Daiana Kathrin Santana Santos
120.357

Implementação do Compilador C-

São José dos Campos - Brasil

Julho de 2021

Daiana Kathrin Santana Santos
120.357

Implementação do Compilador C-

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Compiladores.

Docente: Profa. Dra. Thaína Aparecida Azevedo Tosta

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Julho de 2021

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Compiladores são programas de computador que traduzem de uma linguagem para outra, segundo Louden (1). O compilador deve ser eficiente, para que a tradução não seja exageradamente lenta, portanto o código produzido deve ser eficiente para que seja rápido quando executado (2).

Compiladores são importantes, pois sem eles não haveriam comunicação entre humanos e computadores, já que os seres humanos falam linguagens completamente diferentes da linguagem em que o computador se comunica. Para tanto, a construção de compiladores reúne diversas disciplinas da ciência da computação em uma só aplicação (2), então é de extrema importância para os estudantes da área de engenharia de computação estudar sobre compiladores.

Este projeto é a junção e a continuação de dois projetos das disciplinas de "Laboratório de Sistemas computacionais: Arquitetura e Organização de Computadores"e "Compiladores". Na primeira disciplina foi feito a implementação de um processador baseado na arquitetura MIPS, este é como se fosse o cérebro de um computador, sem ele o computador não consegue executar nenhuma instrução. O processador busca a instrução na memória/registrador, interpreta a ação requerida, obtém, processa e grava os dados de volta (3). Na segunda disciplina foi feita a fase de análise do compilador, em que faz a análise léxica, sintática e semântica do código-fonte, verificando se há algum erro.

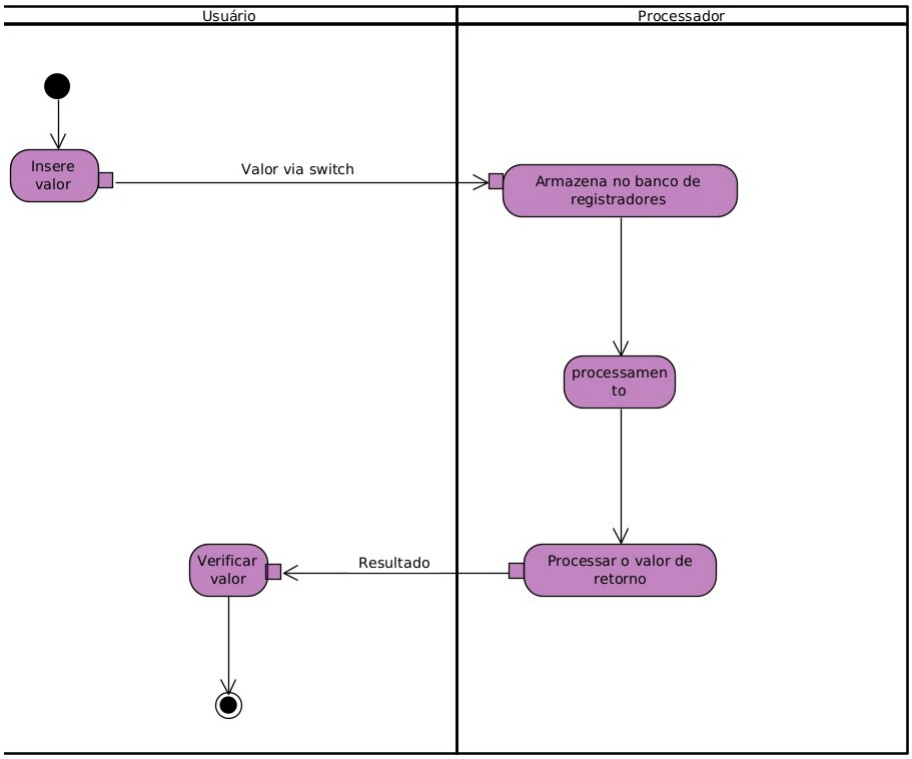
Então nesta disciplina foi feita a junção do processador com o compilador em sua fase de análise e acrescentado a fase de síntese, onde literalmente faz a tradução do código-fonte para o código alvo.

Para explicação do projeto, no Capítulo 2 é explicado o processador desenvolvido, os componentes nele, o conjunto de instruções e a organização de sua memória. No Capítulo 3 é apresentado o compilador, no seu módulo de análise, contemplando a análise léxica, sintática e semântica com o módulo de síntese, desde o código intermediário até o código binário, em seguida, no Capítulo 4 são apresentados exemplos de códigos compilados e por fim no Capítulo 5 as considerações finais.

2 O processador

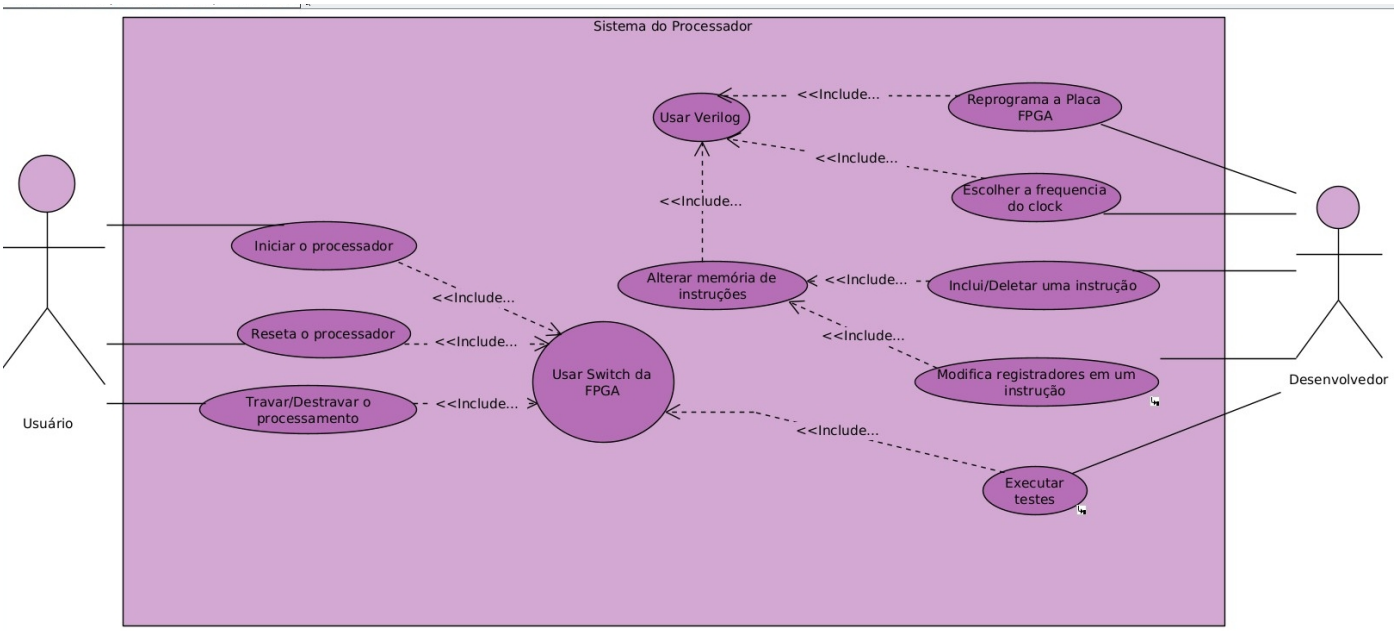
O processador é baseado na arquitetura MIPS monociclo, ou seja, cada instrução é executada em apenas um ciclo de *clock* e possui uma quantidade reduzida de instruções utilizando os conceitos da arquitetura RISC. O projeto do processador foi implementado e testado em uma placa Cyclone IV E FPGA, contendo 18 switches de entrada e 8 displays de 7 segmentos. Nas figuras 1, 2 e 3 faz a modelagem de um processador, em que é um sistema complexo e difícil, portanto não foi possível mostrar todas suas atividades e utilidades, mas sim um resumo delas.

Figura 1 – Diagrama 1 do processador



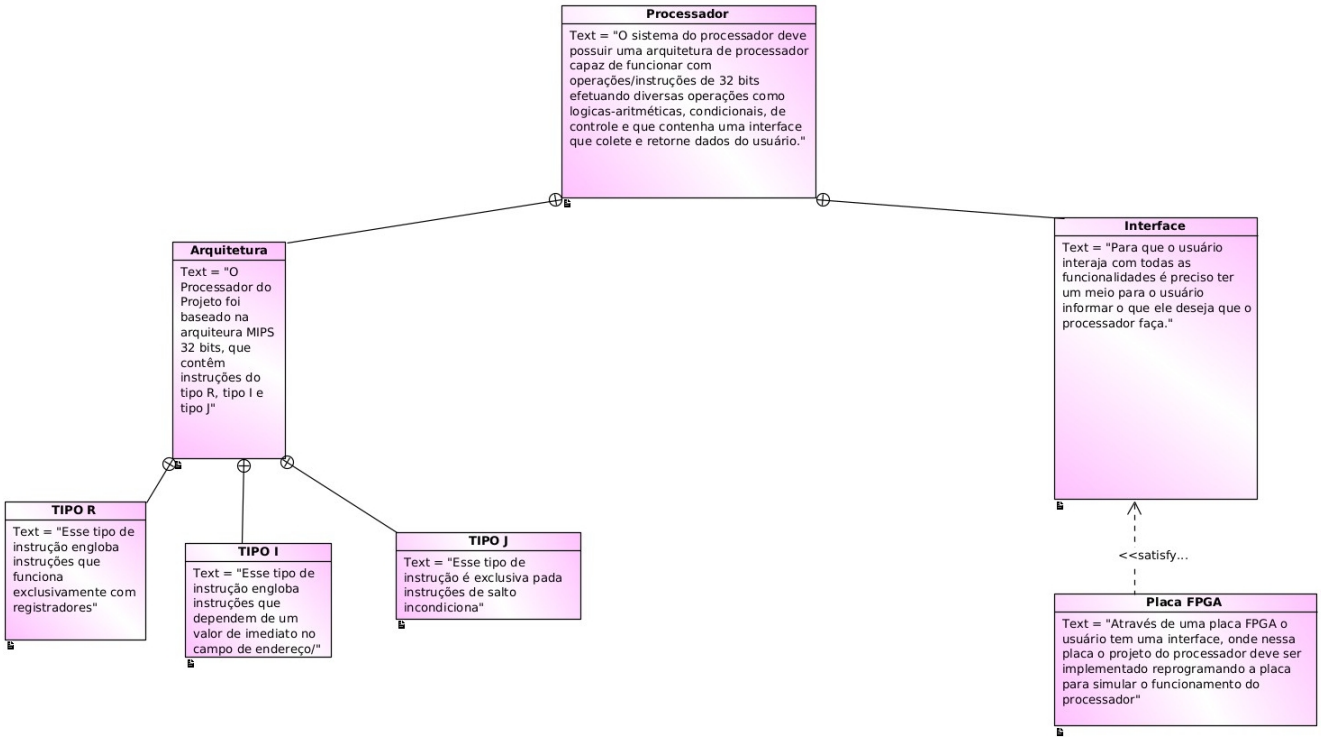
Fonte: Autoria Própria

Figura 2 – Diagrama 2 do processador



Fonte: Autoria Própria

Figura 3 – Diagrama 3 do processador

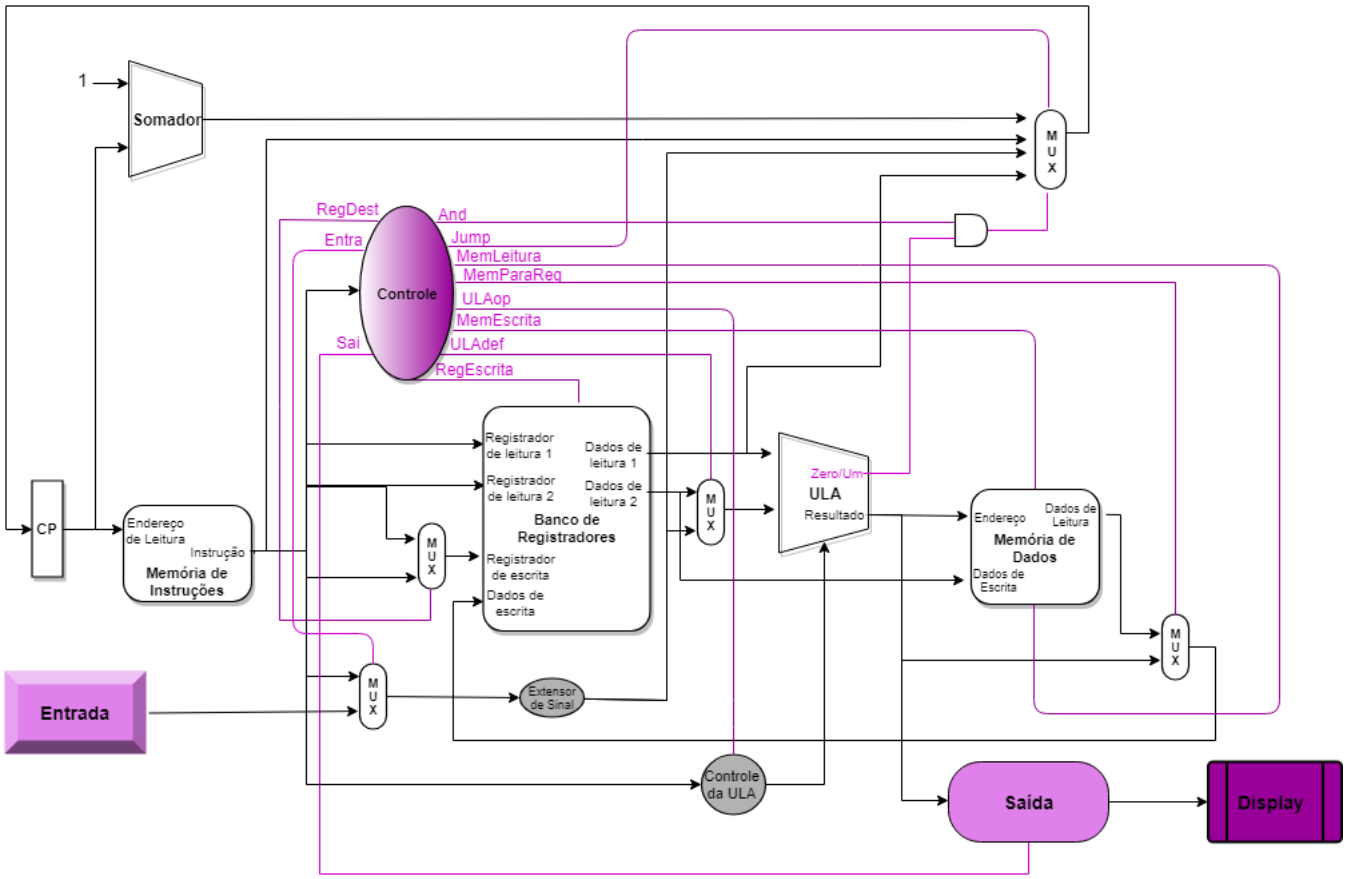


Fonte: Autoria Própria

2.1 Componentes

O caminho de Dados do processador, mostrado na figura Figura 4 foi construído baseado no caminho de dados da arquitetura MIPS.

Figura 4 – Caminho de Dados



Fonte: Autoria Própria

Os componentes utilizados no caminho de dados Figura 4 são explicados abaixo.

2.1.1 Unidade de Controle

A unidade de controle envia sinais e controla quase todos os blocos. Utiliza sinais diferentes em cada instrução para conseguir fazer a comunicação e identificar qual instrução está sendo realizada com os outros componentes.

2.1.2 Multiplexadores

Um multiplexador é um circuito combinacional que contém n entradas e apenas uma saída. O valor do sinal de saída será igual a um dos sinais de entrada, conforme o valor de sinais de entrada denominados seletores (4).

2.1.3 Contador de Programa

É um registrador, em que é usado para conter o endereço da instrução atual (5). A implementação do bloco do Contador de Programa (PC), tem o objetivo de calcular a próxima instrução *(PCOut)*. As instruções são executadas suscetivamente, por isso o endereço da próxima instrução é a anterior mais um *(PCOut = PCIn + 1)*, a menos que haja um desvio, então neste caso a próxima instrução será o endereço de desvio *(PCOut = PCIn)*.

2.1.4 Memória de Instruções

A memória de instruções só precisa fornecer acesso de leitura porque o caminho de dados não escreve instruções. Como a memória de instruções apenas é lida, é tratada como lógica combinatória: a saída em qualquer momento reflete o conteúdo do local especificado pela entrada de endereço, e nenhum sinal de controle de leitura é necessário(5). O bloco da memória de instruções, onde estão as instruções que serão realizadas. O *Endereço* de entrada foi calculado no PC e a partir dele sabe a instrução a ser realizada.

2.1.5 Banco de Registradores

Um Banco de Registradores é uma coleção de registradores em que qualquer registrador pode ser lido ou escrito especificando o número do mesmo a ser acessado(5). No banco de registradores, pode ser realizado a escrita em um registrador e/ou a leitura de até dois registradores.

2.1.6 Unidades Lógica e Aritmética

A unidade lógica e aritmética (ULA) é a responsável pelas operações nas instruções, seja elas o cálculo do desvio, as operações aritméticas, lógicas e entre outras. A Unidade Lógica Aritmética (ULA), onde realiza as operações lógicas e aritméticas como, soma, subtração, multiplicação, divisão, E, OU e entre outras. O *Resultado* é o resultado da operação, *Dado1 e Dado2* são os dados que serão utilizados na operação, enquanto o *binario* indica quando haverá um desvio.

2.1.7 Memória de Dados

É um elemento de estado com entradas para os endereços e os dados de escrita, e uma única saída para o resultado da leitura(5). A memória de dados pode escrever ou ler um dado, dependendo dos sinais de entrada.

2.1.8 Unidade de Saída

Faz o contato com a interface de saída, onde pode mostrar o dado obtido na instrução ou ser *reset*.

2.1.9 Extensor de Sinal

É uma unidade, onde aumenta o tamanho de um item de dados replicando o bit mais alto de sinal do item de dados original nos bits mais altos do item de dados maior de destino(5). Por exemplo, a entrada fornecida pelo usuário é de no máximo 16 bits, porém os dados utilizados no processador são de 32 bits. Para estender o sinal de entrada, o sinal de saída é igual ao de entrada mais 16 bits.

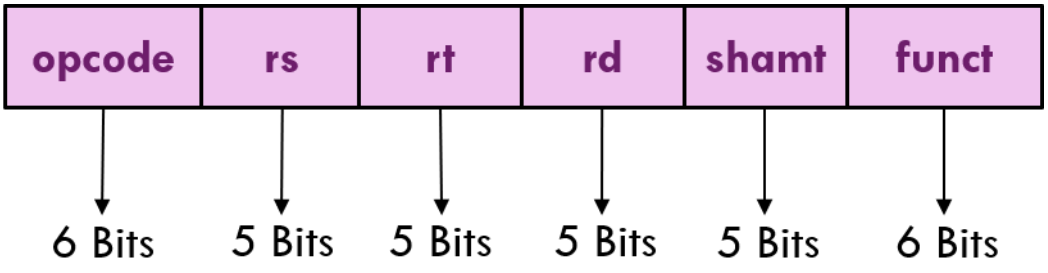
2.2 Instruções

2.2.1 Formato das Instruções

O processador utiliza de três formatos de instruções, sendo eles do tipo R, I e J conforme mostra na Figura 5, Figura 6 e Figura 7. Há 32 registradores de 32 bits, a alocação destes bits está explicado abaixo.

