para inteiro ($B := \text{int}(A)$)
(BLOCK, , ,)	entrada de bloco
(BLOCKEND, , ,)	saída de bloco
(BOUNDS, L, U,)	limites (inferior e superior) de um array (com relação a uma dimensão)
(ADEC, A, ,)	A é um array que está sendo declarado. Se o array tem n dimensões, essa quádrupla é precedida por n quádruplas BOUNDS.

Formas Internas do Programa-Fonte

- No caso de referências a variáveis indexadas pode-se ter operadores com mais de dois operandos (por exemplo, $C := A[i, B[j]]$). A solução é considerar que, na forma intermediária, os arrays são uni-dimensionais.

- Exemplo: Seja $A[1:m, 1:n]$

$$\text{endereço}(A[i, B[j]]) = \underbrace{\text{endereço}(A[1, 1] - n - 1)}_{\text{constante}} + \underbrace{(i * n + B[j])}_{\text{deve ser calculada}}$$

Logo, $C := A[i, B[j]]$ será representado por:

$(*, i, n, T1)$
 $(+, T1, B[j], T2)$
 $(:=, A[T2], , C)$

operandos podem
estar subscritos

Formas Internas do Programa-Fonte

- Para o programa:

```
begin
  integer k;
  array A[1:i-j];
  k := 0;
L:  if (i > j) then k := k + A[i-j]*6
    else begin
      i := i + 1;
      i := i + 1;
      goto L;
    end;
end;
```

teremos:

[1] (BLOCK, , ,)	[11] (BR, 17, ,)
[2] (-, i, j, T1)	[12] (+, i, 1, T5)
[3] (BOUNDS, 1, T1,)	[13] (:=, T5, , i)
[4] (ADEC, A, ,)	[14] (+, i, 1, T6)
[5] (:=, 0, , k)	[15] (:=, T6, , i)
[6] (JLEQ, 12, i, j)	[16] (BR, 6, ,)
[7] (-, i, j, T2)	[17] (BLOCKEND, , ,)
[8] (*, A[T2], 6, T3)	
[9] (+, k, T3, T4)	
[10] (:=, T4, , k)	

Formas Internas do Programa-Fonte

Tripla: usa uma estrutura como:

op	arg1	arg2
----	------	------

onde **arg1** e **arg2** são ponteiros para a tabela de símbolos (como no caso das quádruplas) ou ponteiros para outras triplas.

- Exemplo: $A := -B * (C + D)$

[1] (-, B,)

[2] (+, C, D)

[3] (*, [1], [2])

[4] (:=, A, [3])

representa um ponteiro
para a 2ª tripla



Quádruplas ou Triplas ?

- Quando se usa quádruplas, as posições para os nomes temporários podem ser acessadas diretamente na tabela de símbolos. Para as triplas, a atribuição de memória aos nomes temporários é deixada para a fase seguinte (interpretação ou geração de código).
- Uma vantagem das quádruplas: no caso de otimização, o compilador poderá mudar comandos de lugar. Se um comando que calcula o valor de A é mudado de lugar, isso não terá efeito sobre os comandos que usam o valor de A. No caso das triplas, os comandos que usam A deverão ser refeitos, isto é, os ponteiros deverão ser atualizados.
- Desvantagem das quádruplas: enche a tabela de símbolos com nomes temporários.

Formas Internas do Programa-Fonte

- ♦ A deficiência das triplas apontada em (b) pode ser sanada usando-se triplas indiretas: os comandos em vez de serem triplas são apenas ponteiros para o conjunto de triplas. Se uma alteração na ordem for necessária, basta alterar o vetor COMANDO.
- ♦ Exemplo: $A := -B + (C + D)$ é agora representado por:

	COMANDO
1	23
2	24
3	25
4	26

	TRIPLAS INDIRETAS
23	(-, B,)
24	(+, C, D)
25	(*, [23], [24])
26	(:=, A, [25])

Se quisermos alterar a ordem entre as triplas 23 e 24:

	COMANDO
1	24
2	23
3	25
4	26



só muda aqui

	TRIPLAS INDIRETAS
23	(-, B,)
24	(+, C, D)
25	(*, [23], [24])
26	(:=, A, [25])



aqui, nada muda