

**UNIVERSIDADE FEDERAL DE VIÇOSA – CAMPUS FLORESTAL**

**DANIEL FREITAS MARTINS – 2304**

**JOÃO ARTHUR GONÇALVES DO VALE – 3025**

**MARIA DALILA VIEIRA – 3030**

**NAIARA CRISTIANE DOS REIS DINIZ – 3005**

**Relatório referente ao Trabalho Prático 4 – Análise Semântica e  
Geração de Código Intermediário para a linguagem Chameleon**

**Florestal**

**2020**

**DANIEL FREITAS MARTINS – 2304**  
**JOÃO ARTHUR GONÇALVES DO VALE – 3025**  
**MARIA DALILA VIEIRA – 3030**  
**NAIARA CRISTIANE DOS REIS DINIZ – 3005**

**Relatório referente ao Trabalho Prático 4 – Análise Semântica e  
Geração de Código Intermediário para a linguagem Chameleon**

Documentação apresentada à disciplina CCF  
441 – Compiladores do curso de Bacharelado  
em Ciência da Computação da Universidade  
Federal de Viçosa – *Campus Florestal*.

Orientador: Daniel Mendes Barbosa

**Florestal**

**2020**

## SUMÁRIO

<b>1 - Introdução</b>	<b>3</b>
<b>2 - A Linguagem Chameleon</b>	<b>4</b>
<b>3 - Atualizações na Gramática</b>	<b>4</b>
<b>4 - Modificações no Analisador Léxico</b>	<b>6</b>
<b>5 - Modificações no Analisador Sintático</b>	<b>7</b>
5.1 - Análise Semântica - Recursos do YACC	8
5.2 - Análise Semântica - Uso da Tabela de Símbolos	11
5.3 - Geração de Código Intermediário	15
<b>6 - Geração de Código de máquina (RISC, LLVM)</b>	<b>19</b>
6.1 - RISC	21
6.2 - LLVM	23
<b>7 - Programa de Teste Geral (Código de Três Endereços)</b>	<b>27</b>
<b>8 - Considerações Finais</b>	<b>28</b>
<b>Referências Bibliográficas</b>	<b>30</b>
<b>Apêndice A - Código lex.l para a linguagem Chameleon</b>	<b>31</b>
<b>Apêndice B - Código translate.y para a linguagem Chameleon</b>	<b>35</b>
<b>Apêndice C - symbol.h</b>	<b>47</b>
<b>Apêndice D - symbol.c</b>	<b>48</b>
<b>Apêndice E - hash-table.h</b>	<b>51</b>
<b>Apêndice F - hash-table.c</b>	<b>52</b>
<b>Apêndice G - custom_defines.h</b>	<b>56</b>
<b>Apêndice H - y.tab.h (gerado pelo YACC)</b>	<b>57</b>
<b>Apêndice I - montador_lexer.h</b>	<b>60</b>
<b>Apêndice J - montador_lexer.c</b>	<b>61</b>

## 1 - Introdução

Neste trabalho são apresentadas as alterações nos arquivos do Trabalho Prático 3 para permitir ao compilador realizar as etapas de Análise Semântica e Geração de Código Intermediário para a linguagem Chameleon. Vários recursos das ferramentas LEX e YACC foram utilizados para a realização dessas etapas e alguns exemplos são a explicitação de tipos de *tokens* e variáveis, e estabelecimento de ordem de precedência em operações aritméticas. Além destes, a Tabela de Símbolos foi usada em conjunto em várias situações, incluindo a realização do *backpatch* (ou remendo) para instruções de desvio e instruções de repetição. Sobretudo, em adicional ao código intermediário e a árvore de sintaxe abstrata usada implicitamente, também implementamos a tradução do código intermediário para código de máquina RISC-V e para a IR do LLVM. Por meio do uso do conjunto de instruções do LLVM conseguimos simular operações simples descritas em nossa linguagem.

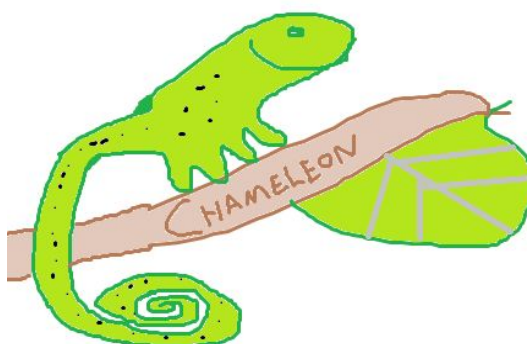
É importante ressaltar que o *script* para a geração do executável do compilador foi alterado. Com a existência dos novos arquivos responsáveis pelo analisador sintático e pela tabela de símbolos, a seguinte sequência de comandos pode ser utilizada para gerar o executável em questão:

1	<b>flex</b> lex.l
2	<b>yacc</b> translate.y <b>-d -v</b>
3	<b>gcc</b> symbol.h symbol.c <b>-lm</b> montador_lexer.h montador_lexer.c <b>-lm</b> hash-table.h hash-table.c y.tab.c lex.yy.c y.tab.h <b>-ll</b>

O parâmetro **-v** na segunda linha é opcional e serve apenas para imprimir informações adicionais em relação à geração do analisador sintático. Os três comandos podem ser colocados em um arquivo *script.sh*, separados por **&&**, para realizar a compilação de uma forma mais conveniente, sendo necessário apenas executar este *script*, se dadas as devidas permissões de execução. Para utilizar o executável do compilador gerado, use **./a.out arquivo\_entrada**,

## 2 - A Linguagem Chameleon

A linguagem de programação Chameleon pretende oferecer um conjunto de comandos que seja interessante para programadores que desejam agilidade, simplicidade e espaço para customização. Sua construção foi baseada nas linguagens C, C++ e Python. Assim, temos uma linguagem imperativa procedural que segue o paradigma estrutural. A origem deste nome veio justamente desta característica de customização da linguagem com o uso de seu pré-processador. Em resumo, ela permite definir novas formas de escrita que são convertidas para a sintaxe padrão da linguagem. Essa ideia é similar ao uso de macros na linguagem C, mas com um propósito diferente. A logo da linguagem Chameleon pode ser visualizada na Fig. 2.1 a seguir. Maiores detalhes a respeito das alterações na gramática da linguagem, nos analisadores léxico e sintático, a criação do analisador semântico e a geração de código intermediário, serão abordados nas próximas seções.



**Figura 2.1:** Logo da linguagem Chameleon. Fonte: Autores, 2020.

## 3 - Atualizações na Gramática

Para a realização das tarefas de análise semântica e geração de código intermediário, foi necessário fazer algumas alterações pontuais na gramática. Tais alterações tiveram como objetivo permitir a utilização de recursos do YACC para facilitar o processo de análise semântica e para permitir a geração de código intermediário.

A alteração na variável *math\_expression* consistiu da eliminação da variável *binary\_operators*, de modo que todas as operações aritméticas e lógicas, com todos os

tokens possíveis, fossem transferidas para *math\_expression* diretamente. Isso foi necessário para o estabelecimento de ordem de precedência das operações aritméticas (veja Seção 5.1).

A alteração da gramática nas estruturas de desvio e nas estruturas de repetição foram necessárias para a aplicação do *backpatch*. Na redução de uma das variáveis extras criada, faz-se algo para permitir o *backpatch* futuro, como a emissão de código com rótulo de desvio vazio, e salvando-se o valor da linha corrente no respectivo símbolo da Tabela de Símbolos.

As alterações podem ser visualizadas logo abaixo. De vermelho estão indicadas as produções removidas ou alteradas, e de azul estão indicadas as produções resultantes das adições e alterações:

**1-a)** *math\_expression* → *ex\_aux\_abre binary\_operators math\_expression*  
                                   | *unary\_operators expression*  
                                   | *ex\_aux\_abre*

**1-b)** *ex\_aux\_abre* → '(' *math\_expression ex\_aux\_fecha*  
                                   | *math\_term*

**2-** *binary\_operators* → ... (REMOVIDO)

**1-a)** *math\_expression* → *ex\_aux\_abre add\_operator ex\_aux\_abre*  
                                   | *ex\_aux\_abre div\_operator ex\_aux\_abre*  
                                   | *ex\_aux\_abre pow\_operator ex\_aux\_abre*  
                                   | *ex\_aux\_abre logic\_operator ex\_aux\_abre*  
                                   | *ex\_aux\_abre rel\_operator ex\_aux\_abre*  
                                   | *unary\_operators ex\_aux\_abre*  
                                   | *math\_term*

**1-b)** *ex\_aux\_abre* → '(' *math\_expression ex\_aux\_fecha*  
                                   | *math\_expression*

**3-** *if\_command* → **if** *expression : statement endif*  
                                   | **if** *expression : statement elif expression : statement endelif*

**3-a)** *if\_command* → *if\_cond statement endif*  
                                   | *if\_cond statement elif\_cond statement endelif*

**3-b)** *if\_cond* → **if** *expression :*

**3-c)** *elif\_cond* → **elif** *expression :*

**4- *for\_command* → **for** *expression* , *expression* , *expression* : *statement* **endfor****

**4-a) *for\_command* → **for** *for\_cond* *for\_cond\_aux* *statement* **endfor****

**4-b) *for\_cond* → *expression* , *expression* ,**

**4-c) *for\_cond\_aux* → *expression* :**

**5- *while\_command* → **while** *expression* : *statement* **endwhile****

**5-a) *while\_command* → **while** *while\_cond* *statement* **endwhile****

**5-b) *while\_cond* → *expression* :**

## 4 - Modificações no Analisador Léxico

Nesta etapa do trabalho, o grupo teve de implementar uma forma de passar os valores lidos do LEX para o analisador sintático YACC. Para isso, a estrutura *symbol* e a Tabela de Símbolos foi utilizada. Essa construção permitiu realizar as tarefas de análise semântica e geração de código de três endereços de maneira mais fluida e natural.

Para fazer isso, as ações correspondentes aos casamentos de padrão foram alteradas, apenas. Note que parte disso havia sido feito no trabalho prático anterior de maneira simplista apenas para imprimir a Tabela de Símbolos. Nesta etapa, os valores agora são atribuídos de acordo com seus tipos, como pode-se observar na Fig. 4.1 abaixo. Das linhas 1 a 5, note que o valor lido de uma *string* é armazenado na estrutura de símbolo e passado desta forma para o YACC. Note que um tamanho máximo associado à palavra foi definido apenas para simplificar certas operações. Caso contrário, veja que para permitir tamanhos variados, poderia ser criada uma estrutura própria, capaz de redimensionar seu vetor de caracteres por meio de *realloc* quando necessário, por exemplo.

Ainda em relação à Fig. 4.1, note que das linhas 6 a 13 os operadores relacionais foram separados de acordo com o que significam cada um. Isso foi necessário para a geração de código intermediário, uma vez que um único *token* é retornado para representar esse tipo de operador. Tais constantes referentes a cada operador estão definidas no arquivo *custom\_defines.h*. Veja, ainda, que das linhas 15 a 22, os números reais e inteiros são armazenados nas estruturas de símbolos, de modo a aproveitar a construção da Tabela de Símbolos, facilitando assim algumas operações que envolvem verificações de tipos, por exemplo.

```

1  /* Outras regras */
2  <word_condition>{word_value} {
3      createSymbol(&(yyval.symbol), NULL, _WORD, _WORD);
4      (yyval.symbol)->data->v.word = (char*) malloc(MAX_TAM_WORD * sizeof(char));
5      sprintf((yyval.symbol)->data->v.word, "%s", yytext);
6      return WORD_VALUE;}
7  {rel_operator} {
8      if(strcmp(yytext, "=") == 0) yyval.op = EQ;
9      else if(strcmp(yytext, "!=") == 0) yyval.op = NE;
10     else if(strcmp(yytext, ">=") == 0) yyval.op = GE;
11     else if(strcmp(yytext, "<=") == 0) yyval.op = LE;
12     else if(strcmp(yytext, ">") == 0) yyval.op = GT;
13     else if(strcmp(yytext, "<") == 0) yyval.op = LT;
14     return REL_OPERATOR;}
15
16  {real_number} {remover_espacos_e_print(_REAL);
17      createSymbol(&(yyval.symbol), NULL, _REAL, _REAL);
18      (yyval.symbol)->data->v.real = atof(yytext);
19      return REAL_NUMBER;}
20  {number} {remover_espacos_e_print(_NUMBER);
21      createSymbol(&(yyval.symbol), NULL, _INTEGER, _INTEGER);
22      (yyval.symbol)->data->v.integer = atoi(yytext);
23      return NUMBER;}
24  /* Outras regras */

```

**Figura 4.1:** Trecho do arquivo correspondente ao analisador léxico. Exemplos de mudanças realizadas são mostradas.

## 5 - Modificações no Analisador Sintático

Para realizar o que foi pedido neste trabalho prático, foi necessário realizar modificações no arquivo *translate.y*, correspondente ao arquivo de acordo com as especificações de Bison/YACC. O uso da tabela de símbolos foi essencial para salvarmos as referências (de acordo com os escopos) e as informações das variáveis e de outros valores importantes para a análise semântica e geração de código intermediário.



Outros recursos, oferecidos pelo próprio YACC, foram utilizados para auxiliar no processo de análise semântica também. A seguir, serão descritas as modificações feitas com os recursos do YACC e o uso da Tabela de Símbolos.