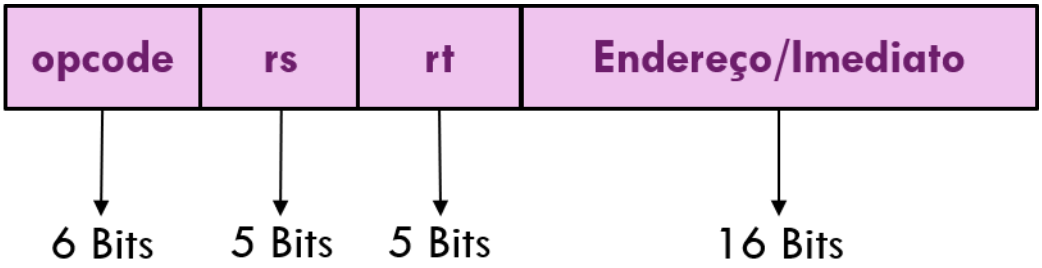
- *Opcode* -> Código referente a cada instrução
- rs e rt -> Endereço dos registradores fontes
- rd -> Endereço do registrador destino
- *Shamt* -> Para instruções de deslocamento
- *Funct* -> A operação a ser realizada
- Endereço -> Endereço da memória

Figura 5 – Formato do tipo R



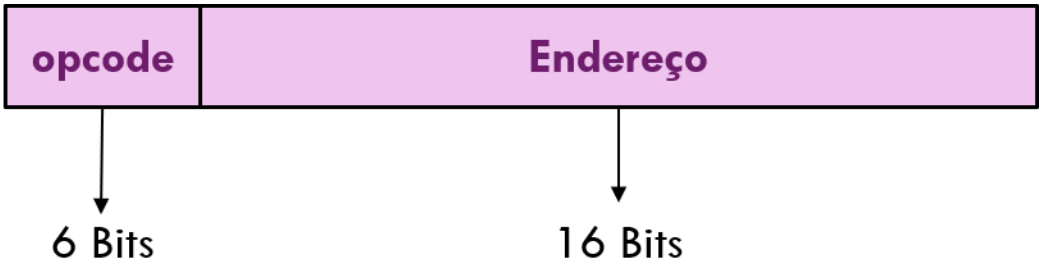
Fonte: Autoria Própria

Figura 6 – Formato do tipo I



Fonte: Autoria Própria

Figura 7 – Formato do tipo J



Fonte: Autoria Própria

2.2.2 Conjunto de Instruções

Na Tabela 1 mostra as 38 instruções definidas para este processador, a primeira coluna está o *OPCode*, código que o Controle identifica qual instrução será executada naquele momento, a segunda está o nome da instrução, na terceira, está o tipo da instrução e por último a quarta mostra qual a operação a ser realizada e na última coluna há uma breve descrição de sua função. Possui instruções aritméticas, lógicas, de deslocamento, desvio incondicional, desvio condicional, acesso à memória, de controle e entrada/saída.

Tabela 1 – Conjunto de Instruções

Opcode	Instrução	Tipo	Operação	Descrição
Aritméticas				
000000	add	R	RD = RS + RT	soma
000001	addi	I	RT = RS + IMEDIATO	soma com imediato
000010	sub	R	RD = RS - RT	subtração
000011	subi	I	RT = RS - IMEDIATO	subtração com imediato
000100	mul	R	RD = RS * RT	multiplicação
000101	muli	I	RT = RS * IMEDIATO	multiplicação com imediato
000110	div	R	RD = RS / RT	divisão
000111	divi	I	RT = RS / IMEDIATO	divisão com imediato
Lógicas				
001000	and	R	RD = RS & RT	E
001001	andi	I	RT = RS & IMEDIATO	E com imediato
001010	or	R	RD = RS | RT	OU
001011	ori	I	RT = RS | IMEDIATO	OU com imediato
001100	xor	R	RD = RS ^ RT	ou exclusivo
001101	xori	I	RT = RS ^ IMEDIATO	ou exclusivo com imediato
001110	not	I	RT = ~ RS	valor negado
Deslocamento				
001111	sll	R	RD = RS « shamt	Deslocamento a esquerda
010000	srl	R	RD = RS » shamt	Deslocamento a direita
Desvio Incondicional				
010001	jump	J	CP = novo CP	Pula para o endereço
010010	jal	J	$ra = CP; CP = novo CP	Pula e armazena endereço de retorno
010011	jr	J	CP = endereço de $ra	pula para registrador
Desvio Condicional				
010100	beq	I	se RS == RT, CP = desvio	Desvia se igual
010101	bnq	I	se RS != RT, CP = desvio	Desvia se diferente
010110	blt	I	se RS < RT, CP = desvio	Desvia se menor
010111	bgt	I	se RS > RT, CP = desvio	Desvia de maior
011000	bltz	I	se RS < 0, CP = desvio	Desvia se menor que zero
011001	bgtz	I	se RS > 0, CP = desvio	Desvia se maior que zero
011010	slt	R	RD = RS < RT?	RD igual a 1 se menor, 0 se maior ou igual
011011	slet	R	RD = RS <= RT?	RD igual a 1 se menor ou igual, 0 se maior
011100	sgt	R	RD = RS > RT?	RD igual a 1 se maior, 0 se menor ou igual
011101	sget	R	RD = RS >= RT?	RD igual a 1 se maior ou igual, 0 se maior
Acesso a Memória				
011110	li	I	RT = Imediato	igual a imediato
011111	lw	I	RT = Mem [RS + offset]	Carrega uma palavra
100000	sw	I	Mem[RS + offset] = RT	Armazena uma palavra
100001	move	I	RT = RS	igual ao valor do registrador
Controle				
100010	nop	J	-	Sem operação para executar
100011	hlt	J	-	Parar processamento
Entrada e Saída				
100100	in	-	-	Sinal de entrada
100101	out	-	-	Sinal de saída

Fonte: Autoria Própria

2.3 Organização da Memória

O processador há 3 categorias de armazenamento, sendo eles a memória de instruções, a memória de dados e o banco de registradores.

A memória de instruções é guardado as instruções em que o programa está executando e então é direcionado pelo registrador qual a instrução que deverá ser decodificada e executada. Memória de dados armazena informações de 32 bits e usado para guardar dados que não serão usados constantemente em operações do programa em cada posição, o seu endereçamento é acessado através da soma de um endereço contido em um registrador base, especificado com um valor de deslocamento. E o banco de registradores possui registradores que armazenam dados que estão sendo manipulados no momento e constantemente pelo processador.

3 O Compilador

Compilador tem o objetivo de traduzir um código-fonte para um código alvo e para isso é dividido em duas partes, a primeira é a fase de análise 3.1, onde verifica se há algum erro no código e a segunda é a fase de síntese 3.2, onde faz a tradução do código-fonte para o código alvo.

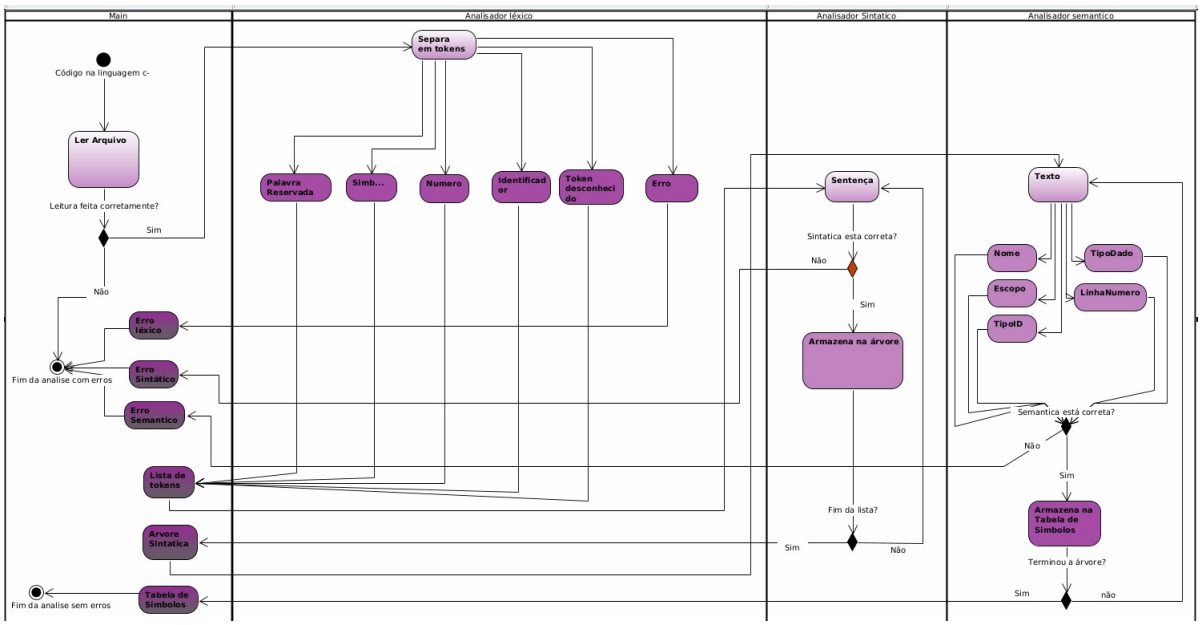
3.1 Fase de Análise

Na fase de análise é feito a análise léxica 3.1.2, sintática 3.1.3 e semântica 3.1.4. O objetivo é verificar a existência de erros antes de passar para fase de tradução do código, para isso é feito a análise das palavras, gerando uma lista de *tokens*, para ser utilizada na análise da frase, qual gera uma árvore sintática para então ser analisado o texto por completo que gera uma tabela de símbolos e caso não encontre nenhum erro, é por fim passado para fase de síntese.

3.1.1 Modelagem

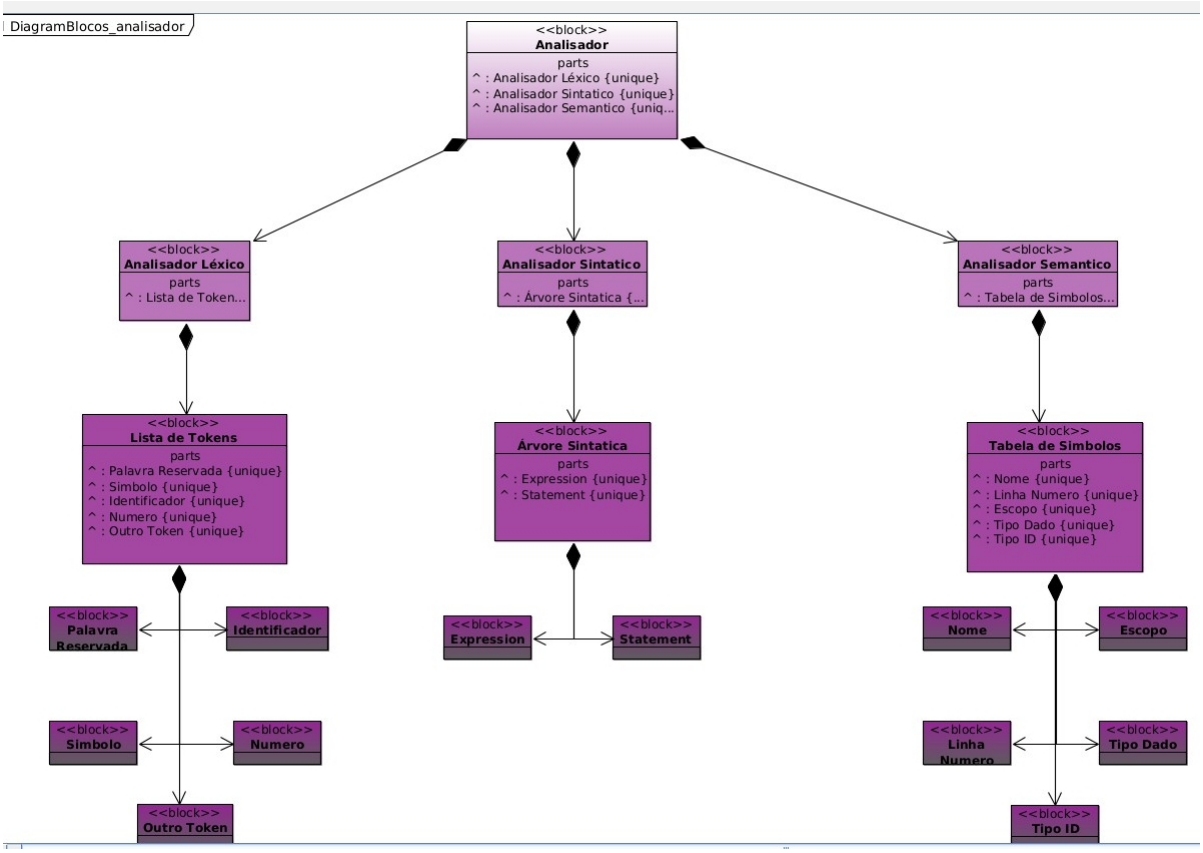
Para explicação da fase de analise foi feito dois diagramas estando na figura 8 e figura 9

Figura 8 – Diagrama de Atividades - Fase de análise



Fonte: Autoria Própria

Figura 9 – Diagrama de blocos - Fase de análise



Fonte: Autoria Própria

3.1.2 Análise Léxica

O analisador léxico interage com o analisador sintático enviando uma lista de *tokens*, porém se um erro léxico é encontrado, toda a execução da análise é interrompida e o erro que gerou a interrupção é indicado na aplicação que executa o flex. Então o objetivo desta análise é verificar se há algum erro nas palavras utilizadas no código-fonte.

3.1.3 Análise Sintática

A 2ª fase é o analisador sintático, em que analisa a frase do código-fonte. Nesta fase é utilizado a ferramenta *Yacc Bison*, qual recebe os *tokens* do analisador léxico e constrói uma árvore de análise sintática.

Caso uma regra não seja obedecida à aplicação é interrompida e o erro é indicado, caso contrário é concluído a construção da árvore de análise sintática em C usada em outras partes do compilador.

3.1.4 Análise Semântica

Na parte da análise semântica é gerada uma tabela de símbolos onde são descritos informações importantes para a análise. É implementada através de uma tabela hash que a cada inserção de nela é verificado se o símbolo já existe na tabela caracterizando um erro. Outros erros de caráter semântico também são verificados, como usar uma variável ou função que não foi declarada, declarar duas vezes a mesma variável, etc. Esta análise é feita sem o uso de uma ferramenta específica, usa-se um código em c que percorre a árvore sintática e constrói a tabela de símbolos para fazer esta análise.

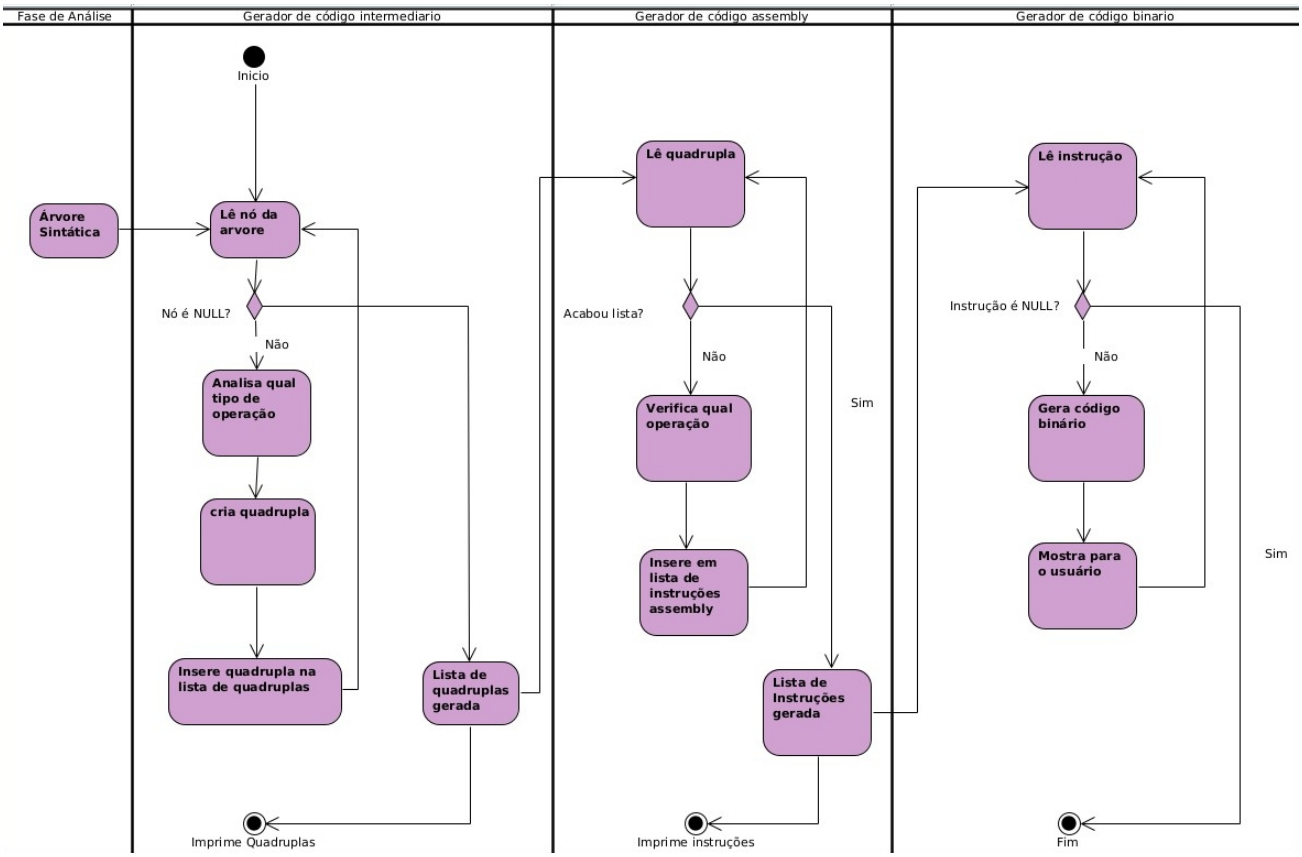
3.2 Fase de Síntese

A fase de síntese do compilador é responsável por traduzir o código-fonte no código-alvo, que no caso deste projeto é o código de máquina e o código binário. Para isso é necessário receber da fase de análise, a árvore sintática e a tabela de símbolos.

O primeiro passo é linearizar a árvore sintática e gerar o código intermediário, a partir disso é traduzido para o código assembly e o executável, considerando características de gerenciamento de memória.

