Júlia Wotzasek Pereira

Documentação do Código Intermediário do Compilador *Panoramix Compiler*

São José dos Campos - Brasil

Maio de 2020

Sumário

1 Quádruplas para Declaração e Chamada de Funções

1.1 Definição de Funções

1.1.1 Aplicação

Definição de Função. Indica que as próximas quádruplas serão referentes ao corpo da função. A função acaba na declaração da quádrupla **END** (ver seção 1.5).

1.1.2 Formato

(FUN, type, function_name, -)

1.1.3 Parâmetros

- **type:** refere-se ao tipo de retorno da função. Se a função tem retorno inteiro, deve ser colocado **INT** e se não tem retorno deve ser colocado **VOID**.
- **function_name:** É o nome dado a função.

1.1.4 Exemplo

Considere o seguinte código em C-:

```
1  /* Codigo em C- */
2  int print_Hello_World()
3  {
4      /* corpo da funcao */
5  }
```

Obtemos as seguintes quádruplas:

(FUN, INT, print_Hello_World, -)
(END, print_Hello_World, -, -)

1.2 Argumento de Funções

1.2.1 Aplicação

Argumento de função. Deve aparecer sempre que uma função é declarada indicando os seus argumentos (ver seção 1.1). Após serem declarados, os argumentos devem ser carregados (ver seção 2.2).

1.2.2 Formato

(ARG, INT, var, scope)

1.2.3 Parâmetros

- **var:** É o nome da variável de argumento.
- **scope:** É o nome da função na qual a variável está sendo declarada como argumento.

1.2.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    int sort(int a[], int low, int high)
3    {
4        /* corpo do codigo */
5    }
```

Obtemos as seguintes quádruplas:

(FUN, INT, sort, -)
(ARG, INT, a, sort)
(ARG, INT, low, sort)
(ARG, INT, high, sort)
(LOAD, $t1, a, -)
(LOAD, $t2, low, -)
(LOAD, $t3, high, -)
(END, sort, -, -)

1.3 Chamada de Funções

1.3.1 Aplicação

Chamada de função. Indica qual função deve ser chamada, em qual variável ela deve ser alocada e quantos parâmetros esta possui. Os parâmetros são passados como **PARAM** (ver seção 1.4).

1.3.2 Formato

(CALL, temp, function_name, n_params)

1.3.3 Parâmetros

- **temp:** é o registrador temporário que recebe o retorno da função.
- **function_name:** é o nome dado a função chamada.
- **n_params:** é o número de parâmetros que devem ser passados para a função.

1.3.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = sort(a, b)
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, a, -)
(PARAM, $t2, -, -)
(LOAD, $t3, b, -)
(PARAM, $t3, -, -)
(CALL, $t4, sort, 3)
(ASSIGN, $t1, $t4, -)
(STORE, i, $t1, -, -)

1.4 Parâmetros de Função

1.4.1 Aplicação

Parâmetro de função. Cada vez que uma variável é declarada como parâmetro, ela é empilhada para ser usada quando a função por chamada. Além disso, conforme os parâmetros são adicionados, um contador é incrementado, para garantir que a chamada de função saiba quantos parâmetros ela possui (ver seção 1.3).

1.4.2 Formato

(PARAM, temp, -, -)

1.4.3 Parâmetros

- **temp:** é o registrador temporário que deve ser empilhado como parâmetro de função.

1.4.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = sort(a, b)
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, a, -)
(PARAM, $t2, -, -)
(LOAD, $t3, b, -)
(PARAM, $t3, -, -)
(CALL, $t4, sort, 3)
(ASSIGN, $t1, $t4, -)
(STORE, i, $t1, -, -)

1.5 Finalização de Funções

1.5.1 Aplicação

Finalização de função. Indica que o corpo da função está encerrado. Deve ser precedido por um indicador de início de função (ver seção 1.1).

1.5.2 Formato

(END, function_name, -, -)

1.5.3 Parâmetros

- **function_name:** É o nome dado a função.

1.5.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    int print_Hello_World()
3    {
4        /* corpo da funcao */
5    }
```

Obtemos as seguintes quádruplas:

(FUN, INT, print_Hello_World, -)
(END, print_Hello_World, -, -)

2 Quádruplas para Leitura/Escrita e Declaração de Variáveis

2.1 Declaração de Variáveis

2.1.1 Aplicação

Alocação de variáveis. Destina a aquele rótulo de variável o espaço de memória necessário. Serve para declaração de variáveis simples ou vetores. Se for vetor, precisa passar o parâmetro de tamanho na declaração.