## 5.1 - Análise Semântica - Recursos do YACC

Como comentado anteriormente, o YACC oferece alguns recursos para facilitar o processo de análise semântica. O primeiro recurso oferecido é a possibilidade de redefinição do tipo de YYSTYPE. Uma vez redefinido, o analisador sintático pode receber como valor para a variável *yylval* o valor correspondente ao tipo que se deseja. Dessa forma, esse valor irá corresponder a um tipo específico que, caso não seja respeitado, a própria ferramenta emitirá um erro informando que os tipos não correspondem. Como comentado na parte anterior deste trabalho, este recurso já estava sendo utilizado apenas para imprimir valores da Tabela de Símbolos. No entanto, para esta etapa, a redefinição de YYSTYPE pela cláusula *%union* foi alterada, como mostrado na Fig. 5.1.1 abaixo. A Fig. 5.1.1. mostra, ainda, a alteração da cláusula *%code requires* que indica as dependências referentes às funções auxiliares criadas para a geração do código intermediário.

```

1  %code requires{
2      #include "hash-table.h"
3      void action_math_expression(Symbol *sd, Symbol *s1, char op, Symbol *s2);
4      void action_math_expression_logic(Symbol *sd, Symbol *s1, char op, Symbol *s2);
5      void action_math_expression_rel(Symbol *sd, Symbol *s1, char op, Symbol *s2);
6      void action_math_expression_unary(Symbol *sd, char op, Symbol *s1);
7      void emitBackpatch(int linha_origem, int linha_destino);
8  }
9
10 %union{
11     Symbol *symbol;
12     hashtable_t *symbol_table;
13     char type_aux[20]; // apenas para print...
14     char op;
15     int type_declaration;
16     int id_temporario;
17 }
```

**Figura 5.1.1:** Trecho de *translate.y* para mostrar mudanças em *%union* e *%code*.

Outro recurso oferecido pelo YACC é a especificação de tipos de *tokens* e variáveis. Para *tokens*, pode-se utilizar o esquema **%token<TIPO> NOME\_TOKEN** e, para variáveis, **%type<TIPO> NOME\_VARIAVEL**. Uma forma equivalente de se fazer isso pelo YACC é especificar no próprio uso da variável da pilha em alguma ação semântica de uma dada produção. Por exemplo, **\$<symbol>\$ = \$<symbol>1** especifica que o tipo de retorno da variável correspondente àquela ação semântica será do tipo *symbol* e corresponderá ao endereço do valor do primeiro parâmetro, também *symbol*. Dessa forma, caso após a redução em alguma parte da gramática não corresponda aos tipos informados, um erro é gerado pela ferramenta para informar essa inconsistência.

Para lidar com expressões aritméticas e estabelecer as ordens das operações, o grupo pensou na criação de uma árvore de expressões e até mesmo na geração das expressões na notação pós-fixada. Como isso iria ficar complexo para este trabalho, e até mesmo pelo tempo para a realização do mesmo, procuramos se o YACC possui algum mecanismo para auxiliar nesta tarefa. Por sorte, o YACC permite especificar a precedência através das palavras-chave **%left** e **%right** [2]. Operadores de mesma precedência podem ser colocados na sequência da especificação, enquanto que operadores de precedência diferentes podem ser especificados em linhas diferentes. A ordem de precedência acaba por se dar também na ordem em que são especificados, da menor para a maior. Veja a Fig. 5.1.2 abaixo. Ela mostra um exemplo de como um trecho é entendido pelo YACC. A Fig. 5.1.3 mostra como isso está definido para a linguagem Chameleon.

1	<b>Se especificar o seguinte no YACC:</b>
2	<b>%right</b> '='
3	<b>%left</b> '+' '-'
4	<b>%left</b> '*' '/'
5	<b>A expressão</b>
6	a = b = c * d - e - f * g
7	<b>Será entendida pelo YACC como</b>
8	a = ( b = ( ((c * d) - e) - (f * g) ) )

**Figura 5.1.2:** Exemplo de controle de precedências pelo YACC. Retirado de [2].

1	<b>%right</b> '='
2	<b>%left</b> ADD_OPERATOR NEG_OPERATOR
3	<b>%left</b> DIV_OPERATOR
4	<b>%left</b> POW_OPERATOR

5	%left LOGIC_OPERATOR
6	%left REL_OPERATOR
7	%left '('

**Figura 5.1.3:** Definição de precedências de operadores aritméticos e lógicos para a linguagem Chameleon.

É importante ressaltar que para o uso das especificações da Fig. 5.1.3, foi necessário reformular as produções relacionadas às variáveis *math\_expression* e *ex\_aux\_abre*: os *tokens* associados às regras de precedência devem vir imediatamente onde se quer estabelecer as ordens de precedência, sem passar por uma variável auxiliar, como estava sendo feito anteriormente. Fazer essa modificação na gramática foi melhor do que criar uma estrutura para definir as ordens de precedência, uma vez que o próprio YACC oferece este recurso. A Fig. 5.1.4 mostra um trecho das produções vinculadas às variáveis *ex\_aux\_abre*, *ex\_aux\_fecha* e *math\_expression*. A partir das precedências definidas na Fig. 5.1.3, as operações de geração de código das operações aritméticas torna-se possível de se fazer da forma indicada. Note que a redução de *math\_term* faz a emissão de códigos para uma atribuição a partir do valor correspondente de determinada operação para um registrador temporário. Essa emissão é apenas para auxiliar as demais impressões de código, feitas por funções auxiliares, como a *action\_math\_expression* e outras ações semânticas de outras produções, como *command* e *expression*, por exemplo.

1	<i>ex_aux_abre</i> : '(' <i>math_expression</i> <i>ex_aux_fecha</i> {PRINT(("ex_aux_abre -> ['(' <i>math_expression</i> <i>ex_aux_fecha</i> ]\n"))
2	\$<symbol>\$ = \$<symbol>2;
3	}
4	<i>math_expression</i> { \$<symbol>\$ = \$<symbol>1; }
5	;
6	
7	<i>ex_aux_fecha</i> : ')' {PRINT(("ex_aux_fecha -> [')']\n"))}
8	;
9	
10	<i>math_expression</i> : <i>ex_aux_abre</i> ADD_OPERATOR <i>ex_aux_abre</i> {
11	PRINT(("math_expression -> [ex_aux_abre ADD_OPERATOR ex_aux_abre]\n"))
12	createSymbol(&(\$<symbol>\$), "", \$<symbol>1->token_type, \$<symbol>1->data->type);
13	action_math_expression(\$<symbol>\$, \$<symbol>1, \$<op>2, \$<symbol>3);
14	}
15	<i>ex_aux_abre</i> DIV_OPERATOR <i>ex_aux_abre</i> {
16	PRINT(("math_expression -> [ex_aux_abre DIV_OPERATOR ex_aux_abre]\n"))
17	createSymbol(&(\$<symbol>\$), "", \$<symbol>1->token_type, \$<symbol>1->data->type);
18	action_math_expression(\$<symbol>\$, \$<symbol>1, \$<op>2, \$<symbol>3);
19	}
20	<i>ex_aux_abre</i> POW_OPERATOR <i>ex_aux_abre</i> {

```

21      PRINT(("math_expression -> [ex_aux_abre POW_OPERATOR ex_aux_abre]\n"))
22      createSymbol(&($<symbol>$), "", $<symbol>1->token_type, $<symbol>1->data->type);
23      action_math_expression($<symbol>$, $<symbol>1, $<op>2, $<symbol>3);
24  }
25  /* ..... */
26  | math_term          { PRINT(("math_expression -> [math_term]\n"))
27      char s_aux[200];
28      if($<symbol>1->token_type != _VARIABLE) {
29          if($<symbol>1->fake_memory_address == -1) $<symbol>1->fake_memory_address = id_temporario++;
30          if($<symbol>1->data->type == _INTEGER) { sprintf(s_aux, "%d", $<symbol>1->data->v.integer);
31              createADDi(&block_mount, $<symbol>1->fake_memory_address, $<symbol>1->fake_memory_address,
32              $<symbol>1->data->v.integer); }
33          else if($<symbol>1->data->type == _REAL) sprintf(s_aux, "%lf", $<symbol>1->data->v.real);
34          else if($<symbol>1->data->type == _WORD) sprintf(s_aux, "%s", $<symbol>1->data->v.word);
35          char s[1000]; sprintf(s, "t%d = %s", $<symbol>1->fake_memory_address, s_aux); emit(s);
36      }
37      $<symbol>$ = $<symbol>1;
38  };

```

**Figura 5.1.3:** Definição de precedências de operadores aritméticos e lógicos para a linguagem Chameleon.

## 5.2 - Análise Semântica - Uso da Tabela de Símbolos

A Tabela de Símbolos foi utilizada para referenciar variáveis e manter valores e outras informações das mesmas, de maneira a auxiliar o processo de análise semântica e geração de código intermediário. A partir disso, foi possível identificar o seguinte:

- **Verificação de redeclaração de variável:** Se em um mesmo escopo um mesmo identificador é utilizado para declarar outra variável, um erro é emitido. Como usamos uma estrutura de Tabela de Símbolos multinível, declarações de variáveis com o mesmo nome de variáveis já declaradas por escopos "pais" é permitida. Essa verificação é simples: ao identificar uma declaração de variável, a função *ht\_get* (Fig. 5.2.1) é utilizada na Tabela de Símbolos atual para o lexema do *token* recebido. Como a Tabela de Símbolos foi construída a partir da estrutura de uma tabela *hash*, essa verificação é trivial. A Fig. 5.2.2 mostra um exemplo em que ambas as situações ocorrem: veja que a redeclaração de uma variável, em um escopo diferente, é permitido. No entanto, uma redeclaração em um mesmo escopo gera um erro customizado, indicando a linha e a variável que está sendo redeclorada.

```

1 Symbol *ht_get( hashtable_t *hashtable, char *key, int find_fathers ) {
2     int bin = 0;
3     entry_t *pair;
4     bin = ht_hash( hashtable, key );
5     pair = hashtable->table[ bin ];
6     while( pair != NULL && pair->key != NULL && strcmp( key, pair->key ) > 0 ) {
7         pair = pair->next;
8     }
9
10    if( pair == NULL || pair->key == NULL || strcmp( key, pair->key ) != 0 ) {
11        if( hashtable->previous_hash != NULL && find_fathers == 1 ){
12            return ht_get( hashtable->previous_hash, key, find_fathers ); // Visitando o pai (chamada recursiva)...
13        }
14        return NULL;
15    } else {
16        return pair->value;
17    }
18 }

```

**Figura 5.2.1:** Função *ht\_get* de *hash-table.c*.

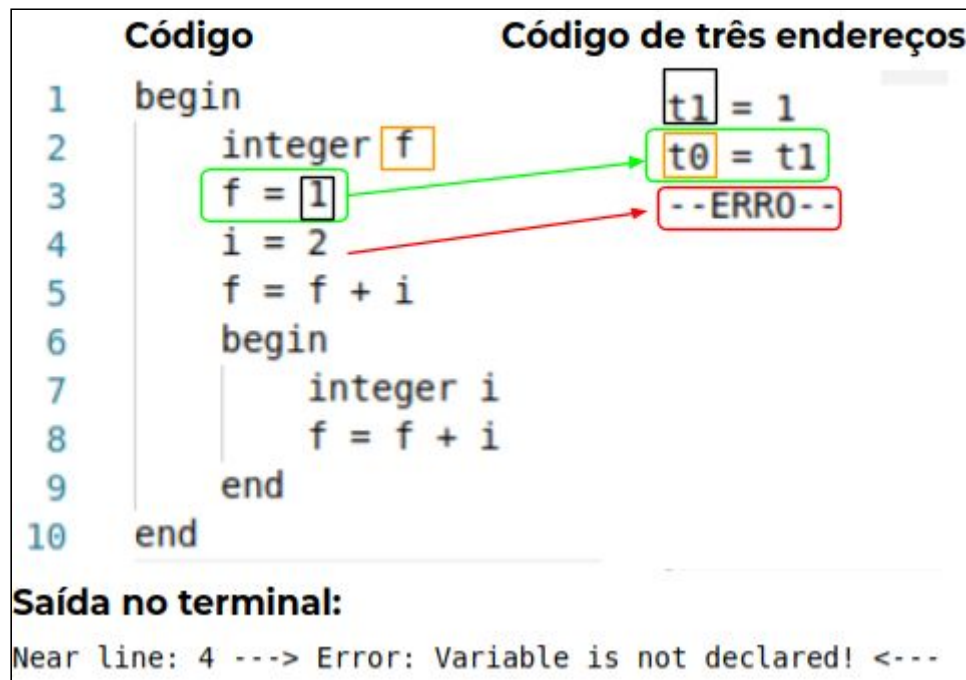
Código	Código de três endereços
1 begin	
2 real juros	t2 = 3.500000
3 real valor	t1 = t2
4 valor = 3.50	t3 = 1.500000
5 juros = 1.5	t0 = t3
6 begin	t6 = 2
7 integer juros	t4 = t6
8 real total	t7 = t1 * t4
9 juros = 2	t5 = t7
10 total = valor * juros	t9 = 987
11 end	t8 = t9
12 real total	--ERRO--
13 total = 987	
14 integer juros	
15 juros = 3	
16 total = valor * juros	
17 end	

**Saída no terminal:**  
variable\_declaration -> [type IDENTIFIER]  
Line: 14 ---> The variable juros has already been declared before! <---  
--ERRO--

**Figura 5.2.2:** Exemplo de código que gera erro de redeclaração de variável. Note que no escopo mais interno, a variável *juros* está sendo redeclarada, mas em seu próprio escopo. No entanto, *juros* está sendo redeclarada na linha 14, em um escopo que já existe essa variável, gerando o erro informado.

- **Verificação de uso de variável não declarada:** Um erro é emitido se uma operação envolve o uso de identificadores não presentes na Tabela de Símbolos

(veja Fig. 5.2.3). A escolha feita no trabalho prático anterior de se construir a Tabela de Símbolos a partir de uma estrutura *hash* tornou essa e outras verificações muito facilitadas. Note que basta olhar a partir da função *ht\_get* se a variável está presente ou não na Tabela de Símbolos atual ou superiores. Uma *flag* na função *ht\_get* foi introduzida para permitir essa busca recursiva “para cima”.