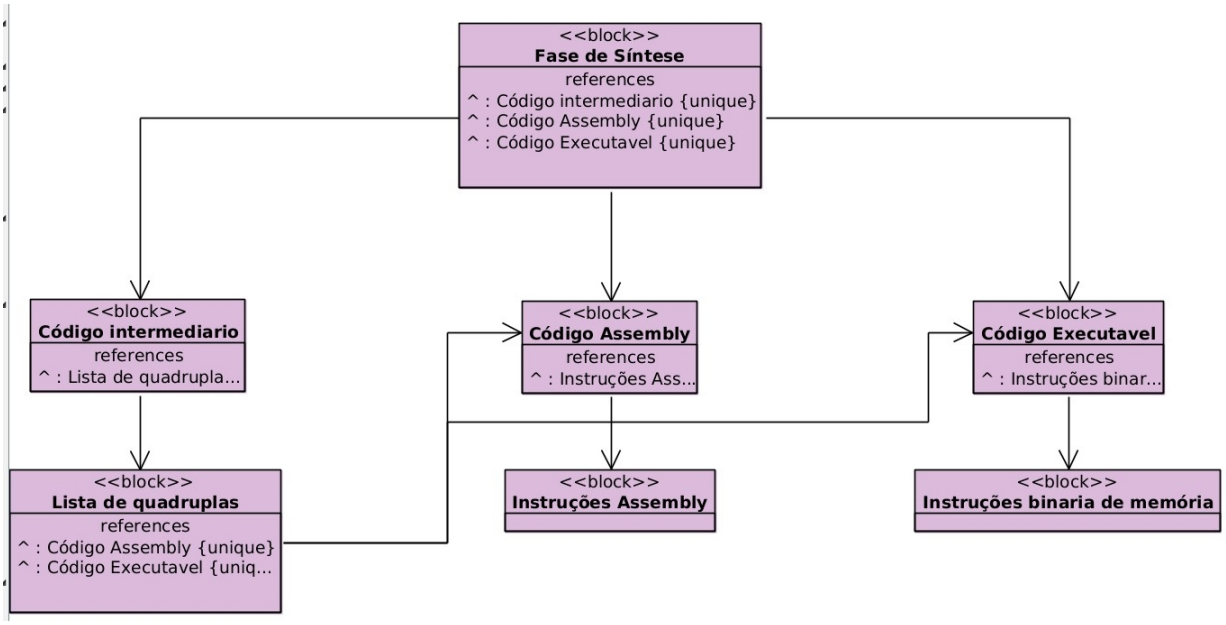
3.2.1 Modelagem

Figura 10 – Diagrama de Atividades - Fase de síntese



Fonte: Autoria Própria

Figura 11 – Diagrama de Blocos - Fase de síntese



Fonte: Autoria Própria

3.2.2 Geração do Código Intemerdiário

Geração do código intermediário é a primeira etapa da fase síntese, ela é responsável por linearizar a árvore de análise sintática. O algoritmo dessa etapa lê cada nó da árvore e faz a análise conforme as características desse nó, pois cada categoria de nó recebe um tratamento diferente, e então realiza a inserção de quádruplas na lista de quádruplas, depois é consultado a tabela de símbolos para usar o campo de endereço alocado para a variável, o qual será necessário para carregar e descarregar dados da memória. O resultado da geração do código intermediário é uma lista encadeada de quádruplas com seus campos definidos conforme a estrutura de cada quádrupla. Cada quadrupla possui 4 campos, sendo 3 símbolos e 1 operação. Neste projeto há 26 operações:

- <: Compara se um valor é menor que o outro.
- >: Compara se um valor é maior que o outro.
- <=: Compara se um valor é menor ou igual ao outro.
- >=:Compara se um valor é maior ou igual ao outro.
- ==: Compara se um valor é igual que o outro.
- !=: Compara se um valor é diferente que o outro.
- +: Faz a soma de dois valores.
- -: Faz a subtração de dois valores.
- *: Faz a multiplicação de dois valores.
- /: Faz a divisão de dois valores.
- nop: Instrução sem operação.
- halt: Encerra o processador.
- store: Um dado do registrador é salvo na memória.
- fun: Declara uma função.
- arg: Declara um argumento.
- alloc: Declara uma variável.
- setArg: Passa um dado por parâmetro em uma função.
- call: Ativa uma função.
- load: Carrega um dado da memória para um registrador.

- ret: Retorno de uma função
- label: Salta de uma linha para outra.
- beq: Se os valores forem iguais faz um salto.
- bne: Se os valores não forem iguais faz um salto.
- jump: faz um salto.
- imed: registra um valor imediato em um registrador.
- end: Finaliza uma função.

3.2.3 Geração do Código Assembly

Código Assembly é um código legível por humanos para linguagem de máquina, então para fazer a geração do código é feito a tradução do código intermediário para o assembly. O primeiro passo é percorrer a lista de quadruplas gerada pelo código intermediário e a cada quadrupla é avaliada o categoria de operação para identificar a instrução e verificada na tabela de símbolos para poder fazer o gerenciamento de memória, no final é gerado uma lista encadeada de instruções. O código para realizar a geração do código assembly está nos apêndices C e D.

3.2.4 Geração do Código Executável

A geração do código executável é feita através da leitura da lista de instruções em assembly e assim conforme as especificações da memória do processador é gerado um código binário para cada instrução. O código binário é o código que o processador consegue interpretar, ou seja, são instruções gravadas na memória de instruções. Os códigos para geração do código executável esta nos apêndices G e H.

3.2.5 Gerenciamento de Memória

Há uma lista com endereços de memória gerada no código executável, onde através dela é possível localizar na memória. O único gerenciamento é na alocação de memória na tabela de símbolos, onde para cada variável é associado a um número inteiro que corresponde á localização da variável na memória.

4 Exemplos

Para testar o que foi explicado acima, foi utilizado 3 exemplos, onde cada exemplo foi gerado seu respectivo código intermediário, assembly e o executável.

4.1 Exemplo fatorial

O primeiro exemplo é o 4.1, onde calcula o fatorial de um número, fatorial é obtido a partir da multiplicação de todos os seus antecessores até o número um, portanto na função *main()* cria a variável x que recebe um número qualquer do usuário e retorna a função *fat()*, nela faz uma verificação se x é igual á zero, se for retorna o valor 1, caso contrário calcula a multiplicação com seu antecessor de forma recursiva, assim o valor retornado pelo código será: *x * (x-1) * (x-2) * ... * 1*, onde x é um número qualquer. Com este exemplo de teste resultou os exemplos 4.3.1, 4.3.2 e 4.3.3

```
1  int fat ( int x ) {
2      if ( x == 0) {
3          return 1;
4      } else {
5          return x * fat (x -1) ;
6      }
7  }
8
9  void main ( void ) {
10     int x ;
11     x = input () ;
12     output ( fat ( x ) ,0) ;
13 }
```

Listing 4.1 – Código Fatorial

4.1.1 Código Intemerdiário

```
1  ( nop   ,          ,          ,          )
2  ( fun   , fat      ,          ,          )
3  ( arg   , x        ,          , fat      )
4  ( load  , t_1      , x        ,          )
5  ( imed  , t_2      ,          , 0        )
6  ( ==    , t_3      , t_1      , t_2      )
7  ( beq   , t_3      , t_0      , L_2      )
8  ( imed  , t_1      ,          , 1        )
9  ( ret   , t_1      ,          ,          )
10 ( jump  ,          ,          , L_3      )
11 ( label ,          ,          , L_2      )
12 ( load  , t_2      , x        ,          )
13 ( load  , t_3      , x        ,          )
14 ( imed  , t_1      ,          , 1        )
15 ( -     , t_2      , t_3      , t_1      )
16 ( setArg,          ,          , t_2      )
```

```
( call  , t_3       ,           , fat       )
( *     , t_4       , t_2       , t_3       )
( ret   , t_4       ,           ,           )
( label ,           ,           , L_3       )
( end   , fat       ,           ,           )
( fun   , main      ,           ,           )
( alloc , x         , 1         , main      )
( call  , t_5       ,           , input     )
( store , x         ,           , t_5       )
( load  , t_6       , x         ,           )
( setArg,           ,           , t_6       )
( call  , t_7       ,           , fat       )
( setArg,           ,           , t_7       )
( imed  , t_8       ,           , 0         )
( setArg,           ,           , t_8       )
( call  ,           ,           , output    )
( end   , main      ,           ,           )
( halt  ,           ,           ,           )
```

Listing 4.2 – Código Fatorial - Quadruplas do Código Intermediário

4.1.2 Código Assembly

```
nop
fat:

        li $r27, 3
        lw $r1, $r27
        li $r2, 0
        set $r3, $r1, $r2
        beq $r3, $r0, L2
        li $r1, 1
        move $r28, $r1
        jr $r31
        jump L3:
        L2:
            li $r27, 3
            lw $r2, $r27
            li $r27, 3
            lw $r3, $r27
            li $r1, 1
            li $r2, $r3, $r1
            move $r22, $r2
            jal fat
            move $r3, $r28
            mult $r4, $r2, $r3
            move $r28, $r4
            jr $r31
        L3:
            jr $r31
main:

        in
        move $r5, $r30
        li $r27, 3
        sw $r27, $r5
        li $r27, 3
        lw $r6, $r27
```

```
36          move $r23, $r6
37          jal fat
38          move $r7, $r28
39          move $r24, $r7
40          li $r8, 0
41          move $r25, $r8
42          out $r22
43          halt
```

Listing 4.3 – Código Fatorial - Assembly

4.1.3 Código Executável

```
1           assign memoria[0]={5'd23, 27'd0 }
2           assign memoria[1]={5'd23, 27'd0 }
3           assign memoria[2]={5'd17, 5'd27, 22'd3 }
4           assign memoria[3]={5'd16, 5'd1, 5'd27, 17'd0 }
5           assign memoria[4]={5'd17, 5'd2, 22'd0 }
6           assign memoria[5]={5'd5, 5'd3, 5'd1, 5'd2, 12'd0 }
7           assign memoria[6]={5'd11, 5'd3, 5'd0, 17'd2 }
8           assign memoria[7]={5'd17, 5'd1, 22'd0 }
9           assign memoria[8]={5'd19, 5'd28, 5'd1, 17'd0 }
10          assign memoria[9]={5'd14, 5'd31, 22'd0 }
11          assign memoria[10]={5'd13, 27'd0 }
12          assign memoria[11]={5'd23, 27'd0 }
13          assign memoria[12]={5'd17, 5'd27, 22'd3 }
14          assign memoria[13]={5'd16, 5'd2, 5'd27, 17'd0 }
15          assign memoria[14]={5'd17, 5'd27, 22'd3 }
16          assign memoria[15]={5'd16, 5'd3, 5'd27, 17'd0 }
17          assign memoria[16]={5'd17, 5'd1, 22'd0 }
18          assign memoria[17]={5'd17, 5'd1, 22'd0 }
19          assign memoria[18]={5'd19, 5'd22, 5'd2, 17'd0 }
20          assign memoria[19]={5'd15, 27'd0 }
21          assign memoria[20]={5'd19, 5'd3, 5'd28, 17'd0 }
22          assign memoria[21]={5'd3, 5'd4, 5'd2, 5'd3, 12'd0 }
23          assign memoria[22]={5'd19, 5'd28, 5'd4, 17'd0 }
24          assign memoria[23]={5'd14, 5'd31, 22'd0 }
25          assign memoria[24]={5'd23, 27'd0 }
26          assign memoria[25]={5'd14, 5'd31, 22'd0 }
27          assign memoria[26]={5'd23, 27'd0 }
28          assign memoria[27]={5'd20, 27'd0 }
29          assign memoria[28]={5'd19, 5'd5, 5'd30, 17'd0 }
30          assign memoria[29]={5'd17, 5'd27, 22'd3 }
31          assign memoria[30]={5'd18, 5'd27, 5'd5, 17'd0 }
32          assign memoria[31]={5'd17, 5'd27, 22'd3 }
33          assign memoria[32]={5'd16, 5'd6, 5'd27, 17'd0 }
34          assign memoria[33]={5'd19, 5'd23, 5'd6, 17'd0 }
35          assign memoria[34]={5'd15, 27'd0 }
36          assign memoria[35]={5'd19, 5'd7, 5'd28, 17'd0 }
37          assign memoria[36]={5'd19, 5'd24, 5'd7, 17'd0 }
38          assign memoria[37]={5'd17, 5'd8, 22'd0 }
39          assign memoria[38]={5'd19, 5'd25, 5'd8, 17'd0 }
40          assign memoria[39]={5'd24, 5'd22, 22'd0 }
41          assign memoria[40]={5'd21, 27'd0 }
```

Listing 4.4 – Código Fatorial - Executável

4.2 Exemplo Sort

O segundo exemplo é o Sort, no qual é um método de ordenação, onde ordena em ordem crescente um vetor de 10 posições. O primeiro na função main() é ler os valores do vetor passado pelo usuário, então é chamado a função sort(), onde a cada posição do vetor é chamado a unção minloc() onde retorna o menor valor do vetor, então ao final terá um vetor ordenado.

```
1  /* programa para ordena  o por sele  o de
2     uma matriz com dez elementos. */
3
4  int vet[ 10 ];
5
6  int minloc ( int a[], int low, int high )
7  {       int i; int x; int k;
8          k = low;
9          x = a[low];
10         i = low + 1;
11         while (i < high){
12                 if (a[i] < x){
13                         x = a[i];
14                         k = i;
15                 }
16                 i = i + 1;
17         }
18         return k;
19 }
20
21 void sort( int a[], int low, int high)
22 {       int i; int k;
23         i = low;
24         while (i < high-1){
25                 int t;
26                 k = minloc(a,i,high);
27                 t = a[k];
28                 a[k] = a[i];
29                 a[i] = t;
30                 i = i + 1;
31         }
32 }
33
34 void main(void)
35 {
36         int i;
37         i = 0;
38         while (i < 10){
39                 vet[i] = input();
40                 i = i + 1;
41         }
42         sort(vet,0,10);
43         i = 0;
44         while (i < 10){
45                 output(vet[i]);
46                 i = i + 1;
47         }
```

```
48 }
```

Listing 4.5 – Código Sort

4.2.1 Código Intemerdiário

```
1  ( nop   ,          ,          ,          )
2  ( alloc , vet      , 10       , global   )
3  ( fun   , minloc   ,          ,          )
4  ( arg   , a        ,          , minloc   )
5  ( arg   , low      ,          , minloc   )
6  ( arg   , high     ,          , minloc   )
7  ( alloc , i        , 1        , minloc   )
8  ( alloc , x        , 1        , minloc   )
9  ( alloc , k        , 1        , minloc   )
10 ( load  , t_1      , low      ,          )
11 ( store , k        ,          , t_1      )
12 ( load  , t_2      , low      ,          )
13 ( load  , t_3      , a        , t_2      )
14 ( store , x        ,          , t_3      )
15 ( load  , t_1      , low      ,          )
16 ( imed  , t_2      ,          , 1        )
17 ( +     , t_3      , t_1      , t_2      )
18 ( store , i        ,          , t_3      )
19 ( label ,          ,          , L_2      )
20 ( load  , t_1      , i        ,          )
21 ( load  , t_2      , high     ,          )
22 ( <     , t_3      , t_1      , t_2      )
23 ( beq   , t_3      , t_0      , L_3      )
24 ( load  , t_1      , i        ,          )
25 ( load  , t_2      , a        , t_1      )
26 ( load  , t_3      , x        ,          )
27 ( <     , t_4      , t_2      , t_3      )
28 ( beq   , t_4      , t_0      , L_4      )
29 ( load  , t_5      , i        ,          )
30 ( load  , t_6      , a        , t_5      )
31 ( store , x        ,          , t_6      )
32 ( load  , t_7      , i        ,          )
33 ( store , k        ,          , t_7      )
34 ( label ,          ,          , L_4      )
35 ( load  , t_8      , i        ,          )
36 ( imed  , t_9      ,          , 1        )
37 ( +     , t_10     , t_8      , t_9      )
38 ( store , i        ,          , t_10     )
39 ( jump  ,          ,          , L_2      )
40 ( label ,          ,          , L_3      )
41 ( load  , t_11     , k        ,          )
42 ( ret   , t_11     ,          ,          )
43 ( end   , minloc   ,          ,          )
44 ( fun   , sort     ,          ,          )
45 ( arg   , a        ,          , sort     )
46 ( arg   , low      ,          , sort     )
47 ( arg   , high     ,          , sort     )
48 ( alloc , i        , 1        , sort     )
49 ( alloc , k        , 1        , sort     )
50 ( load  , t_12     , low      ,          )
51 ( store , i        ,          , t_12     )
52 ( label ,          ,          , L_6      )
```

```
53  ( load  , t_13      , high      ,           )
54  ( imed  , t_14      ,           , 1         )
55  ( -     , t_15      , t_13      , t_14      )
56  ( load  , t_16      , i         ,           )
57  ( <     , t_17      , t_16      , t_15      )
58  ( beq   , t_17      , t_0       , L_7       )
59  ( alloc , t         , 1         , sort      )
60  ( load  , t_18      , a         ,           )
61  ( setArg,           ,           , t_18      )
62  ( load  , t_19      , i         ,           )
63  ( setArg,           ,           , t_19      )
64  ( load  , t_20      , high      ,           )
65  ( setArg,           ,           , t_20      )
66  ( call  , t_21      ,           , minloc    )
67  ( store , k         ,           , t_21      )
68  ( load  , t_22      , k         ,           )
69  ( load  , t_23      , a         , t_22      )
70  ( store , t         ,           , t_23      )
71  ( load  , t_24      , i         ,           )
72  ( load  , t_25      , a         , t_24      )
73  ( load  , t_26      , k         ,           )
74  ( store , a         , t_26      , t_25      )
75  ( load  , t_27      , t         ,           )
76  ( load  , t_28      , i         ,           )
77  ( store , a         , t_28      , t_27      )
78  ( load  , t_29      , i         ,           )
79  ( imed  , t_30      ,           , 1         )
80  ( +     , t_31      , t_29      , t_30      )
81  ( store , i         ,           , t_31      )
82  ( jump  ,           ,           , L_6       )
83  ( label ,           ,           , L_7       )
84  ( end   , sort      ,           ,           )
85  ( fun   , main      ,           ,           )
86  ( alloc , i         , 1         , main      )
87  ( imed  , t_32      ,           , 0         )
88  ( store , i         ,           , t_32      )
89  ( label ,           ,           , L_9       )
90  ( load  , t_33      , i         ,           )
91  ( imed  , t_34      ,           , 10        )
92  ( <     , t_35      , t_33      , t_34      )
93  ( beq   , t_35      , t_0       , L_10      )
94  ( call  , t_36      ,           , input     )
95  ( load  , t_37      , i         ,           )
96  ( store , vet       , t_37      , t_36      )
97  ( load  , t_38      , i         ,           )
98  ( imed  , t_39      ,           , 1         )
99  ( +     , t_40      , t_38      , t_39      )
100 ( store , i         ,           , t_40      )
101 ( jump  ,           ,           , L_9       )
102 ( label ,           ,           , L_10      )
103 ( load  , t_41      , vet       ,           )
104 ( setArg,           ,           , t_41      )
105 ( imed  , t_42      ,           , 0         )
106 ( setArg,           ,           , t_42      )
107 ( imed  , t_43      ,           , 10        )
108 ( setArg,           ,           , t_43      )
109 ( call  ,           ,           , sort      )
110 ( imed  , t_44      ,           , 0         )
111 ( store , i         ,           , t_44      )
112 ( label ,           ,           , L_11      )
```

```
113 ( load  , t_45      , i         ,           )
114 ( imed  , t_46      ,           , 10        )
115 ( <     , t_47      , t_45      , t_46      )
116 ( beq   , t_47      , t_0       , L_12      )
117 ( load  , t_48      , i         ,           )
118 ( load  , t_49      , vet       , t_48      )
119 ( setArg,           ,           , t_49      )
120 ( call  ,           ,           , output    )
121 ( load  , t_50      , i         ,           )
122 ( imed  , t_51      ,           , 1         )
123 ( +     , t_52      , t_50      , t_51      )
124 ( store , i         ,           , t_52      )
125 ( jump  ,           ,           , L_11      )
126 ( label ,           ,           , L_12      )
127 ( end   , main      ,           ,           )
128 ( halt  ,           ,           ,           )
```