2.1.2 Formato

(ALLOC, var_name, scope, [**size**])

2.1.3 Parâmetros

- **var_name:** refere-se ao rótulo dado à variável.
- **scope:** é o escopo de declaração da variável. Se esta for declarada fora de qualquer função, seu escopo deve ser indicado como "GLOBAL".
- **size:** é o tamanho, caso seja vetor. Se for variável simples, não é passado. **(opcional)**

2.1.4 Exemplo

Considere o seguinte código em C-:

```
1  /* Codigo em C- */
2  int vet[10];
3  int sort()
4  {
5      int w;
6      /* corpo da funcao */
7  }
```

Obtemos as seguintes quádruplas:

(ALLOC,vet,GLOBAL,10)
(FUN, INT, sort, -)
(ALLOC,w,sort,-)
(END, sort, -, -)

2.2 Leitura de variáveis da Memória

2.2.1 Aplicação

Leitura de variável da memória. Deve ser utilizada sempre que a variável é utilizada no código e quando a variável é parâmetro da função. Para variáveis indexadas, pode-se carregar ou o vetor ou apenas um campo.

2.2.2 Formato

(LOAD, temp, var_name, [index])

2.2.3 Parâmetros

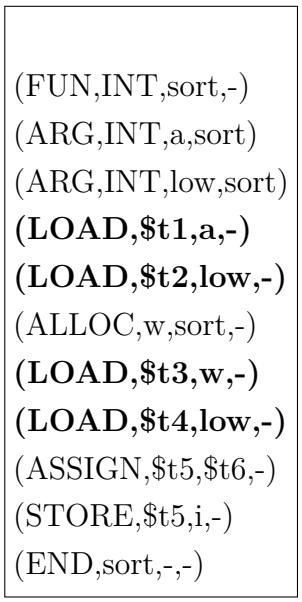
- **temp:** É o registrador temporário que recebe a leitura feita da memória.
- **var_name:** É o nome da variável a ser carregada.
- **index:** É a posição a ser carregada da variável indexada em **var_name**. **(opcional)**

2.2.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    int sort(int a[], int low)
3    {
4        int w;
5        w = low;
6        /* corpo da funcao */
7    }
```

Obtemos as seguintes quádruplas:


(FUN,INT,sort,-)
(ARG,INT,a,sort)
(ARG,INT,low,sort)
(LOAD,$t1,a,-)
(LOAD,$t2,low,-)
(ALLOC,w,sort,-)
(LOAD,$t3,w,-)
(LOAD,$t4,low,-)
(ASSIGN,$t5,$t6,-)
(STORE,$t5,i,-)
(END,sort,-,-)

2.3 Escrita de variáveis na Memória

2.3.1 Aplicação

Escrita de variável na memória. Deve ser utilizada sempre que a variável recebe um novo valor (ver seção 2.4). Para que haja escrita na memória, é preciso ter havido o carregamento da variável em algum registrador temporário (ver seção 2.2).

2.3.2 Formato

(STORE, temp, var_name, [index])

2.3.3 Parâmetros

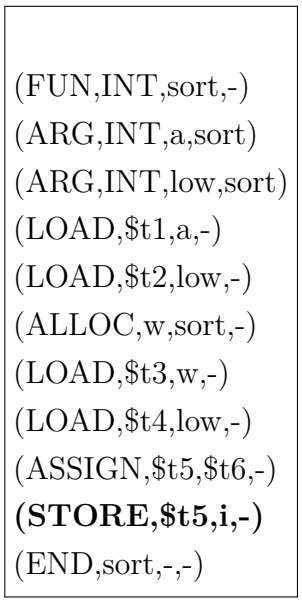
- **temp:** É o registrador temporário que possui o conteúdo a ser escrito na memória.
- **var_name:** É o nome da variável que receberá o valor do registrador temporário.
- **index:** É a posição a ser escrita da variável indexada em **var_name**. **(opcional)**

2.3.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    int sort(int a[], int low)
3    {
4        int w;
5        w = low;
6        /* corpo da funcao */
7    }
```

Obtemos as seguintes quádruplas:


(FUN,INT,sort,-)
(ARG,INT,a,sort)
(ARG,INT,low,sort)
(LOAD,$t1,a,-)
(LOAD,$t2,low,-)
(ALLOC,w,sort,-)
(LOAD,$t3,w,-)
(LOAD,$t4,low,-)
(ASSIGN,$t5,$t6,-)
(STORE,$t5,i,-)
(END,sort,-,-)

2.4 Atribuição

2.4.1 Aplicação

Atribuição. Faz o valor do segundo parâmetro ser atribuído ao primeiro. Sempre é acompanhado de **STORE** (ver seção 2.3).