**Figura 5.2.3:** Exemplo de verificação de uso de variável não declarada.

- **Verificação de tipos:** Se uma variável tenta receber um valor que não corresponde ao seu tipo, ou uma operação é feita de forma errônea, com tipos diferentes, então um erro é emitido (veja Fig 5.2.5). Há, porém, a seguinte exceção: números inteiros (*integer*) somados com números reais (*real*) é permitido se o tipo da variável que receber for do tipo *real*. Essa construção tenta simular um *casting* automático desses valores. Os códigos responsáveis por essas tarefas são mostrados na Fig. 5.2.4. Note que na função *checkTypes*, mesmo se os tipos diferem, e *t1* for do tipo *real* e *t2* for do tipo *integer*, então a construção é permitida pela existência do *auto-casting*, mostrado na função *upTypes*.

```

1  int checkTypes(enum SymbolType t1, enum SymbolType t2){
2      if(t1 != t2){
3          if(t1 == _REAL && t2 == _INTEGER)

```

```

4     return 1;
5     return 0;
6 }
7 return 1;
8 }
9
10 void upTypes(Symbol *symbol, enum SymbolType t1, enum SymbolType t2){
11     if(t1 != t2){
12         if(t1 == _REAL && t2 == _INTEGER || (t1 == _INTEGER && t2 == _REAL))
13             symbol->data->type = _REAL;
14     }
15 }

```

**Figura 5.2.4:** Funções para verificação de tipos (*checkTypes*) e *auto-casting* (*upTypes*).

Código	Código de três endereços
1 begin	t0 = Junior
2 word nome	t2 = 10
3 integer idade	t1 = t2
4 nome = "Junior"	--ERRO--
5 idade = 10	
6 idade = "horizonte"	
7 end	

**Saída no terminal:**

```

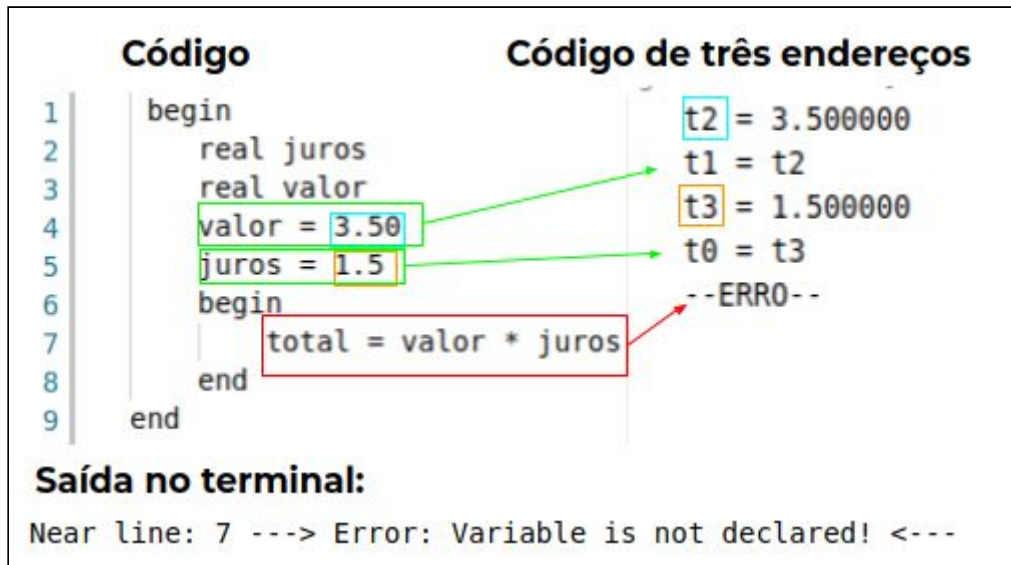
Near line: 7 ---> Error: Wrong data types in attribution - the variable idade
is a INTEGER and the value of the expression is a WORD <---

```

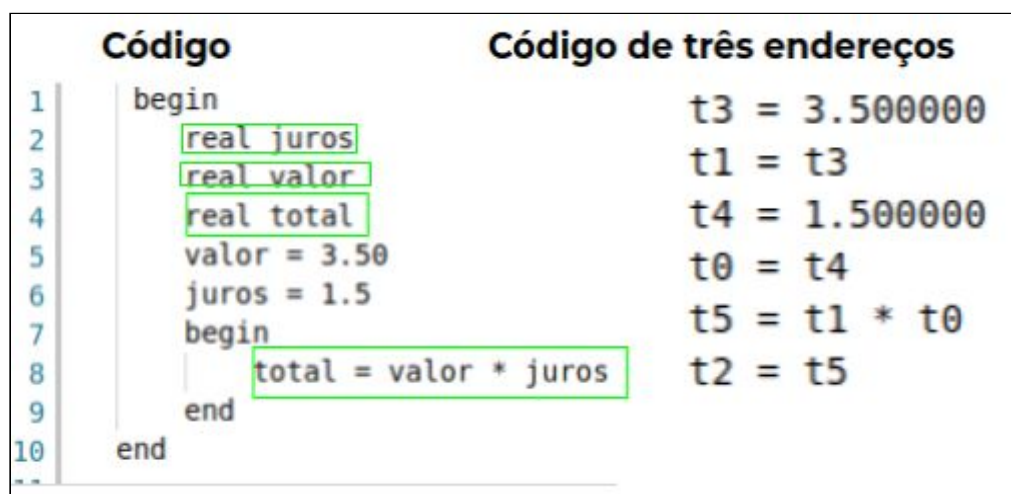
**Figura 5.2.5:** Exemplo de verificação de tipo.

- **Uso de variáveis em escopos diferentes:** Isso acontece quando uma variável tenta acessar uma variável não declarada em seu escopo (veja Fig. 5.2.6). Assim, os escopos “pais” são verificados em busca da variável requerida. Caso encontre, ela é usada e o código prossegue normalmente (veja Fig. 5.2.7). Caso não encontre, um erro de uso de variável não declarada é emitido. A forma como isso foi implementado foi da mesma maneira como fizemos para verificar se uma variável já estava declarada ou não. A estrutura da tabela *hash* e o encadeamento entre os escopos tornou isso possível.





**Figura 5.2.6:** Exemplo de uso de variáveis em escopos diferentes, no caso da variável não estar sendo declarada no escopo.



**Figura 5.2.7:** Exemplo de uso de variáveis em escopos diferentes, no caso da variável estar sendo declarada no escopo.

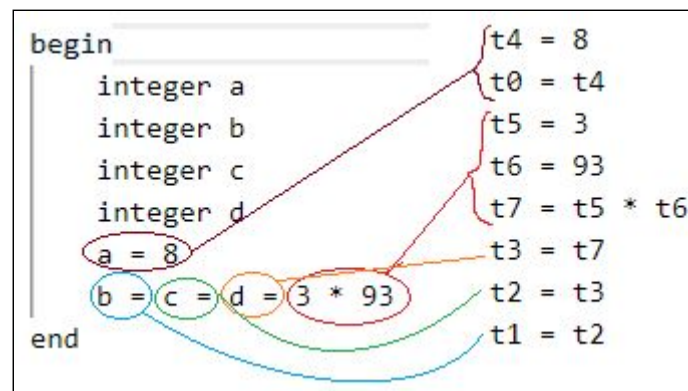
### 5.3 - Geração de Código Intermediário

A geração de código intermediário consistiu das representações definidas pelo grupo e pela representação do LLVM. Um código gerado para a arquitetura RISC-V também é possível de ser gerado à medida que as reduções são feitas na gramática.

Devido ao tempo que tivemos para essa tarefa, os comandos relacionados a *vector*, *squad* e *task* não foram gerados. A representação definida pelo grupo consistiu



de uma representação bem parecida com a qual vimos em sala de aula. Um exemplo de código gerado para a representação do grupo pode ser vista na Fig. 5.3.1 abaixo. Note que é possível de se fazer uma **atribuição múltipla** em uma mesma linha, como representado na figura: **b = c = d = 3 \* 93**. Note que o *fake\_memory\_address*, que havia sido definido no trabalho anterior como sendo o código *hash* do lexema, foi alterado para ser um inteiro incremental. Isso foi necessário apenas para facilitar a geração de representações intermediárias para o LLVM, por exemplo.



**Figura 5.3.1:** Exemplo de representação intermediária do grupo (à direita) para um determinado código simples de operações (à esquerda). As ligações indicam os códigos gerados para cada operação.

Funções auxiliares para a geração de código intermediário foram criadas. A Fig. 5.3.2 mostra uma delas, responsável por gerar os códigos de três endereços para as operações aritméticas. Ela mostra ainda a função *emit*. Note que, assim como foi visto na disciplina, a cada emissão de código, um contador de número de linhas é incrementado. As funções *createADD*, *createSUB*, etc. são responsáveis pela geração de código de RISC-V.

```

1 void action_math_expression(Symbol *sd, Symbol *s1, char op, Symbol *s2){
2   if(sd->fake_memory_address == -1) sd->fake_memory_address = id_temporario++;
3   upTypes(sd, s1->data->type, s2->data->type);
4   sprintf(sd->to_emit, "t%d = t%d %c t%d", sd->fake_memory_address, s1->fake_memory_address, op,
5   s2->fake_memory_address);
6   switch(op){
7     case '+': createADD(&block_mount, sd->fake_memory_address, s1->fake_memory_address,
8     s2->fake_memory_address); break;
9     case '-': createSUB(&block_mount, sd->fake_memory_address, s1->fake_memory_address,
10    s2->fake_memory_address); break;

```

```

11     case '*': createMUL(&block_mount, sd->fake_memory_address, s1->fake_memory_address,
12 s2->fake_memory_address); break;
13     case '/': createDIV(&block_mount, sd->fake_memory_address, s1->fake_memory_address,
14 s2->fake_memory_address); break;
15 }
16 //if($<symbol>2->token_type != _VARIABLE){ // imediato
17 emit(sd->to_emit);
18 }
19 // outras funções auxiliares: action_math_expression_logic, action_math_expression_rel e action_math_expression_unary
20 void emit(char *msg){
21     printf("%s\n", msg);
22     fprintf(arq_three_address_code, "%s\n", msg);
23     num_prox_instr++;
24 }

```

**Figura 5.3.2:** Funções auxiliares para emissão de código de três endereços.

Para as instruções de desvio e instruções de repetição, foi necessário implementar uma estratégia de *backpatch*, assim como vimos em aula. A estratégia segue a mesma ideia vista em aula, com o porém de que, como um arquivo é gerado para o código de instruções de três endereços, o *backpatch* consiste em alterar esse arquivo à medida que o código original vai sendo avaliado. Desta maneira, um arquivo temporário é criado e os devidos *backpatches* são realizados. Ao fim, o arquivo anterior é removido e o temporário é renomeado para ser o original. O arquivo é então reaberto em modo *append* para continuar a geração de código de três endereços. A Fig. 5.3.3 mostra a função responsável pela emissão dos *backpatches* da forma como foi descrito.

```

1 void emitBackpatch(int linha_origem, int linha_destino){
2     fclose(arq_three_address_code);
3     arq_three_address_code = NULL;
4     FILE *arq_three_address_code_temp = NULL;
5     arq_three_address_code = fopen("arq_three_address_code_generated", "r"); // abrindo para leitura
6     arq_three_address_code_temp = fopen("arq_three_address_code_generated_temp", "w"); // abrindo para escrita
7     if(arq_three_address_code == NULL){
8         raiseError("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated'\n");
9     }
10    if(arq_three_address_code_temp == NULL){
11        raiseError("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated_temp'\n");
12    }
13    int count = 0;
14    char buffer[255];
15    while ((fgets(buffer, 255, arq_three_address_code)) != NULL)
16    {
17        count++;
18
19        /* If current line is line to replace */
20        if(count == linha_origem){
21            buffer[strlen(buffer)-4] = '\0';
22            char *str_aux = (char*) malloc(20*sizeof(char));
23            sprintf(str_aux, "%d\n", (linha_destino+1));
24            strcat(buffer, str_aux);
25            free(str_aux);

```

```

26     fputs(buffer, arq_three_address_code_temp);
27 } else {
28     fputs(buffer, arq_three_address_code_temp);
29 }
30 }
31
32 fclose(arq_three_address_code);
33 fclose(arq_three_address_code_temp);
34
35 remove("arq_three_address_code_generated");
36
37 rename("arq_three_address_code_generated_temp", "arq_three_address_code_generated");
38 arq_three_address_code = fopen("arq_three_address_code_generated", "a+"); // abrindo para append para continuar o
39 processo
40 }

```

**Figura 5.3.3:** Função responsável por realizar o *backpatch*. Um arquivo temporário é gerado e as linhas correspondentes aos *backpatches* são alteradas. Código adaptado de [6].

Uma estratégia de otimização de expressões aritméticas é mostrada na Fig. 5.3.4. Ela foi implementada para reduzir as expressões aritméticas com valores constantes operando entre si. Por exemplo, uma expressão como  $a = 5 * (3 + 4 / 2)$  seria reduzida para apenas  $a = 25$ . Note que essa otimização reduziria o número de linhas do código intermediário gerado. Esse tipo de otimização foi comentado em sala de aula e implementamos numa primeira versão da geração de código. Este código funciona, mas com o advento das variáveis, decidimos por não usá-lo. Desta forma, nesta versão final de entrega deste trabalho, apenas o esforço da criação da função está sendo documentado.

```

1 Symbol* applyBinaryOperatorInSymbols(Symbol *s1, char op, Symbol *s2){
2     if(s1->data != NULL && s2->data != NULL){
3         if(s1->data->type == _INTEGER && s2->data->type == _INTEGER){
4             int a, b;
5             a = s1->data->v.integer;
6             b = s2->data->v.integer;
7
8             switch(op){
9                 case '+': a = a + b; break;
10                case '-': a = a - b; break;
11                case '/': if(b == 0) return NULL; a = a / b; break;
12                case '*': a = a * b; break;
13                case '%': a = a % b; break;
14                case '^': a = pow(a, b); break;
15            }
16            s1->data->v.integer = a;
17            return s1;
18        }
19        if(s1->data->type == _INTEGER && s2->data->type == _REAL){
20            int a;
21            double b;
22            a = s1->data->v.integer;
23            b = s2->data->v.real;

```

```

24
25     switch(op){
26         case '+': b = a + b; break;
27         case '-': b = a - b; break;
28         case '/': if(b == 0) return NULL; b = (double) a / b; break;
29         case '*': b = a * b; break;
30         case '%': return NULL; break;
31         case '^': b = pow(a, b); break;
32     }
33     s1->data->type = _REAL;
34     s1->data->v.real = b;
35     return s1;
36 }
37 if(s1->data->type == _REAL && s2->data->type == _INTEGER){
38     double a;
39     int b;
40     a = s1->data->v.real;
41     b = s2->data->v.integer;
42
43     switch(op){
44         case '+': a = a + b; break;
45         case '-': a = a - b; break;
46         case '/': if(b == 0) return NULL; a = (double) a / b; break;
47         case '*': a = a * b; break;
48         case '%': return NULL; break;
49         case '^': a = pow(a, b); break;
50     }
51     s1->data->v.real = a;
52     return s1;
53 }
54 if(s1->data->type == _REAL && s2->data->type == _REAL){
55     double a, b;
56     a = s1->data->v.real;
57     b = s2->data->v.real;
58
59     switch(op){
60         case '+': a = a + b; break;
61         case '-': a = a - b; break;
62         case '/': if(b == 0) return NULL; a = (double) a / b; break;
63         case '*': a = a * b; break;
64         case '%': return NULL; break;
65         case '^': a = pow(a, b); break;
66     }
67     s1->data->v.real = a;
68     return s1;
69 }
70 return NULL;
71 }
72 }

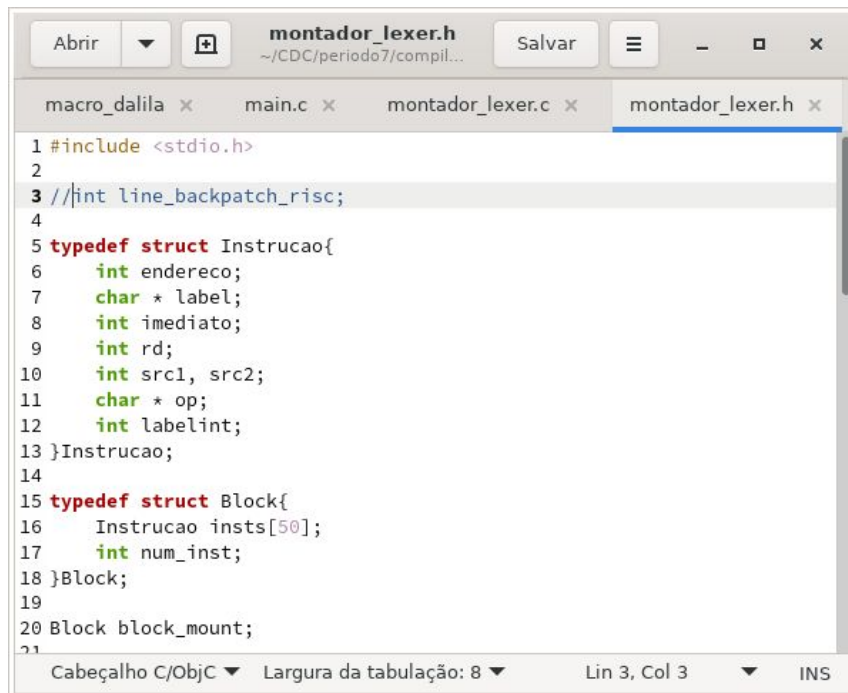
```

**Figura 5.3.4:** Estratégia para otimizar operações aritméticas. Funcionava apenas com valores numéricos em uma dada operação aritmética. Não usado na versão final.