Listing 4.6 – Código Sort

4.2.2 Código Assembly

```
1  nop
2  minloc:
3
4          li $r27, 10
5          lw $r1, $r27
6          li $r27, 13
7          sw $r27, $r1
8          li $r27, 10
9          lw $r2, $r27
10         li $r27, 9
11         add $r27, $r27, $r2
12         lw $r3, $r27
13         li $r27, 6
14         sw $r27, $r3
15         li $r27, 10
16         lw $r1, $r27
17         li $r2, 1
18         add $r3, $r1, $r2
19         li $r27, 16
20         sw $r27, $r3
21         L2:
22             li $r27, 16
23             lw $r1, $r27
24             li $r27, 11
25             lw $r2, $r27
26             slt $r3, $r1, $r2
27             beq $r3, $r0, L3
28             li $r27, 16
29             lw $r1, $r27
30             li $r27, 9
31             add $r27, $r27, $r1
32             lw $r2, $r27
33             li $r27, 6
34             lw $r3, $r27
35             slt $r4, $r2, $r3
36             beq $r4, $r0, L4
37             li $r27, 16
```

```
38              lw $r5, $r27
39              li $r27, 9
40              add $r27, $r27, $r5
41              lw $r6, $r27
42              li $r27, 6
43              sw $r27, $r6
44              li $r27, 16
45              lw $r7, $r27
46              li $r27, 13
47              sw $r27, $r7
48          L4:
49              li $r27, 16
50              lw $r8, $r27
51              li $r9, 1
52              add $r10, $r8, $r9
53              li $r27, 16
54              sw $r27, $r10
55              jump L2:
56          L3:
57              li $r27, 13
58              lw $r11, $r27
59              move $r28, $r11
60              jr $r31
61              jr $r31
62  sort:
63
64          li $r27, 10
65          lw $r12, $r27
66          li $r27, 16
67          sw $r27, $r12
68          L6:
69              li $r27, 11
70              lw $r13, $r27
71              li $r14, 1
72              li $r15, $r13, $r14
73              li $r27, 16
74              lw $r16, $r27
75              slt $r17, $r16, $r15
76              beq $r17, $r0, L7
77              li $r27, 9
78              lw $r18, $r27
79              move $r22, $r18
80              li $r27, 16
81              lw $r19, $r27
82              move $r23, $r19
83              li $r27, 11
84              lw $r20, $r27
85              move $r24, $r20
86              jal minloc
87              move $r21, $r28
88              li $r27, 13
89              sw $r27, $r21
90              li $r27, 13
91              lw $r22, $r27
92              li $r27, 9
93              add $r27, $r27, $r22
94              lw $r23, $r27
95              li $r27, 14
96              sw $r27, $r23
97              li $r27, 16
```

```
98              lw $r24, $r27
99              li $r27, 9
100             add $r27, $r27, $r24
101             lw $r25, $r27
102             li $r27, 13
103             lw $r26, $r27
104             li $r27, 9
105             add $r27, $r27, $r26
106             sw $r27, $r25
107             li $r27, 14
108             lw $r27, $r27
109             li $r27, 16
110             lw $r28, $r27
111             li $r27, 9
112             add $r27, $r27, $r28
113             sw $r27, $r27
114             li $r27, 16
115             lw $r29, $r27
116             li $r30, 1
117             add $r31, $r29, $r30
118             li $r27, 16
119             sw $r27, $r31
120             jump L6:
121         L7:
122             jr $r31
123 main:
124
125         li $r32, 0
126         li $r27, 16
127         sw $r27, $r32
128         L9:
129             li $r27, 16
130             lw $r33, $r27
131             li $r34, 10
132             slt $r35, $r33, $r34
133             beq $r35, $r0, L10
134             in
135             move $r36, $r30
136             li $r27, 16
137             lw $r37, $r27
138             li $r27, 0
139             add $r27, $r27, $r37
140             sw $r27, $r36
141             li $r27, 16
142             lw $r38, $r27
143             li $r39, 1
144             add $r40, $r38, $r39
145             li $r27, 16
146             sw $r27, $r40
147             jump L9:
148         L10:
149             li $r27, 0
150             lw $r41, $r27
151             move $r25, $r41
152             li $r42, 0
153             move $r26, $r42
154             li $r43, 10
155             move $r27, $r43
156             jal sort
157             li $r44, 0
```

```
158             li $r27, 16
159             sw $r27, $r44
160         L11:
161             li $r27, 16
162             lw $r45, $r27
163             li $r46, 10
164             slt $r47, $r45, $r46
165             beq $r47, $r0, L12
166             li $r27, 16
167             lw $r48, $r27
168             li $r27, 0
169             add $r27, $r27, $r48
170             lw $r49, $r27
171             move $r28, $r49
172             out $r22
173             li $r27, 16
174             lw $r50, $r27
175             li $r51, 1
176             add $r52, $r50, $r51
177             li $r27, 16
178             sw $r27, $r52
179             jump L11:
180         L12:
181             halt
```

Listing 4.7 – Código Sort

4.2.3 Código Executável

```
1         assign memoria[0]={5'd23, 27'd0 }
2         assign memoria[1]={5'd23, 27'd0 }
3         assign memoria[2]={5'd17, 5'd27, 22'd10 }
4         assign memoria[3]={5'd16, 5'd1, 5'd27, 17'd0 }
5         assign memoria[4]={5'd17, 5'd27, 22'd13 }
6         assign memoria[5]={5'd18, 5'd27, 5'd1, 17'd0 }
7         assign memoria[6]={5'd17, 5'd27, 22'd10 }
8         assign memoria[7]={5'd16, 5'd2, 5'd27, 17'd0 }
9         assign memoria[8]={5'd17, 5'd27, 22'd9 }
10        assign memoria[9]={5'd1, 5'd27, 5'd27, 5'd2, 12'd0 }
11        assign memoria[10]={5'd16, 5'd3, 5'd27, 17'd0 }
12        assign memoria[11]={5'd17, 5'd27, 22'd6 }
13        assign memoria[12]={5'd18, 5'd27, 5'd3, 17'd0 }
14        assign memoria[13]={5'd17, 5'd27, 22'd10 }
15        assign memoria[14]={5'd16, 5'd1, 5'd27, 17'd0 }
16        assign memoria[15]={5'd17, 5'd2, 22'd0 }
17        assign memoria[16]={5'd1, 5'd3, 5'd1, 5'd2, 12'd0 }
18        assign memoria[17]={5'd17, 5'd27, 22'd16 }
19        assign memoria[18]={5'd18, 5'd27, 5'd3, 17'd0 }
20        assign memoria[19]={5'd23, 27'd0 }
21        assign memoria[20]={5'd17, 5'd27, 22'd16 }
22        assign memoria[21]={5'd16, 5'd1, 5'd27, 17'd0 }
23        assign memoria[22]={5'd17, 5'd27, 22'd11 }
24        assign memoria[23]={5'd16, 5'd2, 5'd27, 17'd0 }
25        assign memoria[24]={5'd9, 5'd3, 5'd1, 5'd2, 12'd0 }
26        assign memoria[25]={5'd11, 5'd3, 5'd0, 17'd3 }
27        assign memoria[26]={5'd17, 5'd27, 22'd16 }
28        assign memoria[27]={5'd16, 5'd1, 5'd27, 17'd0 }
29        assign memoria[28]={5'd17, 5'd27, 22'd9 }
```

```
30          assign memoria[29]={5'd1, 5'd27, 5'd27, 5'd1, 12'd0 }
31          assign memoria[30]={5'd16, 5'd2, 5'd27, 17'd0 }
32          assign memoria[31]={5'd17, 5'd27, 22'd6 }
33          assign memoria[32]={5'd16, 5'd3, 5'd27, 17'd0 }
34          assign memoria[33]={5'd9, 5'd4, 5'd2, 5'd3, 12'd0 }
35          assign memoria[34]={5'd11, 5'd4, 5'd0, 17'd4 }
36          assign memoria[35]={5'd17, 5'd27, 22'd16 }
37          assign memoria[36]={5'd16, 5'd5, 5'd27, 17'd0 }
38          assign memoria[37]={5'd17, 5'd27, 22'd9 }
39          assign memoria[38]={5'd1, 5'd27, 5'd27, 5'd5, 12'd0 }
40          assign memoria[39]={5'd16, 5'd6, 5'd27, 17'd0 }
41          assign memoria[40]={5'd17, 5'd27, 22'd6 }
42          assign memoria[41]={5'd18, 5'd27, 5'd6, 17'd0 }
43          assign memoria[42]={5'd17, 5'd27, 22'd16 }
44          assign memoria[43]={5'd16, 5'd7, 5'd27, 17'd0 }
45          assign memoria[44]={5'd17, 5'd27, 22'd13 }
46          assign memoria[45]={5'd18, 5'd27, 5'd7, 17'd0 }
47          assign memoria[46]={5'd23, 27'd0 }
48          assign memoria[47]={5'd17, 5'd27, 22'd16 }
49          assign memoria[48]={5'd16, 5'd8, 5'd27, 17'd0 }
50          assign memoria[49]={5'd17, 5'd9, 22'd0 }
51          assign memoria[50]={5'd1, 5'd10, 5'd8, 5'd9, 12'd0 }
52          assign memoria[51]={5'd17, 5'd27, 22'd16 }
53          assign memoria[52]={5'd18, 5'd27, 5'd10, 17'd0 }
54          assign memoria[53]={5'd13, 27'd19 }
55          assign memoria[54]={5'd23, 27'd0 }
56          assign memoria[55]={5'd17, 5'd27, 22'd13 }
57          assign memoria[56]={5'd16, 5'd11, 5'd27, 17'd0 }
58          assign memoria[57]={5'd19, 5'd28, 5'd11, 17'd0 }
59          assign memoria[58]={5'd14, 5'd31, 22'd0 }
60          assign memoria[59]={5'd14, 5'd31, 22'd0 }
61          assign memoria[60]={5'd23, 27'd0 }
62          assign memoria[61]={5'd17, 5'd27, 22'd10 }
63          assign memoria[62]={5'd16, 5'd12, 5'd27, 17'd0 }
64          assign memoria[63]={5'd17, 5'd27, 22'd16 }
65          assign memoria[64]={5'd18, 5'd27, 5'd12, 17'd0 }
66          assign memoria[65]={5'd23, 27'd0 }
67          assign memoria[66]={5'd17, 5'd27, 22'd11 }
68          assign memoria[67]={5'd16, 5'd13, 5'd27, 17'd0 }
69          assign memoria[68]={5'd17, 5'd14, 22'd0 }
70          assign memoria[69]={5'd17, 5'd14, 22'd0 }
71          assign memoria[70]={5'd17, 5'd27, 22'd16 }
72          assign memoria[71]={5'd16, 5'd16, 5'd27, 17'd0 }
73          assign memoria[72]={5'd9, 5'd17, 5'd16, 5'd15, 12'd0 }
74          assign memoria[73]={5'd11, 5'd17, 5'd0, 17'd7 }
75          assign memoria[74]={5'd17, 5'd27, 22'd9 }
76          assign memoria[75]={5'd16, 5'd18, 5'd27, 17'd0 }
77          assign memoria[76]={5'd19, 5'd22, 5'd18, 17'd0 }
78          assign memoria[77]={5'd17, 5'd27, 22'd16 }
79          assign memoria[78]={5'd16, 5'd19, 5'd27, 17'd0 }
80          assign memoria[79]={5'd19, 5'd23, 5'd19, 17'd0 }
81          assign memoria[80]={5'd17, 5'd27, 22'd11 }
82          assign memoria[81]={5'd16, 5'd20, 5'd27, 17'd0 }
83          assign memoria[82]={5'd19, 5'd24, 5'd20, 17'd0 }
84          assign memoria[83]={5'd15, 27'd0 }
85          assign memoria[84]={5'd19, 5'd21, 5'd28, 17'd0 }
86          assign memoria[85]={5'd17, 5'd27, 22'd13 }
87          assign memoria[86]={5'd18, 5'd27, 5'd21, 17'd0 }
88          assign memoria[87]={5'd17, 5'd27, 22'd13 }
89          assign memoria[88]={5'd16, 5'd22, 5'd27, 17'd0 }
```

```
90          assign memoria[89]={5'd17, 5'd27, 22'd9 }
91          assign memoria[90]={5'd1, 5'd27, 5'd27, 5'd22, 12'd0 }
92          assign memoria[91]={5'd16, 5'd23, 5'd27, 17'd0 }
93          assign memoria[92]={5'd17, 5'd27, 22'd14 }
94          assign memoria[93]={5'd18, 5'd27, 5'd23, 17'd0 }
95          assign memoria[94]={5'd17, 5'd27, 22'd16 }
96          assign memoria[95]={5'd16, 5'd24, 5'd27, 17'd0 }
97          assign memoria[96]={5'd17, 5'd27, 22'd9 }
98          assign memoria[97]={5'd1, 5'd27, 5'd27, 5'd24, 12'd0 }
99          assign memoria[98]={5'd16, 5'd25, 5'd27, 17'd0 }
100         assign memoria[99]={5'd17, 5'd27, 22'd13 }
101         assign memoria[100]={5'd16, 5'd26, 5'd27, 17'd0 }
102         assign memoria[101]={5'd17, 5'd27, 22'd9 }
103         assign memoria[102]={5'd1, 5'd27, 5'd27, 5'd26, 12'd0 }
104         assign memoria[103]={5'd18, 5'd27, 5'd25, 17'd0 }
105         assign memoria[104]={5'd17, 5'd27, 22'd14 }
106         assign memoria[105]={5'd16, 5'd27, 5'd27, 17'd0 }
107         assign memoria[106]={5'd17, 5'd27, 22'd16 }
108         assign memoria[107]={5'd16, 5'd28, 5'd27, 17'd0 }
109         assign memoria[108]={5'd17, 5'd27, 22'd9 }
110         assign memoria[109]={5'd1, 5'd27, 5'd27, 5'd28, 12'd0 }
111         assign memoria[110]={5'd18, 5'd27, 5'd27, 17'd0 }
112         assign memoria[111]={5'd17, 5'd27, 22'd16 }
113         assign memoria[112]={5'd16, 5'd29, 5'd27, 17'd0 }
114         assign memoria[113]={5'd17, 5'd30, 22'd0 }
115         assign memoria[114]={5'd1, 5'd31, 5'd29, 5'd30, 12'd0 }
116         assign memoria[115]={5'd17, 5'd27, 22'd16 }
117         assign memoria[116]={5'd18, 5'd27, 5'd31, 17'd0 }
118         assign memoria[117]={5'd13, 27'd65 }
119         assign memoria[118]={5'd23, 27'd0 }
120         assign memoria[119]={5'd14, 5'd31, 22'd0 }
121         assign memoria[120]={5'd23, 27'd0 }
122         assign memoria[121]={5'd17, 5'd32, 22'd0 }
123         assign memoria[122]={5'd17, 5'd27, 22'd16 }
124         assign memoria[123]={5'd18, 5'd27, 5'd32, 17'd0 }
125         assign memoria[124]={5'd23, 27'd0 }
126         assign memoria[125]={5'd17, 5'd27, 22'd16 }
127         assign memoria[126]={5'd16, 5'd33, 5'd27, 17'd0 }
128         assign memoria[127]={5'd17, 5'd34, 22'd0 }
129         assign memoria[128]={5'd9, 5'd35, 5'd33, 5'd34, 12'd0 }
130         assign memoria[129]={5'd11, 5'd35, 5'd0, 17'd10 }
131         assign memoria[130]={5'd20, 27'd0 }
132         assign memoria[131]={5'd19, 5'd36, 5'd30, 17'd0 }
133         assign memoria[132]={5'd17, 5'd27, 22'd16 }
134         assign memoria[133]={5'd16, 5'd37, 5'd27, 17'd0 }
135         assign memoria[134]={5'd17, 5'd27, 22'd0 }
136         assign memoria[135]={5'd1, 5'd27, 5'd27, 5'd37, 12'd0 }
137         assign memoria[136]={5'd18, 5'd27, 5'd36, 17'd0 }
138         assign memoria[137]={5'd17, 5'd27, 22'd16 }
139         assign memoria[138]={5'd16, 5'd38, 5'd27, 17'd0 }
140         assign memoria[139]={5'd17, 5'd39, 22'd0 }
141         assign memoria[140]={5'd1, 5'd40, 5'd38, 5'd39, 12'd0 }
142         assign memoria[141]={5'd17, 5'd27, 22'd16 }
143         assign memoria[142]={5'd18, 5'd27, 5'd40, 17'd0 }
144         assign memoria[143]={5'd13, 27'd124 }
145         assign memoria[144]={5'd23, 27'd0 }
146         assign memoria[145]={5'd17, 5'd27, 22'd0 }
147         assign memoria[146]={5'd16, 5'd41, 5'd27, 17'd0 }
148         assign memoria[147]={5'd19, 5'd25, 5'd41, 17'd0 }
149         assign memoria[148]={5'd17, 5'd42, 22'd0 }
```

```
150         assign memoria[149]={5'd19, 5'd26, 5'd42, 17'd0 }
151         assign memoria[150]={5'd17, 5'd43, 22'd0 }
152         assign memoria[151]={5'd19, 5'd27, 5'd43, 17'd0 }
153         assign memoria[152]={5'd15, 27'd0 }
154         assign memoria[153]={5'd17, 5'd44, 22'd0 }
155         assign memoria[154]={5'd17, 5'd27, 22'd16 }
156         assign memoria[155]={5'd18, 5'd27, 5'd44, 17'd0 }
157         assign memoria[156]={5'd23, 27'd0 }
158         assign memoria[157]={5'd17, 5'd27, 22'd16 }
159         assign memoria[158]={5'd16, 5'd45, 5'd27, 17'd0 }
160         assign memoria[159]={5'd17, 5'd46, 22'd0 }
161         assign memoria[160]={5'd9, 5'd47, 5'd45, 5'd46, 12'd0 }
162         assign memoria[161]={5'd11, 5'd47, 5'd0, 17'd12 }
163         assign memoria[162]={5'd17, 5'd27, 22'd16 }
164         assign memoria[163]={5'd16, 5'd48, 5'd27, 17'd0 }
165         assign memoria[164]={5'd17, 5'd27, 22'd0 }
166         assign memoria[165]={5'd1, 5'd27, 5'd27, 5'd48, 12'd0 }
167         assign memoria[166]={5'd16, 5'd49, 5'd27, 17'd0 }
168         assign memoria[167]={5'd19, 5'd28, 5'd49, 17'd0 }
169         assign memoria[168]={5'd24, 5'd22, 22'd0 }
170         assign memoria[169]={5'd17, 5'd27, 22'd16 }
171         assign memoria[170]={5'd16, 5'd50, 5'd27, 17'd0 }
172         assign memoria[171]={5'd17, 5'd51, 22'd0 }
173         assign memoria[172]={5'd1, 5'd52, 5'd50, 5'd51, 12'd0 }
174         assign memoria[173]={5'd17, 5'd27, 22'd16 }
175         assign memoria[174]={5'd18, 5'd27, 5'd52, 17'd0 }
176         assign memoria[175]={5'd13, 27'd156 }
177         assign memoria[176]={5'd23, 27'd0 }
178         assign memoria[177]={5'd21, 27'd0 }
```

Listing 4.8 – Código Sort

4.2.4 Relação Entre os Códigos Gerados

Os códigos estão todos interligados desde a fase de análise, pois para gerar o código executável é necessário a lista de instruções em assembly, que no que lhe concerne precisa da lista de quadruplas gerada no código intermediário, que este precisa da árvore sintática e a tabela de símbolos, que estes precisam da lista de tokens. Então é possível verificar que todos os códigos estão interligados. Para visualizar melhor isso no apêndice C onde gera o código assembly, na linha 128 executa o comando "*quadrupla *aux = lista_quad->primeiro;*", onde demonstra que para gerar a lista de instruções é preciso percorrer a lista de quadruplas e no final de gerar a lista de instruções na linha 520 executa "*GeraExec(listaAsmb->primeiro);*", onde demonstra que para gerar o código executável precisa da lista de instruções, ou seja, todos os códigos possuem relação entre eles.