2.4.2 Formato

(ASSIGN, temp_D, temp_S, -)

2.4.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny) associado a uma variável.
- **temp_S:** é o primeiro registrador temporário de base (S - Source). É o registrador de origem do valor.

2.4.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(ASSIGN, $t1, $t2, -)
(STORE, $t1, i, -)

3 Quádruplas de Controle

3.1 Finalização de Programas

3.1.1 Aplicação

Finalização de programa. Deve aparecer em todo código gerado para encerrar a execução de instruções.

3.1.2 Formato

(HALT, -, -, -)

3.1.3 Parâmetros

Não Possui parâmetros.

3.2 Rótulos de Controle - *Labels*

3.2.1 Aplicação

Labels de controle servem para que saltos possam ser feitos na execução do código. Assim, eles aparecem em para execução de laços *while* e para *if*. Para o salto para uma *label*, utiliza-se a função **GOTO** (ver seção 3.3).

3.2.2 Formato

(LABEL, label_name, -, -)

3.2.3 Parâmetros

- **label_name:** é o nome do *label* de referência para aquele pedaço de código.

3.2.4 Exemplo

Considere o seguinte código em C-:

```
1  /* Codigo em C- */
2  if(i >= 0)
3  {
4      output(i)
5  }
6  else
7  {
8      output(i)
9  }
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(GEQ, $t2, $t1, 0)
(IFF, $t2, L1, 0)
(LOAD, $t3, i, -)
(PARAM, $t3, -, -)
(CALL, $t4, output, 1)
(GOTO, L2, -, -)
(LABEL, L1, -, -)
(LOAD, $t5, i, -)
(PARAM, $t5, -, -)
(CALL, $t6, output, 1)
(LABEL, L2, -, -)

3.3 Desvio de Fluxo

3.3.1 Aplicação

Para desviar o fluxo de execução, utiliza-se a quádrupla **GOTO**. Para que ocorra os desvios, precisa haver os rótulos de partes do código (ver seção 3.2).

3.3.2 Formato

(GOTO, label_name, -, -)

3.3.3 Parâmetros

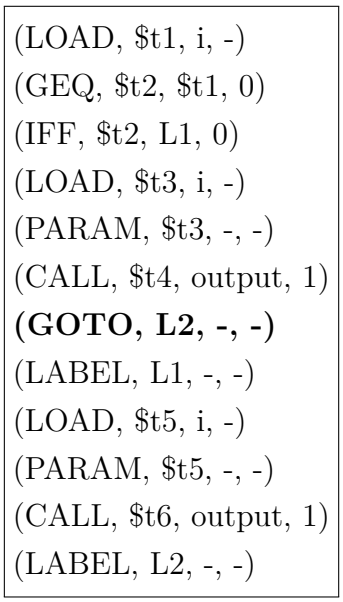
- **label_name:** é o nome do *label* de referência para o pedaço de código que deve passar a ser executado.

3.3.4 Exemplo

Considere o seguinte código em C-:

```
1  /* Codigo em C- */
2  if(i >= 0)
3  {
4      output(i)
5  }
6  else
7  {
8      output(i)
9  }
```

Obtemos as seguintes quádruplas:


(LOAD, $t1, i, -)
(GEQ, $t2, $t1, 0)
(IFF, $t2, L1, 0)
(LOAD, $t3, i, -)
(PARAM, $t3, -, -)
(CALL, $t4, output, 1)
(GOTO, L2, -, -)
(LABEL, L1, -, -)
(LOAD, $t5, i, -)
(PARAM, $t5, -, -)
(CALL, $t6, output, 1)
(LABEL, L2, -, -)

3.4 Desvio Condicional - *If False*

3.4.1 Aplicação

Desvio Condicional. Sempre vem precedido de uma condição. Se a condição for falsa, desvia para o rótulo passado; caso a condição seja verdadeira, continua o fluxo corrente de instruções. (ver seções 3.2 e 3.3)

3.4.2 Formato

(IFF, temp_test, label_branch, -)

3.4.3 Parâmetros

- **temp_test:** É o nome do registrador temporário que possui o teste que foi feito para averiguar se a condição é verdadeira ou falsa.

- **label_branch:** É o nome do rótulo para o qual deve haver desvio caso a condição seja falsa.

3.4.4 Exemplo

Considere o seguinte código em C-:

```
1  /* Codigo em C- */
2  if(i >= 0)
3  {
4      output(i)
5  }
6  else
7  {
8      output(i)
9  }
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(GEQ, $t2, $t1, 0)
(IFF, $t2, L1, 0)
(LOAD, $t3, i, -)
(PARAM, $t3, -, -)
(CALL, $t4, output, 1)
(GOTO, L2, -, -)
(LABEL, L1, -, -)
(LOAD, $t5, i, -)
(PARAM, $t5, -, -)
(CALL, $t6, output, 1)
(LABEL, L2, -, -)

4 Quádruplas Lógico-Aritméticas

4.1 Adição

4.1.1 Aplicação

Soma. Soma os valores contidos nos dois registradores passados de parâmetros e atribui o valor calculado a um terceiro registrador.