## 6 - Geração de Código de máquina (RISC, LLVM)

Com o intuito de propor um back-end para o compilador, foram estudadas as gerações de código de máquina para o conjunto de instruções RISC-V[7] e buscamos um meio de simular as operações aritméticas usadas no código usando a IR do LLVM.

Como auxiliar deste processo, temos estruturas comuns em montadores: instrução, bloco de instruções. Elas são definidas como mostra a imagem:



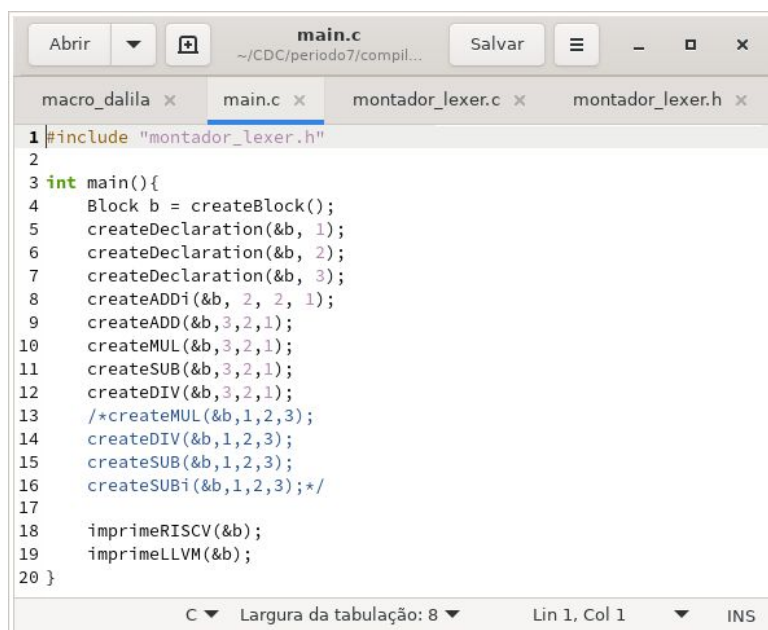
```

1 #include <stdio.h>
2
3 //int line_backpatch_riscv;
4
5 typedef struct Instrucao{
6     int endereco;
7     char * label;
8     int imediato;
9     int rd;
10    int src1, src2;
11    char * op;
12    int labelint;
13 }Instrucao;
14
15 typedef struct Block{
16     Instrucao insts[50];
17     int num_inst;
18 }Block;
19
20 Block block_mount;
21

```

**Figura 6.1:** Exemplo de conjunto de instruções RISC-V.

Para fazer uso desse módulo basta chamar a função create de cada operador desejado. Sendo que além dos operadores convencionais, possuímos create para declaração o qual é responsável por alocar memória no arquivo gerado para LLVM.



```

1 #include "montador_lexer.h"
2
3 int main(){
4     Block b = createBlock();
5     createDeclaration(&b, 1);
6     createDeclaration(&b, 2);
7     createDeclaration(&b, 3);
8     createADDi(&b, 2, 2, 1);
9     createADD(&b, 3, 2, 1);
10    createMUL(&b, 3, 2, 1);
11    createSUB(&b, 3, 2, 1);
12    createDIV(&b, 3, 2, 1);
13    /*createMUL(&b, 1, 2, 3);
14    createDIV(&b, 1, 2, 3);
15    createSUB(&b, 1, 2, 3);
16    createSUBi(&b, 1, 2, 3);*/
17
18    imprimeRISCV(&b);
19    imprimeLLVM(&b);
20 }

```

**Figura 6.2:** Exemplo de conjunto de instruções RISC-V.

## 6.1 - RISC

Para gerar código RISC apenas seguimos a especificação[7]. No entanto, não garantimos que o código gerado seja simulável, visto que não tentamos fazer isso. O objetivo dessa etapa era apenas aproximar o código intermediário produzido de um código que realmente pode ser executado em uma arquitetura real. A tabela 6.1.1 apresenta o conjunto de instruções RISC implementado junto ao montador:

A = B + C	add rd, rs1, rs2
A = B - C	add rd, rs1, -rs2
A = B * C	mul rd, rs1, rs2
A = B / C	div rd, rs1, rs2
A = B and C	and rd, rs1, rs2
A = B or C	or rd, rs1, rs2
A = !B	xor rd, rs1, 0
A == B	and rd, a, b
A != B	xor B, b, 0 and rd, a, B
A < B	slt rd, rs1, rs2
A > B	sgt rd, rs2, rs1
If False/True t1 goto L	beqi t0, boolean(0 ou 1), L
label: (label literal)	label:
3: (label numérico)	3:

**Tabela 6.1.1:** conjunto de instruções RISC usadas.

Assim, a única instrução planejada para RISC mas não implementada foi o operador diferente. Optamos por não a implementar porque dada a ausência do operador NOT em RISCV, implementar diferente demandaria um XOR e um AND. E ter duas instruções reais correspondendo a uma instrução do código intermediário traria problemas na numeração de registradores e de linhas, além de possíveis problemas com labels numéricos que contém numeração de linha. Há outras instruções RISC não

implementadas, no entanto, temos um conjunto mínimo razoável para produzir códigos relevantes.

<b>Código</b>	
1	begin
2	integer num_1
3	integer num_2
4	integer resposta
5	num_1 = 8
6	num_2 = num_1 - 5 * 3
7	num_2 = 7
8	//num_1 = 8
9	num_1 = num_1 + 9\\
10	
11	
12	resposta = (num_1 + num_1 * num_2 + num_2 * 5 * 4 * 3 - 5 * 3) / 67
13	
14	end

<b>Código de três endereços</b>	<b>Código ASM (RISC V)</b>
t3 = 8	ADDi t4, t4, 8
t0 = t3	ADDi t1, t4, 8
t4 = 5	ADDi t5, t5, 5
t5 = 3	ADDi t6, t6, 3
t6 = t4 * t5	MUL t7, t5, t6
t7 = t0 - t6	ADD t8, t1, -t7
t1 = t7	ADDi t2, t8, 0
t8 = 7	ADDi t9, t9, 7
t1 = t8	ADDi t2, t9, 7
t9 = t0 * t1	MUL t10, t1, t2
t10 = t0 + t9	ADD t11, t1, t10
t11 = 5	ADDi t12, t12, 5
t12 = t1 * t11	MUL t13, t2, t12
t13 = 4	ADDi t14, t14, 4
t14 = t12 * t13	MUL t15, t13, t14
t15 = 3	ADDi t16, t16, 3
t16 = t14 * t15	MUL t17, t15, t16
t17 = t10 + t16	ADD t18, t11, t17
t18 = 5	ADDi t19, t19, 5
t19 = 3	ADDi t20, t20, 3
t20 = t18 * t19	MUL t21, t19, t20
t21 = t17 - t20	ADD t22, t18, -t21
t22 = 67	ADDi t23, t23, 67
t23 = t21 / t22	DIV t24, t22, t23
t2 = t23	ADDi t3, t24, 0

**Figura 6.1.1:** Exemplo de conjunto de instruções RISC-V.

## 6.2 - LLVM

Utilizamos a IR do LLVM para simular operações aritméticas, foram criadas apenas as instruções: load, store, add, mul, div e sub. E suas versões que comportam valores imediatos, as constantes do código original. Foi uma tarefa consideravelmente complicada transpor nosso conjunto de instruções para a IR da LLVM, por dois motivos: o novo conjunto deveria ser funcional, para ser usado em simulação; e também a forma como é preciso definir IRs. Para fazer um conjunto de instruções que levam a resultados coerentes usando a IR do LLVM precisamos:

- a cada leitura de variável: fazer load da memória usando essa instrução;
- a cada armazenamento em variável: fazer store na memória usando essa instrução;
- para todo registrador temporário ou não: deve-se alocar memória para ele;

Além disso, todo o código LLVM deve ser definido dentro de alguma função, seja ela chamada pelo método principal, ou o próprio método principal. Aproveitando essa organização do LLVM, sempre que realizamos a operação de alocar em seguida inicializamos a variável com 0, para evitar acessos à memória com lixo.

Sendo assim, além do código intermediário, da árvore sintática abstrata gerada implicitamente mas não impressa, também retornamos arquivos com linguagem de máquina RISC e instruções do código intermediário “.ll” da llvm. Os passos abaixo mostram o script utilizado, partindo do código “.ll” rumo a simulação em máquina real:

```
llvm-as cod.ll -o cod.bc
```

```
llc cod.bc -o cod.s
```

```
clang-10 cod.s -o cod.native
```

```
./cod.native
```

Para realizar essa tarefa, partimos de um código mínimo e geramos o arquivo “.ll” para ele usando o clang, a figura a seguir o apresenta. Conforme necessário foram



adicionadas mais instruções e também uma chamada de sistema para imprimir o resultado final de alguma operação desejada.

```

1
2 target triple = "x86_64-unknown-linux-gnu"
3
4 ; Function Attrs: noinline nounwind optnone uwtable
5 define dso_local @main() #0 {
6     %1 = alloca i32, align 4
7     ret i32 0
8 }
9
10 attributes #0 = { noinline nounwind optnone uwtable "correctly-
    rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
    "frame-pointer"="all" "less-precise-fpmad"="false" "min-legal-vector-
    width"="0" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
    nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
    math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
    "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-
    math"="false" "use-soft-float"="false" }
11
12 !llvm.module.flags = !{!0}
13 !llvm.ident = !{!1}
14
15 !0 = !{i32 1, !"wchar_size", i32 4}
16 !1 = !"clang version 11.0.0 (Fedora 11.0.0-2.fc33)"
  
```

**Figura 6:2.1:** Código mínimo do LLVM.

Códigos adicionados para a impressão:

- **instrução:** `%44 = call i32 @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 %43)`
- **definição de string:** `@.str = private unnamed_addr constant [3 x i8] c"%d\0A", align 1`
- **importar chamada de sistema:** declare `dso_local i32 @printf(i8*, ...) #1`
- **attributes #1 = {** "correctly-rounded-divide-sqrt-fp-math"="false"  
 "disable-tail-calls"="false" "frame-pointer"="all" "less-precise-fpmad"="false"  
 "no-infs-fp-math"="false" "no-nans-fp-math"="false"  
 "no-signed-zeros-fp-math"="false" "no-trapping-math"="true"  
 "stack-protector-buffer-size"="8" "target-cpu"="x86-64"  
 "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87"  
 "unsafe-fp-math"="false" "use-soft-float"="false" }

O exemplo a seguir apresenta instruções para uma calculadora, mostrada na Figura 6.1.1, feita em nossa linguagem e traduzida para IR da LLVM após o processo de geração de código intermediário.

```

1  target triple = "x86_64-unknown-linux-gnu"
2
3  @.str = private unnamed_addr constant [3 x i8] c"%d\0A", align 1
4
5  define dso_local i32 @main() #0 { %1 = alloca i32, align 4
6      store i32 0, i32* %1, align 4
7      %2 = alloca i32, align 4
8      store i32 0, i32* %2, align 4
9      %3 = alloca i32, align 4
10     store i32 0, i32* %3, align 4
11     %4 = alloca i32, align 4
12     store i32 0, i32* %4, align 4
13     %5 = alloca i32, align 4
14     store i32 0, i32* %5, align 4
15     %6 = load i32, i32* %5, align 4
16     %7 = add nsw i32 %6, 8
17     store i32 8, i32* %5, align 4
18     %8 = load i32, i32* %5, align 4
19     %9 = add nsw i32 %8, 8
20     store i32 8, i32* %2, align 4
21     %10 = alloca i32, align 4
22     store i32 0, i32* %10, align 4
23     %11 = load i32, i32* %10, align 4
24     %12 = add nsw i32 %11, 5
25     store i32 5, i32* %10, align 4
26     %13 = alloca i32, align 4
27     store i32 0, i32* %13, align 4
28     %14 = load i32, i32* %13, align 4
29     %15 = add nsw i32 %14, 3
30     store i32 3, i32* %13, align 4
31     %16 = alloca i32, align 4
32     store i32 0, i32* %16, align 4
33     %17 = load i32, i32* %10, align 4
34     %18 = load i32, i32* %13, align 4
35     %19 = mul nsw i32 %18, %17
36     store i32 %19, i32* %16, align 4
37     %20 = alloca i32, align 4
38     store i32 0, i32* %20, align 4
39     %21 = load i32, i32* %2, align 4
40     %22 = load i32, i32* %16, align 4
41     %23 = sub nsw i32 %21, %22
42     store i32 %23, i32* %20, align 4
43     %24 = load i32, i32* %20, align 4
44     %25 = add nsw i32 %24, 0
45     store i32 0, i32* %3, align 4
46     %26 = alloca i32, align 4
47     store i32 0, i32* %26, align 4
48     %27 = load i32, i32* %26, align 4
49     %28 = add nsw i32 %27, 7
50     store i32 7, i32* %26, align 4
51     %29 = load i32, i32* %26, align 4
52     %30 = add nsw i32 %29, 7
53     store i32 7, i32* %3, align 4
54     %31 = alloca i32, align 4
55     store i32 0, i32* %31, align 4
56     %32 = load i32, i32* %2, align 4
57     %33 = load i32, i32* %3, align 4
58     %34 = mul nsw i32 %33, %32
59     store i32 %34, i32* %31, align 4
60     %35 = alloca i32, align 4