4.3 Exemplo GDC

O código gdc calcula o maior divisor comum entre dois números através da função gdc().

```
1 int gdc (int u, int v)
```

```
2  {
3     if (v == 0) return u;
4     else return gdc(v,u-u/v*v);
5
6  }
7
8  void main(void)
9  {
10    int x;
11    int y;
12    x = input();
13    y = input();
14    output(gdc(x,y));
15 }
```

Listing 4.9 – Código GCD

4.3.1 Código Intemerdiário

```
1  ( nop    ,           ,           ,           )
2  ( fun    , gdc       ,           ,           )
3  ( arg    , u         ,           , gdc       )
4  ( arg    , v         ,           , gdc       )
5  ( load   , t_1       , v         ,           )
6  ( imed   , t_2       ,           , 0         )
7  ( ==     , t_3       , t_1       , t_2       )
8  ( beq    , t_3       , t_0       , L_2       )
9  ( load   , t_1       , u         ,           )
10 ( ret    , t_1       ,           ,           )
11 ( jump   ,           ,           , L_3       )
12 ( label  ,           ,           , L_2       )
13 ( load   , t_2       , v         ,           )
14 ( setArg ,           ,           , t_2       )
15 ( load   , t_3       , u         ,           )
16 ( load   , t_1       , v         ,           )
17 ( /      , t_2       , t_3       , t_1       )
18 ( load   , t_3       , v         ,           )
19 ( *      , t_4       , t_2       , t_3       )
20 ( load   , t_5       , u         ,           )
21 ( -      , t_6       , t_5       , t_4       )
22 ( setArg ,           ,           , t_6       )
23 ( call   , t_7       ,           , gdc       )
24 ( ret    , t_7       ,           ,           )
25 ( label  ,           ,           , L_3       )
26 ( end    , gdc       ,           ,           )
27 ( fun    , main      ,           ,           )
28 ( alloc  , x         , 1         , main      )
29 ( alloc  , y         , 1         , main      )
30 ( call   , t_8       ,           , input     )
31 ( store  , x         ,           , t_8       )
32 ( call   , t_9       ,           , input     )
33 ( store  , y         ,           , t_9       )
34 ( load   , t_10      , x         ,           )
35 ( setArg ,           ,           , t_10      )
36 ( load   , t_11      , y         ,           )
37 ( setArg ,           ,           , t_11      )
38 ( call   , t_12      ,           , gdc       )
39 ( setArg ,           ,           , t_12      )
```

```
( call  ,          ,          , output     )
( end   , main     ,          ,            )
( halt  ,          ,          ,            )
```

Listing 4.10 – Código Fatorial - Quadruplas do Código Intermediário

4.3.2 Código Assembly

```
nop
gdc:

        li $r27, 2
        lw $r1, $r27
        li $r2, 0
        set $r3, $r1, $r2
        beq $r3, $r0, L2
        li $r27, 1
        lw $r1, $r27
        move $r28, $r1
        jr $r31
        jump L3:
        L2:
            li $r27, 2
            lw $r2, $r27
            move $r22, $r2
            li $r27, 1
            lw $r3, $r27
            li $r27, 2
            lw $r1, $r27
            lw $r2, $r3, $r1
            li $r27, 2
            lw $r3, $r27
            mult $r4, $r2, $r3
            li $r27, 1
            lw $r5, $r27
            lw $r6, $r5, $r4
            move $r23, $r6
            jal gdc
            move $r7, $r28
            move $r28, $r7
            jr $r31
        L3:
            jr $r31
main:

        in
        move $r8, $r30
        li $r27, 4
        sw $r27, $r8
        in
        move $r9, $r30
        li $r27, 5
        sw $r27, $r9
        li $r27, 4
        lw $r10, $r27
        move $r24, $r10
        li $r27, 5
        lw $r11, $r27
```

```
51        move $r25, $r11
52        jal gdc
53        move $r12, $r28
54        move $r26, $r12
55        out $r22
56        halt
```

Listing 4.11 – Código Fatorial - Assembly

4.3.3 Código Executável

```
1         assign memoria[0]={5'd23, 27'd0 }
2         assign memoria[1]={5'd23, 27'd0 }
3         assign memoria[2]={5'd17, 5'd27, 22'd2 }
4         assign memoria[3]={5'd16, 5'd1, 5'd27, 17'd0 }
5         assign memoria[4]={5'd17, 5'd2, 22'd0 }
6         assign memoria[5]={5'd5, 5'd3, 5'd1, 5'd2, 12'd0 }
7         assign memoria[6]={5'd11, 5'd3, 5'd0, 17'd2 }
8         assign memoria[7]={5'd17, 5'd27, 22'd1 }
9         assign memoria[8]={5'd16, 5'd1, 5'd27, 17'd0 }
10        assign memoria[9]={5'd19, 5'd28, 5'd1, 17'd0 }
11        assign memoria[10]={5'd14, 5'd31, 22'd0 }
12        assign memoria[11]={5'd13, 27'd0 }
13        assign memoria[12]={5'd23, 27'd0 }
14        assign memoria[13]={5'd17, 5'd27, 22'd2 }
15        assign memoria[14]={5'd16, 5'd2, 5'd27, 17'd0 }
16        assign memoria[15]={5'd19, 5'd22, 5'd2, 17'd0 }
17        assign memoria[16]={5'd17, 5'd27, 22'd1 }
18        assign memoria[17]={5'd16, 5'd3, 5'd27, 17'd0 }
19        assign memoria[18]={5'd17, 5'd27, 22'd2 }
20        assign memoria[19]={5'd16, 5'd1, 5'd27, 17'd0 }
21        assign memoria[20]={5'd16, 5'd1, 5'd27, 17'd0 }
22        assign memoria[21]={5'd17, 5'd27, 22'd2 }
23        assign memoria[22]={5'd16, 5'd3, 5'd27, 17'd0 }
24        assign memoria[23]={5'd3, 5'd4, 5'd2, 5'd3, 12'd0 }
25        assign memoria[24]={5'd17, 5'd27, 22'd1 }
26        assign memoria[25]={5'd16, 5'd5, 5'd27, 17'd0 }
27        assign memoria[26]={5'd16, 5'd5, 5'd27, 17'd0 }
28        assign memoria[27]={5'd19, 5'd23, 5'd6, 17'd0 }
29        assign memoria[28]={5'd15, 27'd0 }
30        assign memoria[29]={5'd19, 5'd7, 5'd28, 17'd0 }
31        assign memoria[30]={5'd19, 5'd28, 5'd7, 17'd0 }
32        assign memoria[31]={5'd14, 5'd31, 22'd0 }
33        assign memoria[32]={5'd23, 27'd0 }
34        assign memoria[33]={5'd14, 5'd31, 22'd0 }
35        assign memoria[34]={5'd23, 27'd0 }
36        assign memoria[35]={5'd20, 27'd0 }
37        assign memoria[36]={5'd19, 5'd8, 5'd30, 17'd0 }
38        assign memoria[37]={5'd17, 5'd27, 22'd4 }
39        assign memoria[38]={5'd18, 5'd27, 5'd8, 17'd0 }
40        assign memoria[39]={5'd20, 27'd0 }
41        assign memoria[40]={5'd19, 5'd9, 5'd30, 17'd0 }
42        assign memoria[41]={5'd17, 5'd27, 22'd5 }
43        assign memoria[42]={5'd18, 5'd27, 5'd9, 17'd0 }
44        assign memoria[43]={5'd17, 5'd27, 22'd4 }
45        assign memoria[44]={5'd16, 5'd10, 5'd27, 17'd0 }
46        assign memoria[45]={5'd19, 5'd24, 5'd10, 17'd0 }
47        assign memoria[46]={5'd17, 5'd27, 22'd5 }
```

```
48          assign memoria[47]={5'd16, 5'd11, 5'd27, 17'd0 }
49          assign memoria[48]={5'd19, 5'd25, 5'd11, 17'd0 }
50          assign memoria[49]={5'd15, 27'd0 }
51          assign memoria[50]={5'd19, 5'd12, 5'd28, 17'd0 }
52          assign memoria[51]={5'd19, 5'd26, 5'd12, 17'd0 }
53          assign memoria[52]={5'd24, 5'd22, 22'd0 }
54          assign memoria[53]={5'd21, 27'd0 }
```

Listing 4.12 – Código Fatorial - Executável

5 Considerações Finais

O objetivo deste projeto era implementar um compilador para linguagem c- utilizando os conceitos aprendidos nas disciplinas durante a graduação e este objetivo foi cumprido com sucesso, mesmo que tenha sido um compilador não tão robusto e complexo.

Durante o desenvolvimento deste projeto foi encontrado diversas dificuldades, como já era esperado, pois, este projeto é de complexidade elevada. A primeira dificuldade foi a falta de troca de informações entre aluno-professor e aluno-aluno, pois não está havendo aulas presenciais para tirar dúvidas e ter debates entre alunos, então a única solução foi manter o contato online e buscar informações em outras fontes, como livros, internet e conteúdos aprendidos anteriormente. A segunda dificuldade foi ligar as especificações do processador já feito com o compilador, então a cada etapa desenvolvida no compilador tinha que pensar se estaria fazendo sentido com a arquitetura do processador desenvolvido. Terceira e maior dificuldade foi o gerenciamento de memória usando o método de pilha, devido à alta complexidade do gerenciamento e o tempo para a entrega do dispositivo acarretaram não implementação de funcionamento de algoritmos recursivos e de passagem de vetores por parâmetro de funções.

O projeto todo foi muito enriquecedor para o aprendizado, pois durante a graduação são compilados diversos códigos, mas não é mostrado como é feito de fato um compilador. Doravante terá um pensamento mais técnico na hora de implementar códigos, pois saberá o que está acontecendo por de trás do processo.

Vale destacar que o compilador tem melhorias a serem feitas e que para verificar totalmente o funcionamento do compilador junto ao processador é necessário fazer o teste utilizando uma placa FPGA.

Referências

1 LOUDEN, K. C.; SILVA, F. S. C. *Compiladores-Princípios e Práticas*. [S.l.]: Cengage Learning Editores, 2004. Citado na página 7.

2 SANTOS, T. L. P. R. *Compiladores: da teoria á prática*. [S.l.]: Livros Técnicos e Científicos Editora Ltda., 2018. Citado na página 7.

3 STALLINGS, W. *Computer organization and architecture*. 8th edition. ed. São Paulo: Marina S. Lupinetti, 2010. Citado na página 7.

4 CAPPABIANCO, F. A. M. *Codificadores e Multiplexadores*. UNIFESP, Universidade Federal de São Paulo: [s.n.], 2018. Citado na página 11.

5 PATTERSON, D. A.; HENNESY, J. L. *Computer Organization and Design*. 5th edition. ed. Waltham/MA, EUA: Morgan Kaufmann, 2007. Citado 2 vezes nas páginas 12 e 13.

A Código analyze.c

```
1  /****************************************************/
2  /****************************************************/
3  /*       COMPILADOR PARA LINGUAGEM C-                */
4  /*                                                    */
5  /*                                                    */
6  /*       Daiana Santos   RA: 120.357                 */
7  /****************************************************/
8
9  #include "globals.h"
10 #include "symtab.h"
11 #include "analyze.h"
12
13 static int location = 0;
14 char* escopo = "global";
15
16 void atualizaEscopo(TreeNode * t)
17 {
18   if (t->child[0] != NULL && t->child[0]->kind.exp == functionK) escopo = t->child[0]->
       attr.name;
19 }
20
21 static void traverse( TreeNode * t,
22                void (* preProc) (TreeNode *),
23                void (* postProc) (TreeNode *) )
24 {
25
26         if (t != NULL)
27         {
28                  atualizaEscopo(t);
29     preProc(t);
30         {
31                         int i;
32                  for (i=0; i < MAXCHILDREN; i++)
33                         traverse(t->child[i],preProc,postProc);
34         }
35         if(t->child[0]!= NULL && t->child[0]->kind.exp == functionK) escopo = "global";
36       postProc(t);
37         traverse(t->sibling,preProc,postProc);
38         }
39 }
40
41 static void nullProc(TreeNode * t)
42 {
43         if (t==NULL)
44                 return;
45         else
46                 return;
47 }
48
49 static void insertNode( TreeNode * t)
50 {
51
52         switch (t->nodekind)
53   {
```

```
54                 case statementK:
55       if(t->kind.stmt == assignK)
56       {
57           if (st_lookup(t->child[0]->attr.name) == -1){
58             fprintf(listing,"Erro: A variavel %s n o foi declarada. [%d]\n", t->child
                   [0]->attr.name, t->lineno);
59             Error = TRUE;
60           }
61           else
62             st_insert(t->child[0]->attr.name,t->lineno,0,escopo,INTTYPE,VAR);
63           t->child[0]->add = 1;
64       }
65       break;
66
67       case expressionK:
68         switch (t->kind.exp)
69         {
70           case typeK:
71
72             if(t->child[0] != NULL){
73
74               switch (t->child[0]->kind.exp)
75               {
76                 case variableK:
77                   if (st_lookup(t->attr.name) == -1){
78                 /* n o encontrado na tabela, inserir*/
79                   st_insert(t->child[0]->attr.name,t->lineno,location++, escopo,INTTYPE,
                       VAR);
80
81                 }
82                 else
83                 /* encontrado na tabela, verificar escopo */
84                   st_insert(t->child[0]->attr.name,t->lineno,0, escopo,INTTYPE, VAR);
85                 break;
86
87                 case functionK:
88                   if (st_lookup(t->attr.name) == -1){
89                 /* n o encontrado na tabela, inserir*/
90                   st_insert(t->child[0]->attr.name,t->child[0]->lineno,location++, "
                       global",t->child[0]->type,FUN);
91               }
92                 else
93                 /* encontrado na tabela, verificar escopo */
94                   fprintf(listing,"Erro: Multiplas declara es da fun o %s. [%d]\n",
                        t->child[0]->attr.name, t->lineno);
95
96               break;
97               default:
98               break;
99               }
100          }
101          break;
102
103          case paramK:
104          st_insert(t->attr.name,t->lineno,location++, escopo,INTTYPE, VAR);
105          break;
106        case idK:
107          if(t->add != 1){
108            if (st_lookup(t->attr.name) == -1){
109              fprintf(listing,"Erro: A variavel %s n o foi declarada. [%d]\n", t->attr.
```

```
                    name, t->lineno);
110               Error = TRUE;
111             }
112             else {
113               st_insert(t->attr.name,t->lineno,0, escopo,INTTYPE,FUN);
114             }
115           }
116           break;
117         case activationK:
118           if (st_lookup(t->attr.name) == -1 && strcmp(t->attr.name, "output")!=0 &&
                  strcmp(t->attr.name,"input")!=0){
119             fprintf(listing,"Erro: A fun  o %s n o foi declarada. [%d]\n", t->attr.
                    name, t->lineno);
120             Error = TRUE;
121           }
122           else {
123             st_insert(t->attr.name,t->lineno,0, escopo,0,FUN);
124           }
125           break;
126         default:
127           break;
128       }
129       break;
130       default:
131       break;
132   }
133 }
134
135 void buildSymtab(TreeNode * syntaxTree)
136 {
137
138   traverse(syntaxTree,insertNode,nullProc);
139   busca_main();
140   typeCheck(syntaxTree);
141
142   if (TraceAnalyze && !Error)
143   {
144     fprintf(listing,"\nTabela de simbolos:\n\n");
145     printSymTab(listing);
146   }
147 }
148
149 static void typeError(TreeNode * t, char * message)
150 { fprintf(listing,"Erro de tipo na linha %d: %s\n",t->lineno,message);
151   Error = TRUE;
152 }
153
154 void checkNode(TreeNode * t)
155 {
156
157   switch (t->nodekind)
158   { case expressionK:
159       switch (t->kind.exp)
160       { case operationK:
161           if (((t->child[0]->kind.exp == activationK) &&( getFunType(t->child[0]->attr.
                  name)) == VOIDTYPE) ||
162               ((t->child[1]->kind.exp == activationK) && (getFunType(t->child[1]->attr.
                    name) == VOIDTYPE)))
163                 typeError(t->child[0],"Ativa  o de fun  o do tipo void na express o"
                      );
```

```
164           break;
165
166         default:
167           break;
168       }
169       break;
170     case statementK:
171       switch (t->kind.stmt)
172       {
173         case assignK:
174
175           if (t->child[1]->kind.exp == activationK && getFunType(t->child[1]->attr.name)
                == VOIDTYPE)
176             typeError(t->child[0],"Fun   o com retorno void n o pode ser atribuido a
                  uma variavel");
177           break;
178
179         default:
180           break;
181       }
182       break;
183     default:
184       break;
185
186   }
187 }
188
189
190 void typeCheck(TreeNode * syntaxTree)
191 {
192     traverse(syntaxTree,nullProc,checkNode);
193 }
```

B Código analyze.h

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos   RA: 120.357                 */
6  /****************************************************/
7  #ifndef _ANALYZE_H_
8  #define _ANALYZE_H_
9
10 void buildSymtab(TreeNode *);
11
12 void typeCheck(TreeNode *);
13
14 #endif
```

C Código assmb.c

```
/****************************************************/
/*       COMPILADOR PARA LINGUAGEM C-                */
/*                                                    */
/*                                                    */
/*       Daiana Santos    RA: 120.357                */
/****************************************************/

#include "globals.h"
#include "symtab.h"
#include "analyze.h"
#include "cgen.h"
#include "assmb.h"
#include "exec.h"
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

void insere_ASSMB(OPCODE opcode) {

    INSTRU *novo;
    novo=(INSTRU*)malloc(sizeof(INSTRU));

    novo->proximo= NULL;
    novo->opcode = opcode;

    if(listaAsmb->tamanho==0) {

        listaAsmb->primeiro=novo;

    } else {
        listaAsmb->ultimo->proximo=novo;
    }

    listaAsmb->ultimo=novo;
    listaAsmb->tamanho++;