4.1.2 Formato

(ADD, temp_D, temp_S1, temp_S2)

4.1.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor da soma.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É a primeira parcela da soma.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É a segunda parcela da soma.

4.1.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m + 1;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(ADD, $t3, m, 1)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.2 Subtração

4.2.1 Aplicação

Subtração. Subtrai os valores contidos nos dois registradores passados de parâmetros e atribui o valor calculado a um terceiro registrador.

4.2.2 Formato

(SUB, temp_D, temp_S1, temp_S2)

4.2.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor da subtração.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É a primeira parcela da subtração.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É a segunda parcela da subtração.

4.2.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m - 1;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(SUB, $t3, m, 1)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.3 Multiplicação

4.3.1 Aplicação

Multiplicação. Multiplica os valores contidos nos dois registradores passados de parâmetros e atribui o valor calculado a um terceiro registrador.

4.3.2 Formato

(MULT, temp_D, temp_S1, temp_S2)

4.3.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor da multiplicação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da multiplicação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da multiplicação.

4.3.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m * 2;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(MULT, $t3, m, 2)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.4 Divisão

4.4.1 Aplicação

Divisão. divide os valores contidos nos dois registradores passados de parâmetros e atribui o valor calculado a um terceiro registrador.

4.4.2 Formato

(DIV, temp_D, temp_S1, temp_S2)

4.4.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor da divisão.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o dividendo.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o divisor.

4.4.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m / 2;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(DIV, $t3, m, 2)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.5 Maior que

4.5.1 Aplicação

Maior que. É a comparação se o primeiro valor passado é maior do que o segundo. Retorna 1 se verdadeiro e zero se falso.

4.5.2 Formato

(GT, temp_D, temp_S1, temp_S2)

4.5.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.5.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m > 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(GT, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.6 Menor que

4.6.1 Aplicação

Menor que. É a comparação se o primeiro valor passado é menor do que o segundo. Retorna 1 se verdadeiro e zero se falso.

4.6.2 Formato

(LT, temp_D, temp_S1, temp_S2)

4.6.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.6.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m < 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(LT, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.7 Maior ou igual a

4.7.1 Aplicação

Maior ou igual a. É a comparação se o primeiro valor passado é maior ou igual ao segundo. Retorna 1 se verdadeiro e zero se falso.

4.7.2 Formato

(GEQ, temp_D, temp_S1, temp_S2)

4.7.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.7.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m >= 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(GEQ, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.8 Menor ou igual a

4.8.1 Aplicação

Menor ou igual a. É a comparação se o primeiro valor passado é menor ou igual ao segundo. Retorna 1 se verdadeiro e zero se falso.

4.8.2 Formato

(LEQ, temp_D, temp_S1, temp_S2)

4.8.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.8.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m <= 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(LEQ, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.9 É igual a

4.9.1 Aplicação

É igual a. É a comparação se o primeiro valor passado é igual ao segundo. Retorna 1 se verdadeiro e zero se falso.

4.9.2 Formato

(EQ, temp_D, temp_S1, temp_S2)

4.9.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.9.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m == 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(EQ, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)

4.10 É diferente de

4.10.1 Aplicação

É diferente de. É a comparação se o primeiro valor passado é diferente do segundo. Retorna 1 se verdadeiro e zero se falso.

4.10.2 Formato

(NEQ, temp_D, temp_S1, temp_S2)

4.10.3 Parâmetros

- **temp_D:** é o registrador temporário de destino (D - Destiny). Recebe o valor booleano da comparação.
- **temp_S1:** é o primeiro registrador temporário de base (S - Source). É o primeiro fator da comparação.
- **temp_S2:** é o segundo registrador temporário de base (S- Source). É o segundo fator da comparação.

4.10.4 Exemplo

Considere o seguinte código em C-:

```
1    /* Codigo em C- */
2    i = m != 0;
```

Obtemos as seguintes quádruplas:

(LOAD, $t1, i, -)
(LOAD, $t2, m, -)
(NEQ, $t3, m, 0)
(ASSIGN, $t1, $t3, -)
(STORE, $t1, i, -)