```

```

61 store i32 0, i32* %35, align 4
62 %36 = load i32, i32* %2, align 4
63 %37 = load i32, i32* %31, align 4
64 %38 = add nsw i32 %37, %36
65 store i32 %38, i32* %35, align 4
66 %39 = alloca i32, align 4
67 store i32 0, i32* %39, align 4
68 %40 = load i32, i32* %39, align 4
69 %41 = add nsw i32 %40, 5
70 store i32 5, i32* %39, align 4
71 %42 = alloca i32, align 4
72 store i32 0, i32* %42, align 4
73 %43 = load i32, i32* %3, align 4
74 %44 = load i32, i32* %39, align 4
75 %45 = mul nsw i32 %44, %43
76 store i32 %45, i32* %42, align 4
77 %46 = alloca i32, align 4
78 store i32 0, i32* %46, align 4
79 %47 = load i32, i32* %46, align 4
80 %48 = add nsw i32 %47, 4
81 store i32 4, i32* %46, align 4
82 %49 = alloca i32, align 4
83 store i32 0, i32* %49, align 4
84 %50 = load i32, i32* %42, align 4
85 %51 = load i32, i32* %46, align 4
86 %52 = mul nsw i32 %51, %50
87 store i32 %52, i32* %49, align 4
88 %53 = alloca i32, align 4
89 store i32 0, i32* %53, align 4
90 %54 = load i32, i32* %53, align 4
91 %55 = add nsw i32 %54, 3
92 store i32 3, i32* %53, align 4
93 %56 = alloca i32, align 4
94 store i32 0, i32* %56, align 4
95 %57 = load i32, i32* %49, align 4
96 %58 = load i32, i32* %53, align 4
97 %59 = mul nsw i32 %58, %57
98 store i32 %59, i32* %56, align 4
99 %60 = alloca i32, align 4
100 store i32 0, i32* %60, align 4
101 %61 = load i32, i32* %35, align 4
102 %62 = load i32, i32* %56, align 4
103 %63 = add nsw i32 %62, %61
104 store i32 %63, i32* %60, align 4
105 %64 = alloca i32, align 4
106 store i32 0, i32* %64, align 4
107 %65 = load i32, i32* %64, align 4
108 %66 = add nsw i32 %65, 5
109 store i32 5, i32* %64, align 4
110 %67 = alloca i32, align 4
111 store i32 0, i32* %67, align 4
112 %68 = load i32, i32* %67, align 4
113 %69 = add nsw i32 %68, 3
114 store i32 3, i32* %67, align 4
115 %70 = alloca i32, align 4
116 store i32 0, i32* %70, align 4
117 %71 = load i32, i32* %64, align 4
118 %72 = load i32, i32* %67, align 4
119 %73 = mul nsw i32 %72, %71
120 store i32 %73, i32* %70, align 4
121 %74 = alloca i32, align 4
122 store i32 0, i32* %74, align 4
123 %75 = load i32, i32* %60, align 4
124 %76 = load i32, i32* %70, align 4
125 %77 = sub nsw i32 %75, %76
126 store i32 %77, i32* %74, align 4
127 %78 = alloca i32, align 4
128 store i32 0, i32* %78, align 4
129 %79 = load i32, i32* %78, align 4

```

```

130 %80 = add nsw i32 %79, 67
131 store i32 67, i32* %78, align 4
132 %81 = alloca i32, align 4
133 store i32 0, i32* %81, align 4
134 %82 = load i32, i32* %74, align 4
135 %83 = load i32, i32* %78, align 4
136 %84 = sdiv i32 %82, %83
137 store i32 %84, i32* %81, align 4
138 %85 = load i32, i32* %81, align 4
139 %86 = add nsw i32 %85, 0
140 store i32 0, i32* %4, align 4
141 %87 = call i32 @i8*, ... @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 %86)
142 ret i32 0
143 }
144
145 declare dso_local i32 @printf(i8*, ...) #1
146
147
148 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
149 "disable-tail-calls"="false" "frame-pointer"="all" "less-precise-fpmad"="false" "min-legal-vector-width"="0"
150 "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
151 "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
152 "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
153
154
155 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "frame-pointer"="all"
156 "less-precise-fpmad"="false" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
157 "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
158 "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
159
160 !llvm.module.flags = !{!0}
161 !llvm.ident = !{!1}
162
163 !0 = !{i32 1, !"wchar_size", i32 4}
164 !1 = !{!"clang version 11.0.0 (Fedora 11.0.0-2.fc33)"}

```

Figura 6.2.2: Exemplo de arquivo com IR da LLVM para simulação.

## 7 - Programa de Teste Geral (Código de Três Endereços)

Nesta seção iremos apresentar um exemplo mais geral de código em paralelo com o seu respectivo código de três endereços. Vale ressaltar que na seção 5.2.4 apresentamos, dentre outros, os erros que o nosso algoritmo detecta e seus respectivos exemplos. Portanto, o objetivo do exemplo abaixo é demonstrar algumas funcionalidades operantes em nosso algoritmo, como: a detecção de declaração de variáveis mais externas por parte dos escopos mais internos e a técnica *jump to*. Note que ela está baseada em *label*, enquanto que os *backpatches* dos comandos *if* e *while* estão sendo tratados. Decidimos manter dessa forma por motivos de tratamento de *backpatches* envolvendo a estratégia *jump to* serem mais complicadas. Ainda, existem até mesmo códigos de máquina que permitem o uso de *labels* para os desvios incondicionais, como por exemplo MIPS.

Código	Código de três endereços
1 begin	1 t2 = nome
2 real numeroreal	2 t3 = 123.545000
3 integer numerointeiro	3 t0 = t3
4 word string	4 t4 = 0
5 string = "nome"	5 t1 = t4
6 begin	6 t5 = sobrenome
7 numeroreal = 123.545	7 t6 = Carlinhos Silva
8 numerointeiro = 0	8 t8 = 1
9 word stringdois	9 t7 = t8
10 stringdois= " sobrenome"	10 t9 = 1
11 begin	11 t10 = t7 == t9
12 word concatenacao	12 iffFalse t10 goto 14
13 concatenacao = "Carlinhos " ++ "Silva"	13 goto aqui
14 say string	14 t13 = 0
15 say stringdois	15 t12 = t13
16 end	16 aqui:
17 end	17 t14 = 5
18 integer flagjump	18 t15 = t1 < t14
19 flagjump = 1	19 iffFalse t15 goto 24
20 if flagjump == 1:	20 t16 = 1
21   jumpto aqui	21 t17 = t1 + t16
22 endif	22 t1 = t17
23 integer passouAqui	23 goto 18
24 passouAqui = 0	
25 aqui:	
26 while numerointeiro < 5:	
27   numerointeiro = numerointeiro + 1	
28 endwhile	
29 end	

**Figura 7.1:** Exemplo de código na linguagem Chameleon em paralelo a sua equivalência gerada em código de três endereços.

## 8 - Considerações Finais

Neste trabalho foram desenvolvidas as etapas de Análise Semântica e Geração de Código Intermediário do compilador para a linguagem Chameleon. Além disso, foram geradas representações para códigos LLVM e RISC-V. Algumas alterações na gramática foram necessárias novamente para facilitar processos de análise semântica e geração de código intermediário. Uma dessas facilidades veio de um recurso próprio do YACC para permitir estabelecer ordem de precedência dos operadores, gerando o comportamento esperado para expressões aritméticas, por exemplo. Outro recurso aproveitado foi a geração de código após operações de *reduce* do YACC, o que permitiu criar instruções condicionais e de repetição.

Ao longo de todas essas etapas do trabalho, conseguimos aplicar na prática o conhecimento adquirido durante as aulas. Apesar de nesta versão final nem todas as instruções terem sido contempladas no processo de geração de código intermediário e análise semântica, acreditamos que conseguiríamos, com mais tempo, cumprir todos os requisitos propostos para a linguagem em sua especificação. Além disso, não conseguimos melhorar o pré-processador devido ao tempo e a dificuldade desta última etapa. Em particular, decidimos por tentar conseguir fazer até mesmo a geração de código para uma máquina com o uso do LLVM, e isso foi um sucesso. Portanto, conseguimos montar um compilador que vai desde o *front-end* até o *back-end* e o executável. Essa experiência mostrou o quão complexo é o processo de construção de um compilador e como isso pode ser facilitado pelo uso das ferramentas geradoras de analisadores léxicos (LEX) e sintáticos (YACC).

## Referências Bibliográficas

[1] AHO, A.V.; LAM, M.S.; SETHI, R.; ULLMAN, J.D. Compiladores: Princípios, técnicas e ferramentas. Segunda Edição. Pearson Addison-Wesley, 2008.

[2] Precedence. Disponível em:

<https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dh3/index.html>

[3] Address Code Generation using lex and yacc. Disponível em:

<https://stackoverflow.com/questions/35891282/3-address-code-generation-using-lex-and-yacc>

[4] Code Generation. Disponível em:

<https://book.huihoo.com/compiler-construction-using-flex-and-bison/CodeGen.html>

[5] How to add precedence to LALR parser like in YACC? Disponível em:

<https://softwareengineering.stackexchange.com/questions/178187/how-to-add-precedence-to-lalr-parser-like-in-yacc>

[6] C program to replace specific line in a text file. Disponível em:

<https://codeforwin.org/2018/02/c-program-replace-specific-line-a-text-file.html>

[7] RISC-V "V" Vector Extension. Disponível em:

<https://riscv.github.io/documents/riscv-v-spec/riscv-v-spec.pdf>

## Apêndice A - Código *lex.l* para a linguagem Chameleon

```

1  %{
2  #include "custom_defines.h"
3  #include "y.tab.h" /* para poder usar as constantes dos tokens, por exemplo. */
4
5  #define PRINT_ERROR_EOF "-> Um ou mais erros foram encontrados. Corrija-os!\n"
6
7
8  void remover_espacos_e_print(int t);
9  #define _REAL 1
10 #define _NUMBER 2
11
12 int supress_errors_flag = 0;
13 int erro_encontrado = 0;
14
15 %{
16 /* condicao exclusiva (bloqueia as demais regras) */
17 %x comment_condition
18 %x word_condition
19 /* condicao que e ativa mas mantem as demais ativadas tambem */
20 %s supress_errors_condition
21 /* Permitir a contabilizacao de linhas */
22 %option yylineno
23
24
25 supress_errors      "SUPRESS_ERRORS"
26
27 /* captura uma ocorrencia de espaco em branco, tabulacao ou quebra de linha*/
28 delim              [ \t\n\r]
29 /* ign (ignorador) ira ignorar um ou mais delim*/
30 ign                 {delim}+
31 letter              [A-Za-z]
32 digit               [0-9]
33 underline           _
34 word_value          (\.[\^[^\\])*
35 number              {ign}*({digit}{ign})*+
36 type                "integer"|"word"|"real"
37
38
39
40 squad_declaration   "squad"
41 vector_declaration  "vector"
42
43
44 end_squad           "endsquad"
45 block_begin         "begin"
46 block_end           "end"
47
48 for                 "for"
49 end_for             "endfor"
50 while               "while"
51 end_while           "endwhile"
52 if                  "if"
53 end_if              "endif"
54 elif                "elif"
55 end_elif            "endelif"
56 task                "task"
57 end_task            "endtask"
58
59
60
61 jumpto              "jumpto"
62 farewell            "farewell"
63 say                 "say"
64 listen              "listen"

```



```

65 stop "stop"
66 comma ","
67 open_parenthesis "("
68 close_parenthesis ")"
69
70 identifier_complement ({digit}|{letter}|{underline})*
71 identifier ({letter}|{underline}{letter}){identifier_complement}
72
73
74 vector_access_start "["
75 vector_access_end "]"
76
77
78 squad_access_derreference "->"
79 separator ":"
80 real_number {ign}*( {digit} {ign} *)+{ign}*(\.{ign})*( {digit} {ign} *)+
81 word_concat_operator "++"
82 add_operator [+ -]
83 div_operator [/ * %]
84 pow_operator [^]
85 rel_operator ["="|"!="|">="|"<="|">|<"]
86 logic_operator ["and"|"or"|"!"]
87 attribution "="
88 credits "??credits??"
89
90
91 %%
92 {supress_errors} { BEGIN(supress_errors_condition); supress_errors_flag = 1; }
93 "giveup" {return GIVEUP;}
94 "/" BEGIN(comment_condition);
95 {ign} {}
96
97 <comment_condition>.\n {}
98 <comment_condition>"\\\\" { BEGIN(INITIAL); if(supress_errors_flag) BEGIN(supress_errors_condition); }
99
100 "\\" BEGIN(word_condition);
101 <word_condition>{word_value} {
102     createSymbol(&(yyval.symbol), NULL, _WORD, _WORD);
103     (yyval.symbol)->data->v.word = (char*) malloc(MAX_TAM_WORD * sizeof(char));
104     sprintf((yyval.symbol)->data->v.word, "%s", yytext);
105     return WORD_VALUE;
106 }
107 <word_condition>"\"" {BEGIN(INITIAL); if(supress_errors_flag) BEGIN(supress_errors_condition); }
108
109 {credits} {PRINT(("Feito por:\n%s\n",
110     "Daniel Freitas Martins - 2304\n"
111     "João Arthur Gonçalves do Vale - 3025\n"
112     "Maria Dalila Vieira - 3030\n"
113     "Naiara Cristiane dos Reis Diniz - 3005"))}
114
115 {type} {
116     if(strcmp("integer", yytext) == 0)
117         return INTEGER;
118     if(strcmp("real", yytext) == 0)
119         return REAL;
120     return WORD;
121 }
122 {squad_declaration} {strcpy(yyval.type_aux, "squad"); return SQUAD;}
123 {vector_declaration} {strcpy(yyval.type_aux, "vector"); return VECTOR;}
124
125 {end_squad} {return ENDSQUAD;}
126 {block_begin} {
127
128
129
130
131
132
133