}

void imprime_ASSMB(INSTRU *Noaux){

    char *aux;
    bool label_flag = true;
    bool fun_flag = true;

    printf("\n\n\n             CODIGO ASSEMBLY GERADO:\n\n");

    while(Noaux != NULL) {

        switch (Noaux->opcode){
            case NOP:       aux = "nop";break;
            case HALT:      aux = "halt";break;
            case SW:        aux = "sw";break;
            case LW:        aux = "lw";break;
            case LABEL:
```

```
55                  aux = "";
56                  if(Noaux->op1.type == funck){
57                      fun_flag = true;
58                  }
59                  label_flag = true;
60
61                  break;
62              case BEQ:       aux = "beq";break;
63              case SET:       aux = "set";break;
64              case SDT:       aux = "sdt";break;
65              case SLT:       aux = "slt";break;
66              case SLE:       aux = "sle";break;
67              case SGT:       aux = "sgt";break;
68              case SGE:       aux = "sge";break;
69              case BNE:       aux = "bne";break;
70              case JUMP:      aux = "jump ";break;
71              case LI:        aux = "li";break;
72              case MOVE:      aux = "move";break;
73              case ADD:       aux = "add";break;
74              case SUB:       aux = "sub";break;
75              case MULT:      aux = "mult";break;
76              case DIV:       aux = "div";break;
77              case IN:        aux = "in";break;
78              case JAL:       aux = "jal";break;
79              case JR:        aux = "jr";break;
80              case OUT:       aux = "out";break;
81              case WAIT:      aux = "wait";break;
82
83          }
84
85          if(!fun_flag){
86              printf("        ");
87          }
88
89          if(!label_flag){
90              printf("    ");
91          }
92
93          printf("%s", aux);
94
95          switch (Noaux->op1.type){
96              case regTemp:printf(" $r%d", Noaux->op1.value);break;
97              case Const: printf(" %d",Noaux->op1.value );break;
98              case labelk:
99                  printf("L%d:",Noaux->op1.value );
100                 label_flag = false;
101                 break;
102             case funck:
103                 printf("%s:\n",Noaux->op1.name);
104                 fun_flag = false;
105                 break;
106         }
107
108         switch (Noaux->op2.type){
109
110             case regTemp:printf(", $r%d", Noaux->op2.value);break;
111             case Const: printf(", %d",Noaux->op2.value );break;
112             case funck: printf(" %s",Noaux->op2.name);break;
113         }
114
```

```
115         switch (Noaux->op3.type){
116             case regTemp:printf(", $r%d", Noaux->op3.value);break;
117             case Const: printf(", %d",Noaux->op3.value );break;
118             case labelk: printf(", L%d",Noaux->op3.value );break;
119         }
120 
121         printf("\n");
122         Noaux = Noaux->proximo;
123     }
124 }
125 
126 void percorre_lista (void){
127 
128         quadrupla *aux = lista_quad->primeiro;
129 
130     int argmt = 21;
131     int i;
132 
133 
134         while(aux != NULL){
135 
136                 switch (aux->instrucao){
137                 case nop:
138                 insere_ASSMB(NOP);
139                         break;
140 
141             case halt:
142                 insere_ASSMB(HALT);
143                 break;
144 
145             case store:
146 
147                 insere_ASSMB(LI);
148                 listaAsmb->ultimo->op1.type = regTemp;
149                 listaAsmb->ultimo->op1.value = 27;
150 
151                 listaAsmb->ultimo->op2.type = Const;
152                 listaAsmb->ultimo->op2.value = aux->op1.endereco;
153 
154                 if(aux->op2.type == regTemp){
155 
156                     insere_ASSMB(ADD);
157                     listaAsmb->ultimo->op1.type = regTemp;
158                     listaAsmb->ultimo->op1.value = 27;
159 
160                     listaAsmb->ultimo->op2.type = regTemp;
161                     listaAsmb->ultimo->op2.value = 27;
162 
163                     listaAsmb->ultimo->op3.type = regTemp;
164                     listaAsmb->ultimo->op3.value = aux->op2.value;
165 
166                 }
167 
168                 insere_ASSMB(SW);
169                 listaAsmb->ultimo->op1.type = regTemp;
170                 listaAsmb->ultimo->op1.value = 27;
171 
172                 listaAsmb->ultimo->op2.type = regTemp;
173                 listaAsmb->ultimo->op2.value = aux->op3.value;
174 
```

```
175
176             break;
177
178         case load:
179             insere_ASSMB(LI);
180             listaAsmb->ultimo->op1.type = regTemp;
181             listaAsmb->ultimo->op1.value = 27;
182
183             listaAsmb->ultimo->op2.type = Const;
184             listaAsmb->ultimo->op2.value = aux->op2.endereco;
185
186             if(aux->op3.type == regTemp){
187
188                 insere_ASSMB(ADD);
189                 listaAsmb->ultimo->op1.type = regTemp;
190                 listaAsmb->ultimo->op1.value = 27;
191
192                 listaAsmb->ultimo->op2.type = regTemp;
193                 listaAsmb->ultimo->op2.value = 27;
194
195                 listaAsmb->ultimo->op3.type = regTemp;
196                 listaAsmb->ultimo->op3.value = aux->op3.value;
197
198             }
199
200             insere_ASSMB(LW);
201             listaAsmb->ultimo->op1.type = regTemp;
202             listaAsmb->ultimo->op1.value = aux->op1.value;
203
204             listaAsmb->ultimo->op2.type = regTemp;
205             listaAsmb->ultimo->op2.value = 27;
206             break;
207
208
209         case fun:
210             insere_ASSMB(LABEL);
211             listaAsmb->ultimo->op1.type = funck;
212             listaAsmb->ultimo->op1.name = aux->op1.name;
213             listaAsmb->ultimo->op1.value = aux->op1.value;
214             break;
215
216         case arg:
217             i = 22;
218
219             while(i<=argmt){
220
221                 insere_ASSMB(SW);
222                 listaAsmb->ultimo->op1.type = regTemp;
223                 listaAsmb->ultimo->op1.value = i;
224
225                 listaAsmb->ultimo->op2.type = regTemp;
226                 listaAsmb->ultimo->op2.value = aux->op3.value;
227
228                 i++;
229             }
230
231             argmt = 21;
232
233             break;
234
```

```
235             case setArg:
236 
237                 argmt++;
238 
239                 insere_ASSMB(MOVE);
240                 listaAsmb->ultimo->op1.type = regTemp;
241                 listaAsmb->ultimo->op1.value = argmt;
242 
243                 listaAsmb->ultimo->op2.type = regTemp;
244                 listaAsmb->ultimo->op2.value = aux->op3.value;
245 
246                 break;
247 
248             case call:
249                 if(strcmp(aux->op3.name, "output") == 0){
250 
251 
252                     insere_ASSMB(OUT);
253                     listaAsmb->ultimo->op1.type = regTemp;
254                     listaAsmb->ultimo->op1.value = 22;
255 
256 
257                 }else if(strcmp(aux->op3.name, "input") == 0){
258 
259                     insere_ASSMB(IN);
260 
261 
262                     insere_ASSMB(MOVE);
263                     listaAsmb->ultimo->op1.type = regTemp;
264                     listaAsmb->ultimo->op1.value = aux->op1.value;
265                     listaAsmb->ultimo->op2.type = regTemp;
266                     listaAsmb->ultimo->op2.value = 30;
267                 }
268                 else{
269                     insere_ASSMB(JAL);
270                     listaAsmb->ultimo->op2.type = funck;
271                     listaAsmb->ultimo->op2.name = aux->op3.name;
272                     listaAsmb->ultimo->op1.value = aux->op1.value;
273                     if(aux->op1.type == regTemp){
274                         insere_ASSMB(MOVE);
275                         listaAsmb->ultimo->op1.type = regTemp;
276                         listaAsmb->ultimo->op1.value = aux->op1.value;
277                         listaAsmb->ultimo->op2.type = regTemp;
278                         listaAsmb->ultimo->op2.value = 28;
279                     }
280                 }
281                 break;
282 
283             case end:
284 
285                 if(strcmp(aux->op1.name, "main") != 0){
286                     insere_ASSMB(JR);
287                     listaAsmb->ultimo->op1.type = regTemp;
288                     listaAsmb->ultimo->op1.value = 31;
289 
290                 }
291 
292                 break;
293 
294             case alloc:
```

```
295
296                 break;
297
298             case ret:
299
300                 insere_ASSMB(MOVE);
301                 listaAsmb->ultimo->op1.type = regTemp;
302                 listaAsmb->ultimo->op1.value = 28;
303
304                 listaAsmb->ultimo->op2.type = regTemp;
305                 listaAsmb->ultimo->op2.value = aux->op1.value;
306
307                 insere_ASSMB(JR);
308                 listaAsmb->ultimo->op1.type = regTemp;
309                 listaAsmb->ultimo->op1.value = 31;
310
311                 break;
312
313             case label:
314
315                 insere_ASSMB(LABEL);
316                 listaAsmb->ultimo->op1.type = labelk;
317                 listaAsmb->ultimo->op1.value = aux->op3.value;
318                 break;
319
320             case IGLIGL:
321                 insere_ASSMB(SET);
322
323                 listaAsmb->ultimo->op1.type = regTemp;
324                 listaAsmb->ultimo->op1.value = aux->op1.value;
325
326                 listaAsmb->ultimo->op2.type = regTemp;
327                 listaAsmb->ultimo->op2.value = aux->op2.value;
328
329                 listaAsmb->ultimo->op3.type = regTemp;
330                 listaAsmb->ultimo->op3.value = aux->op3.value;
331                 break;
332
333
334             case DIF:
335                 insere_ASSMB(SDT);
336
337                 listaAsmb->ultimo->op1.type = regTemp;
338                 listaAsmb->ultimo->op1.value = aux->op1.value;
339
340                 listaAsmb->ultimo->op2.type = regTemp;
341                 listaAsmb->ultimo->op2.value = aux->op2.value;
342
343                 listaAsmb->ultimo->op3.type = regTemp;
344                 listaAsmb->ultimo->op3.value = aux->op3.value;
345                 break;
346
347             case MEN:
348                 insere_ASSMB(SLT);
349
350                 listaAsmb->ultimo->op1.type = regTemp;
351                 listaAsmb->ultimo->op1.value = aux->op1.value;
352
353                 listaAsmb->ultimo->op2.type = regTemp;
354                 listaAsmb->ultimo->op2.value = aux->op2.value;
```

```
355
356                 listaAsmb->ultimo->op3.type = regTemp;
357                 listaAsmb->ultimo->op3.value = aux->op3.value;
358                 break;
359
360             case MAI:
361                 insere_ASSMB(SGT);
362
363                 listaAsmb->ultimo->op1.type = regTemp;
364                 listaAsmb->ultimo->op1.value = aux->op1.value;
365
366                 listaAsmb->ultimo->op2.type = regTemp;
367                 listaAsmb->ultimo->op2.value = aux->op2.value;
368
369                 listaAsmb->ultimo->op3.type = regTemp;
370                 listaAsmb->ultimo->op3.value = aux->op3.value;
371                 break;
372
373             case MAIGL:
374                 insere_ASSMB(SGE);
375
376                 listaAsmb->ultimo->op1.type = regTemp;
377                 listaAsmb->ultimo->op1.value = aux->op1.value;
378
379                 listaAsmb->ultimo->op2.type = regTemp;
380                 listaAsmb->ultimo->op2.value = aux->op2.value;
381
382                 listaAsmb->ultimo->op3.type = regTemp;
383                 listaAsmb->ultimo->op3.value = aux->op3.value;
384                 break;
385
386             case MEIGL:
387                 insere_ASSMB(SLE);
388
389                 listaAsmb->ultimo->op1.type = regTemp;
390                 listaAsmb->ultimo->op1.value = aux->op1.value;
391
392                 listaAsmb->ultimo->op2.type = regTemp;
393                 listaAsmb->ultimo->op2.value = aux->op2.value;
394
395                 listaAsmb->ultimo->op3.type = regTemp;
396                 listaAsmb->ultimo->op3.value = aux->op3.value;
397                 break;
398
399
400             case SOM:
401                 insere_ASSMB(ADD);
402
403                 listaAsmb->ultimo->op1.type = regTemp;
404                 listaAsmb->ultimo->op1.value = aux->op1.value;
405
406                 listaAsmb->ultimo->op2.type = regTemp;
407                 listaAsmb->ultimo->op2.value = aux->op2.value;
408
409                 listaAsmb->ultimo->op3.type = regTemp;
410                 listaAsmb->ultimo->op3.value = aux->op3.value;
411
412                 break;
413
414             case SUB:
```

```
415                 insere_ASSMB(SUBI);
416
417                 listaAsmb->ultimo->op1.type = regTemp;
418                 listaAsmb->ultimo->op1.value = aux->op1.value;
419
420                 listaAsmb->ultimo->op2.type = regTemp;
421                 listaAsmb->ultimo->op2.value = aux->op2.value;
422
423                 listaAsmb->ultimo->op3.type = regTemp;
424                 listaAsmb->ultimo->op3.value = aux->op3.value;
425
426                 break;
427
428             case MUL:
429                 insere_ASSMB(MULT);
430
431                 listaAsmb->ultimo->op1.type = regTemp;
432                 listaAsmb->ultimo->op1.value = aux->op1.value;
433
434                 listaAsmb->ultimo->op2.type = regTemp;
435                 listaAsmb->ultimo->op2.value = aux->op2.value;
436
437                 listaAsmb->ultimo->op3.type = regTemp;
438                 listaAsmb->ultimo->op3.value = aux->op3.value;
439
440                 break;
441
442             case DIV:
443                 insere_ASSMB(DIVI);
444
445                 listaAsmb->ultimo->op1.type = regTemp;
446                 listaAsmb->ultimo->op1.value = aux->op1.value;
447
448                 listaAsmb->ultimo->op2.type = regTemp;
449                 listaAsmb->ultimo->op2.value = aux->op2.value;
450
451                 listaAsmb->ultimo->op3.type = regTemp;
452                 listaAsmb->ultimo->op3.value = aux->op3.value;
453
454                 break;
455
456             case imed:
457                 insere_ASSMB(LI);
458
459                 listaAsmb->ultimo->op1.type = regTemp;
460                 listaAsmb->ultimo->op1.value = aux->op1.value;
461
462                 listaAsmb->ultimo->op3.type = Const;
463                 listaAsmb->ultimo->op3.value = aux->op3.value;
464                 break;
465
466             case jump:
467                 insere_ASSMB(JUMP);
468                 listaAsmb->ultimo->op1.type = labelk;
469                 listaAsmb->ultimo->op1.value = aux->op3.value;
470                 break;
471
472             case beq:
473                 insere_ASSMB(BEQ);
474                 listaAsmb->ultimo->op1.type = regTemp;
```

```
475                 listaAsmb->ultimo->op1.value = aux->op1.value;
476
477                 listaAsmb->ultimo->op2.type = regTemp;
478                 listaAsmb->ultimo->op2.value = aux->op2.value;
479
480                 listaAsmb->ultimo->op3.type = labelk;
481                 listaAsmb->ultimo->op3.value = aux->op3.value;
482                 break;
483
484             case bne:
485                 insere_ASSMB(BNE);
486                 listaAsmb->ultimo->op1.type = regTemp;
487                 listaAsmb->ultimo->op1.value = aux->op1.value;
488
489                 listaAsmb->ultimo->op2.type = regTemp;
490                 listaAsmb->ultimo->op2.value = aux->op2.value;
491
492                 listaAsmb->ultimo->op3.type = labelk;
493                 listaAsmb->ultimo->op3.value = aux->op3.value;
494                 break;
495
496             default:
497                 printf("no nao achado\n");
498                 break;
499
500
501                 }
502
503                 aux = aux->proximo;
504         }
505
506 }
507 void GeraAssmb(void){
508
509
510     listaAsmb = (lista_ASSMB*)malloc(sizeof(lista_ASSMB));
511
512     listaAsmb->primeiro = NULL;
513     listaAsmb->ultimo = NULL;
514     listaAsmb->tamanho = 0;
515
516         percorre_lista();
517
518         imprime_ASSMB(listaAsmb->primeiro);
519
520     GeraExec(listaAsmb->primeiro);
521 }
```

D Código assmb.h

```
/*****************************************************/
/*      COMPILADOR PARA LINGUAGEM C-                 */
/*                                                    */
/*                                                   */
/*      Daiana Santos   RA: 120.357                  */
/*****************************************************/

#ifndef _ASSMB_H_
#define _ASSMB_H_

void GeraAssmb(void);

typedef struct nodeassmb {
    typeQuad type;
    char *name;
    int value;

}assmbNode;

typedef enum {
    NOP,
    HALT,
    WAIT,
    SW,
    LW,
    LI,
    MOVE,
    ADD,
    SUBI,
    MULT,
    DIVI,
    LABEL,
    BEQ,
    BNE,
    SET,
    SDT,
    SGT,
    SGE,
    SLT,
    SLE,
    JUMP,
    JAL,
    JR,
    IN,
    OUT