```

```

134         yylval.symbol_table = ht_create(MAX_TAM_HASH);
135         if(curr_symbol_table == NULL){
136             curr_symbol_table = yylval.symbol_table;
137             if(first_symbol_table == NULL){
138                 first_symbol_table = yylval.symbol_table;
139             }
140         }
141     }
142     return BLOCK_BEGIN;
143 {block_end}     {return BLOCK_END;}
144
145 {for}           {return FOR;}
146 {end_for}       {return ENDFOR;}
147 {while}         {return WHILE;}
148 {end_while}     {return ENDWHILE;}
149 {if}            {return IF;}
150 {end_if}        {return ENDIF;}
151 {elif}          {return ELIF;}
152 {end_elif}      {return ENDELIF;}
153 {task}          {strcpy(yylval.type_aux, "task"); return TASK;}
154 {end_task}      {return ENDTASK;}
155
156 {jump_to}       {return JUMPTO;}
157 {farewell}      {return FAREWELL;}
158 {say}           {return SAY;}
159 {listen}        {return LISTEN;}
160 {stop}          {return STOP;}
161
162 {comma}         {return yytext[0];}
163 {open_parenthesis} {return yytext[0];}
164 {close_parenthesis} {return yytext[0];}
165
166 {vector_access_start} {return yytext[0];}
167 {vector_access_end}   {return yytext[0];}
168
169
170
171
172
173
174 {squad_access_derreference} {return SQUAD_ACCESS_DERREFERENCE;}
175 {separator}                 {return yytext[0];}
176 {word_concat_operator}      {return WORD_CONCAT_OPERATOR;}
177 {add_operator}              {yylval.op = yytext[0]; return ADD_OPERATOR;}
178 {div_operator}              {yylval.op = yytext[0]; return DIV_OPERATOR;}
179 {pow_operator}              {yylval.op = yytext[0]; return POW_OPERATOR;}
180 {rel_operator}              {
181     if(strcmp(yytext, "==") == 0) yylval.op = EQ;
182     else if(strcmp(yytext, "!=") == 0) yylval.op = NE;
183     else if(strcmp(yytext, ">=") == 0) yylval.op = GE;
184     else if(strcmp(yytext, "<=") == 0) yylval.op = LE;
185     else if(strcmp(yytext, ">") == 0) yylval.op = GT;
186     else if(strcmp(yytext, "<") == 0) yylval.op = LT;
187     return REL_OPERATOR;
188 }
189
190 {logic_operator}           {
191     if(yytext[0] == '!'){
192         yylval.op = '!';
193         return NEG_OPERATOR;
194     }
195     if(strcmp(yytext, "and") == 0) yylval.op = AND;
196     else yylval.op = OR;
197     return LOGIC_OPERATOR;
198 }
199
200
201
202

```

```

203 {attribution}          {return yytext[0];}
204
205 {real_number}          {remover_espacos_e_print(_REAL);
206                          createSymbol(&(yyval.symbol), NULL, _REAL, _REAL);
207                          (yyval.symbol)->data->v.real = atof(yytext);
208                          return REAL_NUMBER;}
209
210 {number}                {remover_espacos_e_print(_NUMBER);
211                          createSymbol(&(yyval.symbol), NULL, _INTEGER, _INTEGER);
212                          (yyval.symbol)->data->v.integer = atoi(yytext);
213                          return NUMBER;}
214
215
216 {identifier}            { createSymbol(&(yyval.symbol), yytext, NOTHING, NOTHING);
217                          // (yyval.symbol)->data->v.integer = atoi(yytext);
218                          return IDENTIFIER;}
219
220 <supress_errors_condition>. {}
221 ","+                    {} /* ignorando ponto e virgula */
222 .                        {PRINT((PRINT_ERROR "> %s\nint_code_s0: %d\n", yylineno, yytext, yytext[0])) erro_encontrado = 1;}
223 /* Ignorar o que nao foi definido */
224 <<EOF>>                 {
225
226                         if(erro_encontrado){
227                             PRINT((PRINT_ERROR_EOF))
228                             exit(1);
229                         }
230                         return 0;
231                     }
232
233
234 %%
235 void remover_espacos_e_print(int t){
236     char* s; /* tera a nova string sem os espacos em branco */
237     int i, j, tam_yytext = strlen(yytext);
238     s = (char*) malloc(tam_yytext*sizeof(char));
239     j = 0;
240     for(i = 0; i < tam_yytext; i++){
241         if(yytext[i] == ' ' || yytext[i] == '\n' || yytext[i] == '\t')
242             continue;
243         s[j++] = yytext[i];
244     }
245     s[j] = '\0';
246
247     strcpy(yytext, s);
248
249     switch(t){
250     case _REAL:
251         //PRINT((PRINT_REAL_NUMBER PRINT_LEXEME, yylineno, s))
252         break;
253     case _NUMBER:
254         //PRINT((PRINT_NUMBER PRINT_LEXEME, yylineno, s))
255         break;
256     }
257     free(s);
258 }
259
260 int yywrap(){ return 1; } /* se EOF for encontrado, encerre. */
261
262
263

```

## Apêndice B - Código *translate.y* para a linguagem Chameleon

```

1  %{
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      #include "custom_defines.h"
6
7
8      extern int yylineno;
9      extern FILE* yyin;
10     int yylex();
11     int yyerror(const char*);
12     void raiseErrorVariableRedeclaration(char *lexem);
13     void raiseError(char *msg);
14     void emit(char *msg);
15
16
17     short flag_block_continue = 0;
18     int num_prox_instr = 0;
19     FILE *arq_three_address_code;
20     int id_temporario;
21     int *id_temporario;
22     char type_names[][20] = {"???", "REAL", "FUNCTION", "INTEGER", "WORD", "SQUAD", "VARIABLE"};
23     char rel_operators_str[][3] = {"==", "!=", ">=", "<=", ">", "<"};
24     int line_backpatch_for_aux, line_backpatch_for_aux2;
25 }
26
27 %code requires {
28     #include "hash-table.h"
29     #include "montador_lexer.h"
30     void action_math_expression(Symbol *sd, Symbol *s1, char op, Symbol *s2);
31     void action_math_expression_logic(Symbol *sd, Symbol *s1, char op, Symbol *s2);
32     void action_math_expression_rel(Symbol *sd, Symbol *s1, char op, Symbol *s2);
33     void action_math_expression_unary(Symbol *sd, char op, Symbol *s1);
34     void emitBackpatch(int linha_origem, int linha_destino);
35 }
36
37 %union{
38     Symbol *symbol;
39     hashtable_t *symbol_table;
40     char type_aux[20]; // apenas para print...
41     char op;
42     int type_declaration;
43     int id_temporario;
44 }
45
46 %define parse.error verbose
47 %define parse.lac full
48 %token WORD_VALUE
49 %token NUMBER_REAL_NUMBER
50 %token SQUAD_ACCESS_DERREFERENCE
51 %token INTEGER_WORD_REAL
52 %token SQUAD_ENDSQUAD
53 %token VECTOR
54 %token IDENTIFIER
55 %token ADD_OPERATOR DIV_OPERATOR POW_OPERATOR LOGIC_OPERATOR
56 %token NEG_OPERATOR REL_OPERATOR WORD_CONCAT_OPERATOR
57 %token GIVEUP
58 %token BLOCK_BEGIN BLOCK_END
59 %token SAY_LISTEN
60 %token IF_ENDIF_ELIF_ENDELIF
61 %token FOR_ENDFOR
62 %token WHILE_ENDWHILE
63 %token TASK_ENDTASK
64 %token FAREWELL
65 %token JUMPTO

```

```

65 %token STOP
66
67 %right '='
68 %left ADD_OPERATOR NEG_OPERATOR
69 %left DIV_OPERATOR
70 %left POW_OPERATOR
71 %left LOGIC_OPERATOR
72 %left REL_OPERATOR
73 %left '('
74
75 %start block
76 %%
77
78 block: BLOCK_BEGIN {
79     if(flag_block_continue){
80         flag_block_continue = 0;
81         if(curr_symbol_table->brother_hash == NULL){
82             hashtable_t *brother_symbol_table = $<symbol_table>1;
83             curr_symbol_table->brother_hash = brother_symbol_table;
84             curr_symbol_table = brother_symbol_table;
85         } else{
86             raiseError("O Brother e nao nulo!");
87         }
88     }
89 } statement BLOCK_END block_continue {PRINT(("block -> [BLOCK_BEGIN statement BLOCK_END
90 block_continue]\n"))}
91 | task_command block_continue {PRINT(("block -> [task_command block_continue]\n"))}
92 ;
93
94
95 block_continue: /* palavra vazia */ {PRINT(("block_continue -> []\n"))}
96 | { flag_block_continue = 1; } block {PRINT(("block_continue -> [block]\n"))}
97 ;
98
99 statement: /* palavra vazia */ {PRINT(("statement -> []\n"))}
100 | command statement {PRINT(("statement -> [command statement]\n"))}
101 | variable_declaration statement {PRINT(("statement -> [variable_declaration statement]\n"))}
102 | BLOCK_BEGIN {
103     if(curr_symbol_table->child_hash != NULL){
104         while(curr_symbol_table->child_hash != NULL){
105             curr_symbol_table = curr_symbol_table->child_hash;
106         }
107         hashtable_t *brother_symbol_table = $<symbol_table>1;
108         curr_symbol_table->brother_hash = brother_symbol_table;
109         brother_symbol_table->previous_hash = curr_symbol_table->previous_hash;
110         curr_symbol_table = brother_symbol_table;
111     } else{
112         hashtable_t *child_symbol_table = $<symbol_table>1;
113         child_symbol_table->previous_hash = curr_symbol_table;
114         curr_symbol_table->child_hash = child_symbol_table;
115         curr_symbol_table = child_symbol_table;
116     }
117 } statement BLOCK_END {
118     if(curr_symbol_table->previous_hash != NULL)
119         curr_symbol_table = curr_symbol_table->previous_hash;
120 } statement {PRINT(("statement -> [BLOCK_BEGIN statement BLOCK_END statement]\n"))}
121 ;
122
123 command: variable '=' expression {PRINT(("command -> [variable '=' expression]\n"))
124 //printf("--> %s %d\n", $<symbol>1->lexem, $<symbol>1->fake_memory_address);
125 if($<symbol>1 == NULL)
126     raiseError("Variable is not declared!");
127 if(!checkTypes($<symbol>1->data->type, $<symbol>3->data->type)){
128     char *lexem_copy = strdup($<symbol>1->lexem);
129     lexem_copy[strlen(lexem_copy)-4]='\0';
130     char s[200]; sprintf(s, "Wrong data types in attribution - the variable %s is a %s and the value of the
131 expression is a %s",
132 lexem_copy, type_names[$<symbol>1->data->type], type_names[$<symbol>3->data->type]);
133 free(lexem_copy);

```

```

134         raiseError(s);
135     }
136     $<symbol>$ = $<symbol>1;
137     if($<symbol>1->fake_memory_address != $<symbol>3->fake_memory_address){
138         char s[200]; sprintf(s, "t%d = t%d", $<symbol>1->fake_memory_address,
139 $<symbol>3->fake_memory_address); emit(s);
140         if($<symbol>3->data->type == _INTEGER)
141             createADDi(&block_mount, $<symbol>1->fake_memory_address+1,
142 $<symbol>3->fake_memory_address+1, $<symbol>3->data->v.integer);
143         else
144             createADDi(&block_mount, $<symbol>1->fake_memory_address+1,
145 $<symbol>3->fake_memory_address+1, 0);
146     }
147 }
148 | variable '=' word_expression {PRINT(("command -> [variable '=' word_expression]\n"))
149     if($<symbol>1 == NULL)
150         raiseError("Variable is not declared!");
151     if(!checkTypes($<symbol>1->data->type, $<symbol>3->data->type)){
152         char *lexem_copy = strdup($<symbol>1->lexem);
153         lexem_copy[strlen(lexem_copy)-4]='\0';
154         char s[200]; sprintf(s, "Wrong data types in attribution - the variable %s is a %s and the value of the
155 expression is a %s",
156             lexem_copy, type_names[$<symbol>1->data->type], type_names[$<symbol>3->data->type]);
157         free(lexem_copy);
158         raiseError(s);
159     }
160     $<symbol>$ = $<symbol>1;
161     if($<symbol>1->fake_memory_address != $<symbol>3->fake_memory_address){
162         char s[200]; sprintf(s, "t%d = %s", $<symbol>1->fake_memory_address,
163 $<symbol>3->data->v.word); emit(s);
164     }
165 }
166 | variable '=' task_call {PRINT(("command -> [variable '=' task_call]\n"))}
167 | givingup {PRINT(("command -> [givingup]\n"))}
168 | if_command {PRINT(("command -> [if_command]\n"))}
169 | for_command {PRINT(("command -> [for_command]\n"))}
170 | while_command {PRINT(("command -> [while_command]\n"))}
171 | farewell_command {PRINT(("command -> [farewell_command]\n"))}
172 | STOP {PRINT(("command -> [STOP]\n"))}
173 | jumpto_command {PRINT(("command -> [jumpto_command]\n"))}
174 | say_command {PRINT(("command -> [say_command]\n"))}
175 | listen_command {PRINT(("command -> [listen_command]\n"))}
176 | task_call {PRINT(("command -> [task_call]\n"))}
177 | label {PRINT(("command -> [label]\n"))}
178 ;
179
180 say_command: SAY expression {PRINT(("say_command -> [SAY expression]\n"))}
181 | SAY word_expression {PRINT(("say_command -> [SAY word_expression]\n"))}
182 ;
183
184 listen_command: LISTEN variable {PRINT(("LISTEN -> [variable]\n"))}
185 ;
186
187 if_cond: IF expression ':' { PRINT(("if_cond -> [IF expression ':']\n"))
188     $<symbol>$ = $<symbol>2;
189     sprintf($<symbol>$->to_emit, "ifFalse t%d goto __", $<symbol>2->fake_memory_address);
190 emit($<symbol>$->to_emit);
191     $<symbol>$->line_backpatch_risc = block_mount.num_inst;
192     createBeqEndi(&block_mount, $<symbol>2->fake_memory_address+1, 0, 0);
193     $<symbol>$->line_backpatch = num_prox_instr;
194 }
195 ;
196
197 elif_cond: ELIF expression ':' { PRINT(("elif_cond -> [ELIF expression ':']\n"))
198     $<symbol>$ = $<symbol>2;
199     sprintf($<symbol>$->to_emit, "ifFalse t%d goto __", $<symbol>2->fake_memory_address);
200 emit($<symbol>$->to_emit);
201     $<symbol>$->line_backpatch_risc = block_mount.num_inst;
202     createBeqEndi(&block_mount, $<symbol>2->fake_memory_address+1, 0, 0);

```

```

203         $<symbol>$->line_backpatch = num_prox_instr;
204     }
205 ;
206
207 if_command: if_cond statement ENDF {PRINT(("if_command -> [IF expression ':' statement ENDF]\n"))
208     emitBackpatch($<symbol>1->line_backpatch, num_prox_instr);
209     block_mount.insts[$<symbol>1->line_backpatch_risc].endereco = num_prox_instr;
210 }
211 | if_cond statement elif_cond statement ENDELIF {PRINT(("if_command -> [IF expression ':' statement ELIF
212 expression ':' statement ENDELIF]\n"))
213     emitBackpatch($<symbol>1->line_backpatch, $<symbol>3->line_backpatch - 2);
214     emitBackpatch($<symbol>3->line_backpatch, num_prox_instr);
215     block_mount.insts[$<symbol>1->line_backpatch_risc].endereco = $<symbol>3->line_backpatch - 2;
216     block_mount.insts[$<symbol>3->line_backpatch_risc].endereco = num_prox_instr;
217 }
218 ;
219
220 for_cond_aux: expression ':' {
221     PRINT(("for_cond_aux -> [expression ':']\n"))
222     $<symbol>$ = $<symbol>1;
223     $<symbol>$->line_backpatch = num_prox_instr;
224     emitBackpatch(line_backpatch_for_aux, num_prox_instr+1);
225     char *s = (char*) malloc(50*sizeof(char));
226     sprintf(s, "goto ____"); emit(s);
227     line_backpatch_for_aux = num_prox_instr;
228     free(s);
229 }
230 ;
231
232 for_cond: expression ',' expression ',' {
233     PRINT(("for_cond -> [expression ',' expression ',']\n"))
234     $<symbol>$ = $<symbol>3;
235     sprintf($<symbol>$->to_emit, "ifFalse t%d goto ____", $<symbol>3->fake_memory_address);
236 emit($<symbol>$->to_emit);
237     $<symbol>$->line_backpatch = num_prox_instr;
238     //$<symbol>$->line_backpatch_risc = block_mount.num_inst;
239     //createBeqEndi(&block_mount, $<symbol>2->fake_memory_address, 0, 0);
240     char *s = (char*) malloc(50*sizeof(char));
241     sprintf(s, "goto ____"); emit(s);
242     free(s);
243     line_backpatch_for_aux = num_prox_instr;
244     line_backpatch_for_aux2 = num_prox_instr;
245 }
246 ;
247
248 for_command: FOR for_cond for_cond_aux statement ENDFOR {
249     PRINT(("for_command -> [FOR for_cond for_cond_aux statement ENDFOR]\n"))
250     emitBackpatch(line_backpatch_for_aux, $<symbol>2->line_backpatch-2);
251     char *s = (char*) malloc(50*sizeof(char));
252     sprintf(s, "goto %d", line_backpatch_for_aux2 + 1); emit(s);
253     free(s);
254     emitBackpatch($<symbol>2->line_backpatch, num_prox_instr);
255 }
256 ;
257
258 while_cond: expression ':' {
259     PRINT(("expression -> [expression ':']\n"))
260     $<symbol>$ = $<symbol>1;
261     sprintf($<symbol>$->to_emit, "ifFalse t%d goto ____", $<symbol>1->fake_memory_address);
262 emit($<symbol>$->to_emit);
263     $<symbol>$->line_backpatch = num_prox_instr;
264 }
265 ;
266
267 while_command: WHILE while_cond statement ENDWHILE {
268     PRINT(("while_command -> [WHILE expression ':' statement ENDWHILE]\n"))
269     $<symbol>$ = $<symbol>2;
270     sprintf($<symbol>$->to_emit, "goto %d", $<symbol>$->line_backpatch - 1); emit($<symbol>$->to_emit);
271     emitBackpatch($<symbol>2->line_backpatch, num_prox_instr);

```

```

272     }
273 ;
274
275 farewell_command: FAREWELL expression {PRINT(("farewell_command -> [FAREWELL expression]\n"))}
276 ;
277
278 task_command: TASK IDENTIFIER task_parameters ':' statement ENDTASK {PRINT(("task_command -> [TASK
279 IDENTIFIER task_parameters ':' ENDTASK]\n"))
280 // TODO
281 }
282 ;
283
284 task_parameter: expression {PRINT(("task_parameter -> [expression]\n"))}
285 | expression ',' task_parameter {PRINT(("task_parameter -> [expression ',' task_parameter]\n"))}
286 ;
287
288 task_parameters: /* palavra vazia */ {PRINT(("task_parameter -> []\n"))}
289 | task_parameter {PRINT(("task_parameter -> [task_parameter]\n"))}
290 ;
291
292 task_call: TASK IDENTIFIER '(' task_parameters ')' {PRINT(("task_call -> [TASK IDENTIFIER '(' task_parameter ')]\n"))}
293 if(ht_get(curr_symbol_table, $<symbol>2->lexem, 1) == NULL){
294     $<symbol>2->data->v.word = strdup($<type_aux>1);
295     ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
296 }
297 }
298 ;
299
300 jumpto_command: JUMPTO IDENTIFIER {PRINT(("jumpto_command -> [JUMPTO IDENTIFIER]\n"))}
301 if(ht_get(curr_symbol_table, $<symbol>2->lexem, 1) == NULL){
302     //$<symbol>2->data->v.word = strdup("jumpto");
303     ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
304     $<symbol>$ = $<symbol>2;
305     char *lexem_copy = strdup($<symbol>2->lexem);
306     lexem_copy[strlen(lexem_copy)-4]='\0';
307     sprintf($<symbol>$->to_emit, "goto %s", lexem_copy); emit($<symbol>$->to_emit);
308     free(lexem_copy);
309     $<symbol>$->line_backpatch = num_prox_instr;
310 }
311 }
312 ;
313
314 label: IDENTIFIER ':' {PRINT(("label -> [IDENTIFIER ':']\n"))}
315 if(ht_get(curr_symbol_table, $<symbol>1->lexem, 1) == NULL){
316     $<symbol>1->fake_memory_address = num_prox_instr;
317     ht_set(curr_symbol_table, $<symbol>1->lexem, $<symbol>2);
318 } /* else{
319     emitBackpatch($<symbol>2->line_backpatch, num_prox_instr);
320 } */
321
322 $<symbol>$ = $<symbol>1;
323 char *lexem_copy = strdup($<symbol>1->lexem);
324 lexem_copy[strlen(lexem_copy)-4]=':';
325 lexem_copy[strlen(lexem_copy)-3]='\0';
326 sprintf($<symbol>$->to_emit, "%s", lexem_copy); emit($<symbol>$->to_emit);
327 free(lexem_copy);
328 }
329 ;
330
331 expression: math_expression {PRINT(("expression -> [math_expression]\n"))}
332     $<symbol>$ = $<symbol>1;
333     //$<id_temporario>$ = $<id_temporario>1;
334 }
335 | variable '=' expression {PRINT(("expression -> [variable '=' expression]\n"))}
336 if($<symbol>1 == NULL)
337     raiseError("Variable is not declared!");
338 if(!checkTypes($<symbol>1->data->type, $<symbol>3->data->type)){
339     char *lexem_copy = strdup($<symbol>1->lexem);
340     lexem_copy[strlen(lexem_copy)-4]='\0';

```



```

341         char s[200]; sprintf(s, "Wrong data types in attribution - the variable %s is a %s and the value of the
342 expression is a %s",
343         lexem_copy, type_names[$<symbol>1->data->type], type_names[$<symbol>3->data->type]);
344         free(lexem_copy);
345         raiseError(s);
346     }
347     $<symbol>$ = $<symbol>1;
348     if($<symbol>1->fake_memory_address != $<symbol>3->fake_memory_address){
349         sprintf($<symbol>$->to_emit, "t%d = t%d", $<symbol>1->fake_memory_address,
350 $<symbol>3->fake_memory_address); emit($<symbol>$->to_emit);
351         if($<symbol>3->data->type == _INTEGER)
352             createADDi(&block_mount, $<symbol>1->fake_memory_address+1,
353 $<symbol>3->fake_memory_address+1, $<symbol>3->data->v.integer);
354         else
355             createADDi(&block_mount, $<symbol>1->fake_memory_address+1,
356 $<symbol>3->fake_memory_address+1, 0);
357     }
358 }
359 ;
360
361 ex_aux_abre: '(' math_expression ex_aux_fecha {PRINT(("ex_aux_abre -> ['( math_expression ex_aux_fecha)\n"))
362         $<symbol>$ = $<symbol>2;
363         //$<id_temporario>$ = $<id_temporario>2;
364     }
365     | math_expression {$<symbol>$ = $<symbol>1;}
366 ;
367
368 ex_aux_fecha: ')' {PRINT(("ex_aux_fecha -> [')']\n"))}
369 ;
370
371 math_expression: ex_aux_abre ADD_OPERATOR ex_aux_abre { PRINT(("math_expression -> [ex_aux_abre
372 ADD_OPERATOR ex_aux_abre]\n"))
373         createSymbol(&($<symbol>$), "", $<symbol>1->token_type, $<symbol>1->data->type);
374         action_math_expression($<symbol>$, $<symbol>1, $<op>2, $<symbol>3); }
375     | ex_aux_abre DIV_OPERATOR ex_aux_abre { PRINT(("math_expression -> [ex_aux_abre DIV_OPERATOR
376 ex_aux_abre]\n"))
377         createSymbol(&($<symbol>$), "", $<symbol>1->token_type, $<symbol>1->data->type);
378         action_math_expression($<symbol>$, $<symbol>1, $<op>2, $<symbol>3); }
379     | ex_aux_abre POW_OPERATOR ex_aux_abre { PRINT(("math_expression -> [ex_aux_abre POW_OPERATOR
380 ex_aux_abre]\n"))
381         createSymbol(&($<symbol>$), "", $<symbol>1->token_type, $<symbol>1->data->type);
382         action_math_expression($<symbol>$, $<symbol>1, $<op>2, $<symbol>3); }
383     | ex_aux_abre LOGIC_OPERATOR ex_aux_abre { PRINT(("math_expression -> [ex_aux_abre
384 LOGIC_OPERATOR ex_aux_abre]\n"))
385         createSymbol(&($<symbol>$), "", $<symbol>1->token_type, $<symbol>1->data->type);
386         action_math_expression_logic($<symbol>$, $<symbol>1, $<op>2, $<symbol>3); }
387     | ex_aux_abre REL_OPERATOR ex_aux_abre { PRINT(("math_expression -> [ex_aux_abre REL_OPERATOR
388 ex_aux_abre]\n"))
389         createSymbol(&($<symbol>$), "", REL_OPERATOR, _INTEGER);
390         action_math_expression_rel($<symbol>$, $<symbol>1, $<op>2, $<symbol>3); }
391     | unary_operators ex_aux_abre %prec DIV_OPERATOR { PRINT(("math_expression -> [unary_operators ex_aux_abre
392 %prec DIV_OPERATOR]\n"))
393         createSymbol(&($<symbol>$), "", $<symbol>2->token_type, $<symbol>2->data->type);
394         action_math_expression_unary($<symbol>$, $<op>1, $<symbol>2); }
395     | math_term
396         char s_aux[200];
397         if($<symbol>1->token_type != _VARIABLE) {
398             if($<symbol>1->fake_memory_address == -1) { $<symbol>1->fake_memory_address = id_temporario++;
399 createDeclaration(&(block_mount), $<symbol>1->fake_memory_address + 1); }
400             //printf("TIIPO: %d\n", $<symbol>1->data->type);
401             if($<symbol>1->data->type == _INTEGER) { sprintf(s_aux, "%d", $<symbol>1->data->v.integer);
402                 createADDi(&block_mount, $<symbol>1->fake_memory_address+1, $<symbol>1->fake_memory_address+1,
403 $<symbol>1->data->v.integer); }
404             else if($<symbol>1->data->type == _REAL) sprintf(s_aux, "%lf", $<symbol>1->data->v.real);
405             else if($<symbol>1->data->type == _WORD) sprintf(s_aux, "%s", $<symbol>1->data->v.word);
406             char s[1000]; sprintf(s, "t%d = %s", $<symbol>1->fake_memory_address, s_aux); emit(s);
407         }
408         $<symbol>$ = $<symbol>1;
409         //$<id_temporario>$ = id_temporario++; char s[1000]; sprintf(s, "t%d = %s", $<id_temporario>$, s_aux); emit(s);