} OPCODE;

struct t_assmb {

    OPCODE opcode;
    assmbNode op1;
    assmbNode op2;
    assmbNode op3;
```

```
55
56      struct t_assmb *proximo;
57  };
58
59  typedef struct t_assmb INSTRU;
60
61  typedef struct {
62      INSTRU *primeiro;
63      INSTRU *ultimo;
64      int tamanho;
65
66  } lista_ASSMB;
67
68  //Variavel que contem a lista encadeada de quadruplas
69  lista_ASSMB *listaAsmb;
70
71  #endif
```

E Código cgen.c

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos    RA: 120.357                */
6  /****************************************************/
7
8  #include "globals.h"
9  #include "symtab.h"
10 #include "analyze.h"
11 #include "cgen.h"
12 #include "assmb.h"
13 #include <stdio.h>
14 #include <string.h>
15
16 char* escopoAux = "global";
17
18 int temp = 0;
19 int linha = 0;
20
21 void temporario(void){
22     temp++;
23
24     if(temp == 4){
25         temp = 1;
26     }
27 }
28
29 void insere(instr instrucao, char* op1, char* op2, char* op3) {
30
31     quadrupla *novo;
32     novo=(quadrupla*)malloc(sizeof(quadrupla));
33
34     novo->proximo= NULL;
35     novo->instrucao = instrucao;
36     novo->op1.name = op1;
37     novo->op2.name = op2;
38     novo->op3.name = op3;
39
40     if(lista_quad->tamanho==0) {
41
42         lista_quad->primeiro=novo;
43
44     } else {
45
46         lista_quad->ultimo->proximo=novo;
47
48     }
49
50     lista_quad->ultimo=novo;
51     lista_quad->tamanho++;
52
53 }
54
```

```
55 //imprime a lista de quadrupla
56 void imprime(quadrupla *quadrupla_aux) {
57     char* aux;
58 
59     while(quadrupla_aux!=NULL) {
60 
61         switch (quadrupla_aux->instrucao){
62             case nop:       aux = "nop";break;
63             case halt:      aux = "halt";break;
64             case store:      aux = "store";break;
65             case fun:       aux = "fun";break;
66             case arg:       aux = "arg";break;
67             case setArg:    aux = "setArg";break;
68             case call:      aux = "call";break;
69             case end:       aux = "end";break;
70             case load:      aux = "load";break;
71             case alloc:     aux = "alloc";break;
72             case ret:      aux = "ret";break;
73             case label:     aux = "label";break;
74             case IGLIGL:    aux = "==";break;
75             case DIF:       aux = "!=";break;
76             case MEN:       aux = "<";break;
77             case MAI:       aux = ">";break;
78             case MEIGL:     aux = "<=";break;
79             case MAIGL:     aux = ">=";break;
80             case SOM:       aux = "+";break;
81             case SUB:       aux = "-";break;
82             case MUL:       aux = "*";break;
83             case DIV:       aux = "/";break;
84             case imed:      aux = "imed";break;
85             case jump:      aux = "jump";break;
86             case beq:       aux = "beq";break;
87         }
88 
89         printf("( %-6s,", aux);
90 
91         switch (quadrupla_aux->op1.type){
92             case geral:printf(" %-10s,",quadrupla_aux->op1.name );break;
93             case id: break;
94             case regTemp:printf(" t_%-8d,", quadrupla_aux->op1.value);break;
95             case Const: printf(" %-10d,",quadrupla_aux->op1.value );break;
96             case labelk: printf(" L_%-8d,",quadrupla_aux->op1.value );break;
97         }
98 
99         switch (quadrupla_aux->op2.type){
100             case geral:printf(" %-10s,",quadrupla_aux->op2.name );break;
101             case id: break;
102             case regTemp:printf(" t_%-8d,", quadrupla_aux->op2.value);break;
103             case Const: printf(" %-10d,",quadrupla_aux->op2.value );break;
104             case labelk: printf(" L_%-8d,",quadrupla_aux->op2.value );break;
105         }
106 
107         switch (quadrupla_aux->op3.type){
108             case geral:printf(" %-10s )\n",quadrupla_aux->op3.name );break;
109             case id: break;
110             case regTemp:printf(" t_%-8d )\n", quadrupla_aux->op3.value);break;
111             case Const: printf(" %-10d )\n",quadrupla_aux->op3.value );break;
112             case labelk: printf(" L_%-8d )\n",quadrupla_aux->op3.value );break;
113         }
114 
```

```
115         quadrupla_aux = quadrupla_aux->proximo;
116 
117     }
118 }
119 
120 
121 //constroi a quadrupla referente ao n   da arvore
122 int make_quad(TreeNode * tree) {
123 
124     quadrupla *quadrupla_aux;
125     TreeNode * aux;
126     int direita, esquerda, auxiliar;
127     int endr = 0;
128 
129     if (tree->nodekind==statementK) {
130         switch (tree->kind.stmt) {
131         case ifK:
132 
133             esquerda = make_quad(tree->child[0]);
134 
135             insere(beq,"","","");
136             lista_quad->ultimo->op1.type = regTemp;
137             lista_quad->ultimo->op1.value = esquerda;
138 
139             lista_quad->ultimo->op2.type = regTemp;
140             lista_quad->ultimo->op2.value = 0;
141 
142             lista_quad->ultimo->op3.type = labelk;
143             linha++;
144             lista_quad->ultimo->op3.value = linha;
145 
146             auxiliar = linha;
147             percorre(tree->child[1]);
148 
149             if(tree->child[2] == NULL){
150                 insere(label,"", "", "");
151                 lista_quad->ultimo->op3.type = labelk;
152                 lista_quad->ultimo->op3.value = auxiliar;
153                 break;
154             }else{
155                 insere(jump, "", "", "");
156                 lista_quad->ultimo->op3.type = labelk;
157                 linha++;
158                 lista_quad->ultimo->op3.value = linha;
159 
160                 insere(label,"", "", "");
161                 lista_quad->ultimo->op3.type = labelk;
162                 lista_quad->ultimo->op3.value = auxiliar;
163 
164                 percorre(tree->child[2]);
165 
166                 insere(label,"", "", "");
167                 lista_quad->ultimo->op3.type = labelk;
168                 lista_quad->ultimo->op3.value = auxiliar+1;
169 
170             }
171             break;
172 
173 
174         case whileK:
```

```
175             linha++;
176 
177             insere(label,"", "", "");
178             lista_quad->ultimo->op3.type = labelk;
179             lista_quad->ultimo->op3.value = linha;
180             esquerda = make_quad(tree->child[0]);
181 
182             insere(beq,"","","");
183             lista_quad->ultimo->op1.type = regTemp;
184             lista_quad->ultimo->op1.value = esquerda;
185 
186             lista_quad->ultimo->op2.type = regTemp;
187             lista_quad->ultimo->op2.value = 0;
188 
189             lista_quad->ultimo->op3.type = labelk;
190             linha++;
191             lista_quad->ultimo->op3.value = linha;
192 
193             auxiliar = linha;
194             percorre(tree->child[1]);
195 
196             insere(jump, "", "", "");
197             lista_quad->ultimo->op3.type = labelk;
198             lista_quad->ultimo->op3.value = auxiliar-1;
199 
200             insere(label,"", "", "");
201             lista_quad->ultimo->op3.type = labelk;
202             lista_quad->ultimo->op3.value = auxiliar;
203 
204             break;
205 
206 
207         case assignK:
208             direita = make_quad(tree->child[1]);
209 
210             endr = busca_end(tree->child[0]->attr.name, escopoAux);
211 
212 
213             if(tree->child[0]->child[0] != NULL){
214 
215                 esquerda = make_quad(tree->child[0]->child[0]);
216                 insere(store, tree->child[0]->attr.name, "", "");
217                 lista_quad->ultimo->op2.type = regTemp;
218                 lista_quad->ultimo->op2.value = esquerda;
219 
220             }else{
221                 insere(store, tree->child[0]->attr.name, "", "");
222             }
223 
224             lista_quad->ultimo->op1.endereco = endr;
225 
226             lista_quad->ultimo->op3.type = regTemp;
227             lista_quad->ultimo->op3.value = direita;
228 
229 
230             break;
231 
232 
233         case returnK:
234             if(tree->child[0] != NULL){
```

```
235                 esquerda = make_quad(tree->child[0]);
236                 insere(ret, "", "", "");
237                 lista_quad->ultimo->op1.type = regTemp;
238                 lista_quad->ultimo->op1.value = esquerda;
239             }
240             else{
241                 insere(ret, "", "", "");
242             }
243             break;
244 
245         default:
246             break;
247         }
248     } else if (tree->nodekind==expressionK) {
249         switch (tree->kind.exp) {
250         case operationK:
251 
252             if(tree->child[0]->kind.exp != operationK && tree->child[1]->kind.exp ==
                    operationK){
253 
254                 direita = make_quad(tree->child[1]);
255                 esquerda = make_quad(tree->child[0]);
256 
257             }else{
258                 esquerda = make_quad(tree->child[0]);
259                 direita = make_quad(tree->child[1]);
260 
261             }
262                 insere(tree->attr.op, "", "", "");
263                 temp++;
264                 lista_quad->ultimo->op1.type = regTemp;
265                 lista_quad->ultimo->op1.value = temp;
266 
267                 lista_quad->ultimo->op2.type = regTemp;
268                 lista_quad->ultimo->op2.value = esquerda;
269 
270 
271                 lista_quad->ultimo->op3.type = regTemp;
272                 lista_quad->ultimo->op3.value = direita;
273                 return temp;
274             break;
275 
276         case constantK:
277             insere(imed, "", "", "");
278             lista_quad->ultimo->op1.type = regTemp;
279             temporario();
280             lista_quad->ultimo->op1.value = temp;
281             lista_quad->ultimo->op3.type = Const;
282             lista_quad->ultimo->op3.value = tree->attr.val;
283             return temp;
284             break;
285 
286 
287         case idK:
288 
289                 endr = busca_end(tree->attr.name, escopoAux);
290 
291 
292             if(tree->child[0] != NULL){
293                 direita = make_quad(tree->child[0]);
```

```
294                 insere(load, "", tree->attr.name, "");
295                 lista_quad->ultimo->op3.type = regTemp;
296                 lista_quad->ultimo->op3.value = direita;
297 
298             }else{
299                 insere(load, "", tree->attr.name, "");
300 
301             }
302             lista_quad->ultimo->op2.endereco = endr;
303 
304             lista_quad->ultimo->op1.type = regTemp;
305             temporario();
306             lista_quad->ultimo->op1.value = temp;
307             return temp;
308                 break;
309 
310 
311         case variableK:
312              endr = busca_end(tree->attr.name, escopoAux);
313 
314             if(tree->child[0] != NULL){
315 
316                //verifica se o filho eh um const
317                 if(tree->child[0]->kind.exp == constantK){
318 
319                     //se for CONST busca o valor do vetor
320                     insere(alloc, tree->attr.name, "", escopoAux);
321                     lista_quad->ultimo->op2.type = Const;
322                     lista_quad->ultimo->op2.value = tree->child[0]->attr.val;
323 
324                 }
325 
326             }else{
327                 insere(alloc, tree->attr.name, "", escopoAux);
328                 lista_quad->ultimo->op2.type = Const;
329                 lista_quad->ultimo->op2.value = 1;
330 
331             }
332             lista_quad->ultimo->op1.endereco = endr;
333             break;
334 
335 
336         case functionK:
337             insere(fun, tree->attr.name, "", "");
338             escopoAux = tree->attr.name;
339             linha++;
340             lista_quad->ultimo->op1.value = linha;
341             percorre(tree->child[0]);
342             percorre(tree->child[1]);
343             break;
344 
345 
346         case activationK:
347 
348             aux = tree->child[0];
349             while(aux != NULL){
350 
351                 auxiliar = make_quad(aux);
352                 insere(setArg, "", "", "");
353                 lista_quad->ultimo->op3.type = regTemp;
```

```
354                 lista_quad->ultimo->op3.value = auxiliar;
355 
356                 aux = aux->sibling;
357             }
358             insere(call, "", "",tree->attr.name);
359 
360             if(getFunType(tree->attr.name) == INTTYPE){
361 
362                 if(strcmp(tree->attr.name,"output") != 0){
363                     temporario();
364                     lista_quad->ultimo->op1.type = regTemp;
365                     lista_quad->ultimo->op1.value = temp;
366                 }
367             }
368             return temp;
369             break;
370 
371 
372         case typeK:
373             percorre(tree->child[0]);
374             break;
375 
376 
377         case paramK:
378             endr = busca_end(tree->attr.name, escopoAux);
379 
380 
381             insere(arg, tree->attr.name, "", escopoAux);
382 
383             lista_quad->ultimo->op1.endereco = endr;
384             break;
385 
386 
387         default:
388             break;
389         }
390     }
391     return -1;
392 
393 }
394 
395 //percorre a arvore
396 void percorre( TreeNode * t) {
397     int i, x;
398 
399     if (t != NULL) {
400 
401         x = make_quad(t);
402 
403         percorre(t->sibling);
404         if(t->nodekind==expressionK && t->kind.exp == functionK){
405             insere(end, t->attr.name, "", "");
406         }
407 
408     }
409 }
410 
411 
412 
413 //Constroi a lista de quadruplas
```

```
414 void gen_quad(TreeNode * tree) {
415
416     lista_quad=(lista*)malloc(sizeof(lista));
417
418     lista_quad->primeiro=NULL;
419     lista_quad->ultimo=NULL;
420     lista_quad->tamanho=0;
421
422     insere(nop, "", "", "");
423
424     percorre(tree);
425
426     insere(halt, "", "", "");
427
428     imprime(lista_quad->primeiro);
429
430 }
```

F Código cgen.h

```
/*****************************************************/
/*      COMPILADOR PARA LINGUAGEM C-                 */
/*                                                    */
/*                                                   */
/*      Daiana Santos   RA: 120.357                  */
/*****************************************************/

#ifndef _CGEN_H_
#define _CGEN_H_


//funcao que cria as quadruplas
void gen_quad(TreeNode * tree);

void percorre(TreeNode * t);

typedef enum {geral, id, Const, regTemp, labelk, funck} typeQuad;

typedef struct nodeQuad {

    typeQuad type;
    char* name;
    int regTemp;
    int value;
    int endereco;


}quadNode;


typedef enum {
        nop,
        halt,
        store,
        fun,
        arg,
        setArg,
        call,
    load,
    alloc,
    ret,
    label,
    beq,
    bne,
    jump,
    imed,
        end
} instr;

struct t_quad {

    instr instrucao;
    quadNode op1;
    quadNode op2;
```

```
55      quadNode op3;
56
57      struct t_quad *proximo;
58  };
59
60  typedef struct t_quad quadrupla;
61
62  typedef struct {
63      quadrupla *primeiro;
64      quadrupla *ultimo;
65      int tamanho;
66
67  } lista;
68
69  //Variavel que contem a lista encadeada de quadruplas
70  lista *lista_quad;
71
72  #endif
```

G Código exec.c

```
/*****************************************************/
/*      COMPILADOR PARA LINGUAGEM C-                 */
/*                                                    */
/*                                                   */
/*      Daiana Santos   RA: 120.357                  */
/*****************************************************/

#include "globals.h"
#include "cgen.h"
#include "assmb.h"
#include "exec.h"
#include <stdio.h>
#include <string.h>

#define MAX 10000

void GeraExec(INSTRU *aux){

        int PC = 0;
        int vet_linhas[MAX];
        int opcode;
        int formato;
        int val_op1;
        int val_op2;
        int val_op3;

        printf( "\n\nExecutavel: \n");
        printf( "module MEMORIA_INSTRUCAO(\n");
        printf( "       input [11:0] PC,\n");
        printf( "       output wire [31:0] INSTRUCAO\n);\n");
        printf( "       wire [31:0] memoria[511:0];\n\n");


        while(aux != NULL){

                printf("        assign memoria[%d]=", PC);
                switch (aux->opcode){
                        case NOP:
                                opcode = 23;
                                formato = 4;
                                val_op1 = 0;
                                break;


                        case HALT:
                                opcode = 21;
                                formato = 4;
                                val_op1 = 0;
                                break;


                        case WAIT:
                                opcode = 22;
                                formato = 4;
```

```
55                                  val_op1 = 0;
56                                  break;
57
58
59                          case SW:
60                                  opcode = 18;
61                                  formato = 2;
62                                  val_op1 = aux->op1.value;
63                                  val_op2 = aux->op2.value;
64                                  val_op3 = 0;
65                                  break;
66
67
68                          case LW:
69                                  opcode = 16;
70                                  formato = 2;
71                                  val_op1 = aux->op1.value;
72                                  val_op2 = aux->op2.value;
73                                  val_op3 = 0;
74                                  break;
75
76
77                          case LI:
78                                  opcode = 17;
79                                  formato = 3;
80                                  val_op1 = aux->op1.value;
81                                  val_op2 = aux->op2.value;
82                                  break;
83
84
85                          case MOVE:
86                                  opcode = 19;
87                                  formato = 2;
88                                  val_op1 = aux->op1.value;
89                                  val_op2 = aux->op2.value;
90                                  val_op3 = 0;
91                                  break;
92
93
94                          case ADD:
95                                  opcode = 1;
96                                  formato = 1;
97                                  val_op1 = aux->op1.value;
98                                  val_op2 = aux->op2.value;
99                                  val_op3 = aux->op3.value;
100                                 break;
101
102
103                         case SUB:
104                                 opcode = 2;
105                                 formato = 1;
106                                 val_op1 = aux->op1.value;
107                                 val_op2 = aux->op2.value;
108                                 val_op3 = aux->op3.value;
109                                 break;
110
111
112                         case MULT:
113                                 opcode = 3;
114                                 formato = 1;
```

```
115                                 val_op1 = aux->op1.value;
116                                 val_op2 = aux->op2.value;
117                                 val_op3 = aux->op3.value;
118                                 break;
119
120
121                         case DIV:
122                                 opcode = 4;
123                                 formato = 1;
124                                 val_op1 = aux->op1.value;
125                                 val_op2 = aux->op2.value;
126                                 val_op3 = aux->op3.value;
127                                 break;
128
129
130                         case LABEL:
131                                 vet_linhas[aux->op1.value] = PC;
132                                 opcode = 23;
133                                 formato = 4;
134                                 val_op1 = 0;
135                                 break;
136
137
138                         case BEQ:
139                                 opcode = 11;
140                                 formato = 2;
141                                 val_op1 = aux->op1.value;
142                                 val_op2 = aux->op2.value;
143                                 val_op3 = aux->op3.value;
144                                 break;
145
146
147                         case BNE:
148                                 opcode = 12;
149                                 formato = 2;
150                                 val_op1 = aux->op1.value;
151                                 val_op2 = aux->op2.value;
152                                 val_op3 = aux->op3.value;
153                                 break;
154
155
156                         case SET:
157                                 opcode = 5;
158                                 formato = 1;
159                                 val_op1 = aux->op1.value;
160                                 val_op2 = aux->op2.value;
161                                 val_op3 = aux->op3.value;
162                                 break;
163
164
165                         case SDT:
166                                 opcode = 6;
167                                 formato = 1;
168                                 val_op1 = aux->op1.value;
169                                 val_op2 = aux->op2.value;
170                                 val_op3 = aux->op3.value;
171                                 break;
172
173
174                         case SGT:
```

```
175                                 opcode = 7;
176                                 formato = 1;
177                                 val_op1 = aux->op1.value;
178                                 val_op2 = aux->op2.value;
179                                 val_op3 = aux->op3.value;
180                                 break;
181
182
183                         case SGE:
184                                 opcode = 8;
185                                 formato = 1;
186                                 val_op1 = aux->op1.value;
187                                 val_op2 = aux->op2.value;
188                                 val_op3 = aux->op3.value;
189                                 break;
190
191                         case SLT:
192                                 opcode = 9;
193                                 formato = 1;
194                                 val_op1 = aux->op1.value;
195                                 val_op2 = aux->op2.value;
196                                 val_op3 = aux->op3.value;
197                                 break;
198
199
200                         case SLE:
201
202                                 opcode = 10;
203                                 formato = 1;
204                                 val_op1 = aux->op1.value;
205                                 val_op2 = aux->op2.value;
206                                 val_op3 = aux->op3.value;
207                                 break;
208
209
210                         case JUMP:
211                                 opcode = 13;
212                                 formato = 4;
213                                 val_op1 = vet_linhas[aux->op1.value];
214                                 val_op2 = 0;
215                                 break;
216
217
218                         case JAL:
219                                 opcode = 15;
220                                 formato = 4;
221                                 val_op1 = vet_linhas[aux->op1.value];
222                                 val_op2 = 0;
223                                 break;
224
225                         case JR:
226                                 opcode = 14;
227                                 formato = 3;
228                                 val_op1 = aux->op1.value;
229                                 val_op2 = 0;
230                                 break;
231
232
233                         case IN:
234                                 opcode = 20;
```

```
235                                 formato = 4;
236                                 val_op1 = 0;
237                                 break;
238
239
240                         case OUT:
241                                 opcode = 24;
242                                 formato = 3;
243                                 val_op1 = aux->op1.value;
244                                 val_op2 = 0;
245                                 break;
246
247
248                 }
249
250                 printf("{5'd%d, ", opcode);
251                 if(formato == 1){
252                         printf("5'd%d, 5'd%d, 5'd%d, 12'd0 }\n", val_op1, val_op2,
                            val_op3);
253                 }else if(formato == 2){
254                         printf("5'd%d, 5'd%d, 17'd%d }\n", val_op1, val_op2, val_op3);
255                 }else if(formato == 3){
256                         printf("5'd%d, 22'd%d }\n", val_op1, val_op2);
257                 }else{
258                         printf("27'd%d }\n", val_op1);
259                 }
260
261                 PC++;
262                 aux = aux->proximo;
263         }
264
265         printf("\n      assign INSTRUCAO = memoria [PC]; \n\nendmodule\n");
266         printf("\n\n          *** EXECUTAVEL GERADO!!! ***\n\n");
267 }
```

H Código exec.h

```
1  /*****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                    */
5  /*       Daiana Santos   RA: 120.357                 */
6  /*****************************************************/
7
8  #ifndef _BIN_H_
9  #define _BIN_H_
10
11 void GeraExec(INSTRU *aux);
12
13 #endif
```

I Código globals.h

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos    RA: 120.357                */
6  /****************************************************/
7
8
9  #ifndef _GLOBALS_H_
10 #define _GLOBALS_H_
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <ctype.h>
15 #include <string.h>
16
17
18
19 #ifndef YYPARSER
20
21
22 #include "Cmenos.tab.h"
23
24
25 #define ENDFILE 0
26
27 #endif
28
29 #ifndef FALSE
30 #define FALSE 0
31 #endif
32
33 #ifndef TRUE
34 #define TRUE 1
35 #endif
36
37 #define MAXRESERVED 6
38
39 extern FILE* source;
40 extern FILE* listing;
41 extern FILE* code;
42
43 extern int lineno;
44
45
46 typedef int TokenType;
47
48 typedef enum
49 {
50         statementK, expressionK
51 } NodeKind;
52
53 typedef enum
54 {
```

```
55          ifK, whileK, assignK, returnK
56
57  } StatementKind;
58
59  typedef enum
60  {
61          variableK, paramK, functionK, callK,
62     operationK, activationK, constantK, idK,
63     vectorK, vectorIdK, typeK, numberK
64
65  } ExpressionIdentifier;
66
67  typedef enum {INTTYPE, VOIDTYPE, BOOLTYPE} dataTypes;
68
69  typedef enum {VAR, FUN} IDTypes;
70
71  /* ExpType is used for type checking */
72  typedef enum
73  {
74          voidK, integerK, booleanK
75
76  } ExpressionType;
77
78
79  #define MAXCHILDREN 3
80
81
82  typedef struct treeNode
83  {
84           struct treeNode * child[MAXCHILDREN];
85       struct treeNode * sibling;
86       int lineno;
87       int size;
88       int add;
89       NodeKind nodekind;
90
91       union
92       {
93                  StatementKind stmt;
94          ExpressionIdentifier exp;
95       } kind;
96
97       union
98       {
99              TokenType op;
100         int val;
101        //   int len;
102         char* name;
103        //   char* scope;
104      } attr;
105      dataTypes type; /* for type checking of exps */
106 } TreeNode;
107
108
109 extern int EchoSource;
110
111 extern int TraceScan;
112
113 extern int TraceParse;
114
```

```
115 extern int TraceAnalyze;
116
117 extern int TraceCode;
118
119 extern int Error;
120
121 #endif
```

J Código main.c

```
/****************************************************/
/*      COMPILADOR PARA LINGUAGEM C-                */
/*                                                   */
/*                                                   */
/*      Daiana Santos   RA: 120.357                 */
/****************************************************/

#include "globals.h"

#define NO_PARSE FALSE
#define NO_ANALYZE FALSE
#define NO_CODE FALSE
#define NO_ASSMB FALSE

#include "util.h"
#if NO_PARSE
#include "scan.h"
#else
#include "parse.h"
#if !NO_ANALYZE
#include "analyze.h"
#if !NO_CODE
#include "cgen.h"
#if !NO_ASSMB
#include "assmb.h"

#endif
#endif
#endif
#endif

/* allocate global variables */
int lineno = 0;
FILE * source;
FILE * listing;
FILE * code;

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = TRUE;
int TraceParse = TRUE;
int TraceAnalyze = TRUE;
int TraceCode = FALSE;

int Error = FALSE;

int main( int argc, char * argv[] )
{ TreeNode * syntaxTree;
  char pgm[120]; /* source code file name */
  if (argc != 2)
    { fprintf(stderr,"usage: %s <filename>\n",argv[0]);
      exit(1);
    }
  strcpy(pgm,argv[1]) ;
```

```
55    if (strchr (pgm, '.') == NULL)
56       strcat(pgm,".cms");
57    source = fopen(pgm,"r");
58    if (source==NULL)
59    { fprintf(stderr,"File %s not found\n",pgm);
60      exit(1);
61    }
62    listing = stdout; /* send listing to screen */
63    fprintf(listing,"\nCompilador C menos: %s\n",pgm);
64  #if NO_PARSE
65    while (getToken()!=ENDFILE);
66  #else
67    syntaxTree = parse();
68    if (TraceParse) {
69      fprintf(listing,"\nArvore Sintatica:\n");
70      printTree(syntaxTree);
71    }
72  #if !NO_ANALYZE
73   // if (Error)
74    //{
75  if (TraceAnalyze) fprintf(listing,"\nConstruindo Tabela de Simbolos...\n");
76      buildSymtab(syntaxTree);
77      if (TraceAnalyze) fprintf(listing,"\nChecando tipos...\n");
78      typeCheck(syntaxTree);
79      if (TraceAnalyze) fprintf(listing,"\nChecagem de tipos terminada\n");
80   // }
81  #if !NO_CODE
82    if (! Error)
83    {
84      fprintf(listing,"\n\n\n           LISTA DE QUADRUPLAS:\n\n");
85      gen_quad(syntaxTree);
86    }
87  #if !NO_ASSMB
88    GeraAssmb();
89
90  #endif
91  #endif
92  #endif
93  #endif
94    fclose(source);
95    return 0;
96  }
```

K Código symtab.c

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos    RA: 120.357                */
6  /****************************************************/
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "symtab.h"
12
13 /* SIZE is the size of the hash table */
14 #define SIZE 211
15
16 /* SHIFT is the power of two used as multiplier
17    in hash function  */
18 #define SHIFT 4
19
20 /* the hash function */
21 int hash (char* name)
22 {
23     int temp = 0;
24     int i = 0;
25     while (name[i] != '\0')
26     {
27         temp = ((temp << SHIFT) + name[i]) % SIZE;
28         ++i;
29     }
30     i = 0;
31    return temp;
32 }
33
34
35 void st_insert(char * name, int lineno, int op, char* escopo, dataTypes DType, IDTypes
       IType)
36 {
37
38     int h = hash(name);
39     BucketList l =  hashTable[h];
40
41     while ((l != NULL) && ((strcmp(name,l->name) != 0))){
42         l = l->next;
43     }
44
45     if (l == NULL || (op != 0 && l->escopo != escopo && l->IType != FUN)) /* variable not
           yet in table */
46     {
47         l = (BucketList) malloc(sizeof(struct BucketListRec));
48         l->name = name;
49         l->lines = (LineList) malloc(sizeof(struct LineListRec));
50         l->lines->lineno = lineno;
51         l->memloc = op;
52         l->IType = IType;
```

```
53          l->Dtype = DType;
54          l->escopo = escopo;
55          l->lines->next = NULL;
56          l->next = hashTable[h];
57          hashTable[h] = l;
58      }
59      else if( l->IType == FUN  && IType == VAR){
60      fprintf(listing,"Erro: Nome da variavel %s j  utilizada como nome de fun  o.[%d]\n
            ",name, lineno);
61      Error = TRUE;
62    }
63    else if( l->escopo == escopo && op != 0)
64    {
65      fprintf(listing,"Erro: Variavel %s j  declarada neste escopo.[%d]\n",name, lineno);
66      Error = TRUE;
67    }
68    else if(l->escopo != escopo && (strcmp(l->escopo,"global") != 0) ){
69      //procura por variavel global entes de supor que n o existe
70      while ((l != NULL)){
71        if((strcmp(l->escopo, "global")==0)&& ((strcmp( name,l->name) == 0))){
72          LineList t = l->lines;
73          while (t->next != NULL) t = t->next;
74          t->next = (LineList) malloc(sizeof(struct LineListRec));
75          t->next->lineno = lineno;
76          t->next->next = NULL;
77          break;
78        }
79        l = l->next;
80      }
81      if(l == NULL){
82        fprintf(listing,"Erro: Variavel %s n o declarada neste escopo.[%d]\n",name, lineno
              );
83        Error = TRUE;
84      }
85    }
86    else if(op == 0)
87    {
88      LineList t = l->lines;
89      while (t->next != NULL) t = t->next;
90      t->next = (LineList) malloc(sizeof(struct LineListRec));
91      t->next->lineno = lineno;
92      t->next->next = NULL;
93    }
94  }
95
96  /* Function st_lookup returns the memory
97   * location of a variable or -1 if not found
98   */
99  int st_lookup (char* name)
100 {
101   int h = hash(name);
102   BucketList l =  hashTable[h];
103   while ((l != NULL) && (strcmp(name,l->name) != 0))
104         l = l->next;
105   if (l == NULL)
106       return -1;
107   else
108       return l->memloc;
109 }
110
```

```
111 int busca_end ( char * name, char * fun)
112 {
113   int h = hash(name);
114   BucketList l =  hashTable[h];
115   while ((l != NULL) && (strcmp(name,l->name) != 0) && (strcmp(fun,l->escopo) != 0))
116     l = l->next;
117   if (l == NULL) return -1;
118   else return l->memloc;
119 }
120 
121 void busca_main ()
122 {
123   int h = hash("main");
124   BucketList l =  hashTable[h];
125   while ((l != NULL) && ((strcmp("main",l->name) != 0 || l->IType == VAR)))
126     l = l->next;
127   if (l == NULL) {
128     fprintf(listing,"Erro: Fun   o main n o declarada\n");
129     Error = TRUE;
130   }
131 
132 }
133 dataTypes getFunType(char* nome){
134   int h = hash(nome);
135   BucketList l =  hashTable[h];
136   while ((l != NULL) && (strcmp(nome,l->name) != 0))
137     l = l->next;
138 
139   if (l == NULL) return -1;
140   else return l->Dtype;
141 }
142 
143 
144 /* Procedure printSymTab prints a formatted
145  * listing of the symbol table contents
146  * to the listing file
147  */
148 void printSymTab(FILE * listing)
149 { int i;
150   fprintf(listing," Nome      Escopo      TipoID        TipoDado     Linha Numeros\n");
151   fprintf(listing,"------    ---------   ----------    ----------    ------------\n");
152   for (i=0;i<SIZE;++i)
153   { if (hashTable[i] != NULL)
154     { BucketList l = hashTable[i];
155       while (l != NULL)
156       { LineList t = l->lines;
157         //fprintf(listing,"%-8d ",l->memloc);
158         fprintf(listing,"%-10s  ",l->name);
159                     fprintf(listing,"%-12s  ",l->escopo);
160 
161         char* id, *data;
162         if(l->IType == VAR){
163           id = "VAR";
164         }
165         else if(l->IType == FUN){
166           id = "FUN";
167         }
168         if(l->Dtype == INTTYPE){
169           data = "INT";
170         }
```

```
171         else if(l->Dtype == VOIDTYPE){
172           data = "VOID";
173         }
174         fprintf(listing,"%-10s  ",id);
175         fprintf(listing,"%-10s  ",data);
176 
177         while (t != NULL)
178         { fprintf(listing,"%4d ",t->lineno);
179           t = t->next;
180         }
181         fprintf(listing,"\n");
182         l = l->next;
183       }
184     }
185   }
186 } /* imprimeTabelaSimbolos */
```

L Código sytamb.h

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos   RA: 120.357                 */
6  /****************************************************/
7
8  #ifndef _SYMTAB_H_
9  #define _SYMTAB_H_
10
11 #include "globals.h"
12
13 #define SIZE 211
14
15 typedef struct LineListRec
16    { int lineno;
17      struct LineListRec * next;
18    } * LineList;
19
20 typedef struct BucketListRec
21    { char * name;
22      dataTypes Dtype;
23      IDTypes IType;
24      char* escopo;
25      LineList lines;
26      int memloc ; /* memory location for variable */
27      struct BucketListRec * next;
28    } * BucketList;
29
30 BucketList hashTable[SIZE];
31
32 int hash ( char * key );
33
34  void st_insert( char * name, int lineno, int loc, char* escopo, dataTypes Dtype, IDTypes
       IType );
35
36
37 int st_lookup (char * name);
38
39 int busca_end ( char * name, char * fun);
40
41
42 void printSymTab(FILE * listing);
43
44 void busca_main ();
45 dataTypes getFunType(char* nome);
46
47 #endif
```

M Código util.c

```
1  /****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                   */
5  /*       Daiana Santos    RA: 120.357                */
6  /****************************************************/
7
8  #include "globals.h"
9  #include "util.h"
10
11
12
13 void printToken( TokenType token, const char* tokenString )
14 {
15   switch (token)
16   {
17         case IF: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
18     case ELSE: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
19     case INT: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
20     case RETURN: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
21     case VOID: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
22     case WHILE: fprintf(listing, "Palavra Reservada: %s\n",tokenString); break;
23     case IGL: fprintf(listing,"Simbolo: =\n"); break;
24     case MEN: fprintf(listing,"Simbolo: <\n"); break;
25     case IGLIGL: fprintf(listing,"Simbolo: ==\n"); break;
26     case MAI: fprintf(listing,"Simbolo: >\n"); break;
27     case MEIGL: fprintf(listing, "Simbolo: <=\n"); break;
28     case MAIGL: fprintf(listing, "Simbolo: >=\n"); break;
29     case DIF: fprintf(listing, "Simbolo: !=\n"); break;
30     case ACO: fprintf(listing, "Simbolo: [\n"); break;
31     case FCO: fprintf(listing, "Simbolo: ]\n"); break;
32     case ACH: fprintf(listing, "Simbolo: {\n"); break;
33     case FCH: fprintf(listing, "Simbolo: }\n"); break;
34     case APR: fprintf(listing,"Simbolo: (\n"); break;
35     case FPR: fprintf(listing,"Simbolo: )\n"); break;
36     case PEV: fprintf(listing,"Simbolo: ;\n"); break;
37     case VIR: fprintf(listing,"Simbolo: ,\n"); break;
38     case SOM: fprintf(listing,"Simbolo: +\n"); break;
39     case SUB: fprintf(listing,"Simbolo: -\n"); break;
40     case MUL: fprintf(listing,"Simbolo: *\n"); break;
41     case DIV: fprintf(listing,"Simbolo: /\n"); break;
42     case ENDFILE: fprintf(listing,"EOF\n"); break;
43     case NUM:
44       fprintf(listing, "Numero: %s\n",tokenString);
45       break;
46     case ID:
47       fprintf(listing, "Identificador: %s\n",tokenString);
48       break;
49     case ERR:
50       fprintf(listing, "Erro: %s\n",tokenString);
51       break;
52     default:
53       fprintf(listing,"Token desconhecido: %d\n",token);
54   }
```

```
55 }
56
57
58 TreeNode * newStmtNode(StatementKind kind)
59 {
60     TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
61     int i;
62     if (t==NULL)
63         fprintf(listing,"Erro de acesso de memoria na linha %d\n",lineno);
64     else
65     {
66         for (i=0;i<MAXCHILDREN;i++)
67             t->child[i] = NULL;
68         t->sibling = NULL;
69         t->nodekind = statementK;
70         t->kind.stmt = kind;
71         t->lineno = lineno;
72     }
73     return t;
74 }
75
76
77 TreeNode * newExpNode(ExpressionIdentifier kind)
78 {
79     TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
80     int i;
81     if (t==NULL)
82         fprintf(listing,"Erro de acesso de memoria na linha %d\n",lineno);
83     else
84     {
85         for (i=0;i<MAXCHILDREN;i++)
86             t->child[i] = NULL;
87         t->sibling = NULL;
88         t->nodekind = expressionK;
89         t->kind.exp = kind;
90         t->lineno = lineno;
91         t->type = VOID;
92     }
93     return t;
94 }
95
96
97 char * copyString(char * s)
98 {
99     int n;
100    char * t;
101    if (s==NULL)
102        return NULL;
103    n = strlen(s)+1;
104    t = malloc(n);
105    if (t==NULL)
106        fprintf(listing,"Erro de acesso de memoria na linha %d\n",lineno);
107    else
108        strcpy(t,s);
109    return t;
110 }
111
112
113 static int indentno = 0;
114
```

```
115 #define INDENT indentno+=2
116 #define UNINDENT indentno-=2
117
118 static void printSpaces(void)
119 {
120   int i;
121     for (i=0;i<indentno;i++)
122       fprintf(listing," ");
123 }
124
125 void printTree( TreeNode * tree )
126 { int i;
127   INDENT;
128   while (tree != NULL) {
129     printSpaces();
130     if (tree->nodekind==statementK)
131     { switch (tree->kind.stmt) {
132         case ifK:
133           fprintf(listing,"If\n");
134           break;
135         case assignK:
136           fprintf(listing,"Atribuicao\n");
137           break;
138         case whileK:
139          fprintf(listing,"While\n");
140         break;
141         case returnK:
142                fprintf(listing, "Return\n");
143              break;
144         default:
145           fprintf(listing,"Tipo StatementNode desconhecido\n");
146           break;
147       }
148     }
149
150   else if(tree->nodekind== expressionK)
151   { switch (tree->kind.exp){
152     case operationK:
153       fprintf(listing,"Op: ");
154       printToken(tree->attr.op,"\0");
155     break;
156     case variableK:
157     fprintf(listing,"Variavel: %s\n", tree->attr.name);
158     break;
159     case functionK:
160     fprintf(listing,"Funcao: %s\n", tree->attr.name);
161       break;
162     case constantK:
163     fprintf(listing,"Constante: %d\n", tree->attr.val);
164     break;
165     case idK:
166     fprintf(listing,"Id: %s\n",tree->attr.name);
167     break;
168     case activationK:
169     fprintf(listing,"Ativa  o: %s\n",tree->attr.name);
170     break;
171     case typeK:
172       fprintf(listing,"Tipo: %s\n",tree->attr.name);
173       break;
174     case paramK:
```

```
175       fprintf(listing,"Var: %s\n",tree->attr.name);
176       break;
177       case vectorK:
178       fprintf(listing,"Vetor: %s\n",tree->attr.name);
179       break;
180       case vectorIdK:
181       fprintf(listing,"Indice [%d]\n",tree->attr.val);
182       break;
183       default:
184       fprintf(listing,"Tipo ExpNode desconhecido\n");
185       break;
186       }
187     }
188       else fprintf(listing,"Tipo de no desconhecido\n");
189       for (i=0;i<MAXCHILDREN;i++)
190            printTree(tree->child[i]);
191       tree = tree->sibling;
192     }
193     UNINDENT;
194   }
```

N Código util.h

```
1  /*****************************************************/
2  /*       COMPILADOR PARA LINGUAGEM C-                */
3  /*                                                    */
4  /*                                                    */
5  /*       Daiana Santos   RA: 120.357                 */
6  /*****************************************************/
7
8  #ifndef _UTIL_H_
9  #define _UTIL_H_
10
11 void printToken( TokenType, const char* );
12
13 TreeNode * newStmtNode(StatementKind );
14
15 TreeNode * newExpNode(ExpressionIdentifier );
16
17
18 char * copyString( char * );
19
20 void printTree( TreeNode * );
21
22 #endif
```