```

```

410     }
411     ;
412
413     variable_declarations: variable_declaration {PRINT(("variable_declarations -> [variable_declaration]\n"))}
414     | variable_declaration variable_declarations {PRINT(("variable_declarations -> [variable_declaration
415     variable_declarations]\n"))}
416     ;
417     /* {PRINT(("Squad encontrado %d\n", yyval))} guardando o yyval pra nao esquecermos */
418     variable_declaration: type IDENTIFIER {PRINT(("variable_declaration -> [type IDENTIFIER]\n"))
419     if(ht_get(curr_symbol_table, $<symbol>2->lexem, 0) == NULL){
420         $<symbol>2->data->type = $<type_declaration>;
421         $<symbol>2->token_type = _VARIABLE;
422         ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
423         createDeclaration(&(block_mount), $<symbol>2->fake_memory_address + 1);
424     } else {
425         raiseErrorVariableRedeclaration($<symbol>2->lexem);
426     }
427     }
428     | squad_declaration {PRINT(("variable_declaration -> [squad_declaration]\n"))}
429     | vector_declaration {PRINT(("variable_declaration -> [vector_declaration]\n"))}
430     ;
431
432     vector_access: IDENTIFIER '[' NUMBER ']' {PRINT(("vector_access -> [IDENTIFIER '[' NUMBER '']\n"))
433     /* PRINT((PRINT_ERROR "-> %s\n", yylineno, $<symbol>3->lexem))*/
434     | IDENTIFIER '[' IDENTIFIER ']' {PRINT(("vector_access -> [IDENTIFIER '[' IDENTIFIER '']\n"))}
435     ;
436
437     squad_access: IDENTIFIER SQUAD_ACCESS_DERREFERENCE IDENTIFIER {PRINT(("squad_access ->
438     [IDENTIFIER '->' IDENTIFIER ]\n"))}
439     | squad_access SQUAD_ACCESS_DERREFERENCE IDENTIFIER {PRINT(("squad_access -> [IDENTIFIER '->'
440     IDENTIFIER ]\n"))}
441     ;
442
443     squad_declaration: SQUAD IDENTIFIER ':' variable_declarations ENDSQUAD {PRINT(("squad_declaration -> [SQUAD
444     IDENTIFIER ':' variable_declarations ENDSQUAD]\n"))
445     if(ht_get(curr_symbol_table, $<symbol>2->lexem, 0) == NULL){
446         $<symbol>2->data->v.word = strdup($<type_aux>1);
447         ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
448     } else {
449         raiseErrorVariableRedeclaration($<symbol>2->lexem);
450     }
451     }
452     ;
453
454     vector_declaration: VECTOR IDENTIFIER NUMBER {PRINT(("vector_declaration -> [VECTOR IDENTIFIER
455     NUMBER]\n"))
456     if(ht_get(curr_symbol_table, $<symbol>2->lexem, 0) == NULL){
457         $<symbol>2->data->v.word = strdup($<type_aux>1);
458         ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
459     } else {
460         raiseErrorVariableRedeclaration($<symbol>2->lexem);
461     }
462     }
463     | VECTOR IDENTIFIER IDENTIFIER {PRINT(("vector_declaration -> [VECTOR IDENTIFIER
464     IDENTIFIER]\n"))
465     if(ht_get(curr_symbol_table, $<symbol>2->lexem, 0) == NULL){
466         $<symbol>2->data->v.word = strdup($<type_aux>1);
467         ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
468     } else {
469         raiseErrorVariableRedeclaration($<symbol>2->lexem);
470     }
471     }
472     ;
473
474     type: INTEGER {PRINT(("type -> [INTEGER]\n"))
475     $<type_declaration>$_ = _INTEGER;
476     }
477     | REAL {PRINT(("type -> [REAL]\n"))
478     $<type_declaration>$_ = _REAL;

```

```

479     }
480     | WORD      {PRINT(("type -> [WORD]\n"))
481                 $<type_declaration>$ = _WORD;
482     }
483 ;
484
485 unary_operators: ADD_OPERATOR {PRINT(("unary_operators -> [ADD_OPERATOR]\n")) $<op>$ = $<op>1;}
486                 | NEG_OPERATOR {PRINT(("unary_operators -> [NEG_OPERATOR]\n")) $<op>$ = $<op>1;}
487 ;
488
489 word_term: WORD_VALUE {PRINT(("word_term -> [WORD_VALUE]\n"))
490                     $<symbol>$ = $<symbol>1;
491                     }
492     | expression {PRINT(("word_term -> [expression]\n"))}
493 ;
494
495 word_term_aux: word_term {PRINT(("word_term_aux -> [word_term]\n"))
496                     $<symbol>$ = $<symbol>1;
497                     }
498     | word_term WORD_CONCAT_OPERATOR word_term_aux {PRINT(("word_term_aux -> [word_term
499 WORD_CONCAT_OPERATOR word_term]\n"))
500                     $<symbol>$ = $<symbol>1;
501                     sprintf($<symbol>$->data->v.word, "%s%s", $<symbol>$->data->v.word,
502 $<symbol>3->data->v.word);
503                     }
504 ;
505
506 word_expression: WORD_VALUE {PRINT(("word_expression -> [WORD_VALUE]\n"))
507                     $<symbol>$ = $<symbol>1;
508                     }
509     | word_term WORD_CONCAT_OPERATOR word_term_aux {PRINT(("word_expression -> [word_term
510 WORD_CONCAT_OPERATOR expression]\n"))
511                     $<symbol>$ = $<symbol>1;
512                     sprintf($<symbol>$->data->v.word, "%s%s", $<symbol>$->data->v.word,
513 $<symbol>3->data->v.word);
514                     }
515 ;
516
517 variable: IDENTIFIER {PRINT(("variable -> [IDENTIFIER]\n"))
518                     //printf("%s\n", $<symbol>1->lexem);
519                     $<symbol>$ = ht_get(curr_symbol_table, $<symbol>1->lexem, 1);
520                     if($<symbol>$ == NULL){
521                         raiseError("Variable is not declared!");
522                     }
523                     //destroySymbol(&($<symbol>1));
524                     }
525     | vector_access {PRINT(("variable -> [vector_access]\n"))}
526     | squad_access {PRINT(("variable -> [squad_access]\n"))}
527 ;
528
529 math_term: NUMBER {PRINT(("math_term -> [NUMBER]\n"))
530             $<symbol>$ = $<symbol>1;
531             }
532     | REAL_NUMBER {PRINT(("math_term -> [REAL_NUMBER]\n"))
533             $<symbol>$ = $<symbol>1;
534             }
535     | variable {PRINT(("math_term -> [variable]\n"))
536             $<symbol>$ = $<symbol>1;
537             } /* ATENCAO A ESTA VARIABEL AQUI */
538 ;
539
540
541 givingup: GIVEUP {PRINT(("givingup -> [GIVEUP]\nADEUS :)"))
542             exit(0);}
543 ;
544
545 %%
546 void action_math_expression(Symbol *sd, Symbol *s1, char op, Symbol *s2){
547     if(sd->fake_memory_address == -1) { sd->fake_memory_address = id_temporario++; createDeclaration(&(block_mount),

```

```

548 sd->fake_memory_address + 1); }
549 upTypes(sd, s1->data->type, s2->data->type);
550 sprintf(sd->to_emit, "t%d = t%d %c t%d", sd->fake_memory_address, s1->fake_memory_address, op,
551 s2->fake_memory_address);
552 switch(op){
553     case '+': createADD(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1,
554 s2->fake_memory_address+1); break;
555     case '-': createSUB(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1,
556 s2->fake_memory_address+1); break;
557     case '*': createMUL(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1,
558 s2->fake_memory_address+1); break;
559     case '/': createDIV(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1,
560 s2->fake_memory_address+1); break;
561 }
562 //if($<symbol>2->token_type != _VARIABLE){ // immediato
563 emit(sd->to_emit);
564 }
565
566 void action_math_expression_logic(Symbol *sd, Symbol *s1, char op, Symbol *s2){
567 if(sd->fake_memory_address == -1) { sd->fake_memory_address = id_temporario++; createDeclaration(&(block_mount),
568 sd->fake_memory_address + 1); }
569 upTypes(sd, s1->data->type, s2->data->type);
570 switch((int)op){
571     case AND: sprintf(sd->to_emit, "t%d = t%d AND t%d", sd->fake_memory_address, s1->fake_memory_address,
572 s2->fake_memory_address); break;
573     case OR: sprintf(sd->to_emit, "t%d = t%d OR t%d", sd->fake_memory_address, s1->fake_memory_address,
574 s2->fake_memory_address); break;
575 }
576 emit(sd->to_emit);
577 }
578
579 void action_math_expression_rel(Symbol *sd, Symbol *s1, char op, Symbol *s2){
580 if(sd->fake_memory_address == -1) { sd->fake_memory_address = id_temporario++; createDeclaration(&(block_mount),
581 sd->fake_memory_address + 1); }
582 switch((int)op){
583     case EQ: sprintf(sd->to_emit, "t%d = t%d == t%d", sd->fake_memory_address, s1->fake_memory_address,
584 s2->fake_memory_address); break;
585     case NE: sprintf(sd->to_emit, "t%d = t%d != t%d", sd->fake_memory_address, s1->fake_memory_address,
586 s2->fake_memory_address); break;
587     case GE: sprintf(sd->to_emit, "t%d = t%d >= t%d", sd->fake_memory_address, s1->fake_memory_address,
588 s2->fake_memory_address); break;
589     case LE: sprintf(sd->to_emit, "t%d = t%d <= t%d", sd->fake_memory_address, s1->fake_memory_address,
590 s2->fake_memory_address); break;
591     case GT:
592         createSgt(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1, s2->fake_memory_address+1);
593         sprintf(sd->to_emit, "t%d = t%d > t%d", sd->fake_memory_address, s1->fake_memory_address,
594 s2->fake_memory_address); break;
595     case LT:
596         createSlgt(&block_mount, sd->fake_memory_address+1, s1->fake_memory_address+1, s2->fake_memory_address+1);
597         sprintf(sd->to_emit, "t%d = t%d < t%d", sd->fake_memory_address, s1->fake_memory_address,
598 s2->fake_memory_address); break;
599 }
600 emit(sd->to_emit);
601 }
602
603 void action_math_expression_unary(Symbol *sd, char op, Symbol *s1){
604 if(sd->fake_memory_address == -1) { sd->fake_memory_address = id_temporario++; createDeclaration(&(block_mount),
605 sd->fake_memory_address + 1); }
606 switch(op){
607     case '!': sprintf(sd->to_emit, "t%d = !t%d", s1->fake_memory_address, s1->fake_memory_address); break;
608     case '-': sprintf(sd->to_emit, "t%d = -t%d", s1->fake_memory_address, s1->fake_memory_address); break;
609 }
610 emit(sd->to_emit);
611 }
612
613 void raiseError(char *msg){
614 fprintf(stderr, "Near line: %d ---- Error: %s <---\n\n", yylineno, msg);
615 emit("--ERRO--");
616 fclose(arq_three_address_code);

```

```

617     exit(1);
618     //yyerror(msg);
619 }
620
621 void raiseErrorVariableRedeclaration(char *lexem){
622     char error_msg[50];
623     char *lexem_copy = strdup(lexem);
624     lexem_copy[strlen(lexem_copy)-4]='\0';
625     sprintf(error_msg, "The variable %s has already been declared before!", lexem_copy);
626     free(lexem_copy);
627     yyerror(error_msg);
628 }
629
630 int yyerror(const char *s){
631     fprintf(stderr, "Line: %d ---> %s <---\n\n", yylineno, s);
632     emit("--ERRO--");
633     fclose(arq_three_address_code);
634     exit(1);
635 }
636
637 void emitBackpatch(int linha_origem, int linha_destino){
638     fclose(arq_three_address_code);
639     arq_three_address_code = NULL;
640     FILE *arq_three_address_code_temp = NULL;
641     arq_three_address_code = fopen("arq_three_address_code_generated", "r"); // abrindo para leitura
642     arq_three_address_code_temp = fopen("arq_three_address_code_generated_temp", "w"); // abrindo para escrita
643     if(arq_three_address_code == NULL){
644         raiseError("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated'\n");
645     }
646     if(arq_three_address_code_temp == NULL){
647         raiseError("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated_temp'\n");
648     }
649     int count = 0;
650     char buffer[255];
651     while ((fgets(buffer, 255, arq_three_address_code)) != NULL)
652     {
653         count++;
654
655         /* If current line is line to replace */
656         if (count == linha_origem){
657             buffer[strlen(buffer)-4] = '\0';
658             char *str_aux = (char*) malloc(20*sizeof(char));
659             sprintf(str_aux, "%d\n", (linha_destino+1));
660             strcat(buffer, str_aux);
661             free(str_aux);
662             fputs(buffer, arq_three_address_code_temp);
663         } else {
664             fputs(buffer, arq_three_address_code_temp);
665         }
666     }
667
668     fclose(arq_three_address_code);
669     fclose(arq_three_address_code_temp);
670
671     remove("arq_three_address_code_generated");
672
673     rename("arq_three_address_code_generated_temp", "arq_three_address_code_generated");
674     arq_three_address_code = fopen("arq_three_address_code_generated", "a+"); // abrindo para append para continuar o
675     processo
676 }
677
678 void emit(char *msg){
679     printf("%s\n", msg);
680     fprintf(arq_three_address_code, "%s\n", msg);
681     num_prox_instr++;
682 }
683
684 void emitSymbol(Symbol *symbol){
685     PRINT(("%"d", symbol->data->v.integer))

```

```

686 //PRINT(("s", symbol->to_emit))
687 //PRINT((symbol->))
688 /*switch(symbol->token_type){
689     case _VARIABLE: {
690         PRINT(("t%d", symbol->data->v.integer))
691         //switch(symbol->data->token_type){
692             // case
693             //}
694     } break;
695 }*/
696 }
697
698 void emitCode(){
699     hashtable_t *hashtable = first_symbol_table;
700     if(hashtable == NULL){
701         return;
702     }
703     hashtable_t *st = hashtable;
704     int i = 0;
705     entry_t *pair;
706
707     pair = hashtable->first_entry;
708     printf("IMPRIMINDO CODIGO\n");
709     while(pair != NULL){
710         Symbol *s = pair->value;
711         emitSymbol(s);
712
713         pair = pair->next;
714     }
715
716     if(hashtable->child_hash != NULL){ // imprime os filhos primeiro
717         ht_print(hashtable->child_hash);
718     }
719     if(hashtable->brother_hash != NULL){
720         ht_print(hashtable->brother_hash);
721     }
722 }
723
724
725 int main(int argc, char *argv[]){
726     first_symbol_table = NULL;
727     curr_symbol_table = NULL;
728     id_temporario = 0; //MAX_TAM_HASH;
729     _id_temporario = &id_temporario;
730
731     block_mount = createBlock();
732     //printf("Numero de parametros: %d\n", argc);
733     if(argc == 2){
734         //FILE *f;
735         if(!(yyin = fopen(argv[1], "r"))){
736             PRINT(("Could not open the file!\n"))
737             return 0;
738         }
739         char c;
740         printf("----- Readed Code ----- \n\n");
741         printf("1 ");
742         int l = 2;
743         while((c = fgetc(yyin)) != EOF){
744             printf("%c", c);
745             if(c == '\n'){
746                 printf("%d ", l++);
747             }
748         }
749         printf("\n\n");
750         rewind(yyin);
751     }
752     arq_three_address_code = NULL;
753     arq_three_address_code = fopen("arq_three_address_code_generated", "w");
754     if(arq_three_address_code == NULL){

```

```
755     raiseError("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated\n");
756 }
757 int r = yyparse();
758 //ht_print(first_symbol_table);
759 //emitCode();
760 fclose(arq_three_address_code);
761 imprimeRISCV(&block_mount);
762 imprimeLLVM(&block_mount);
763 return r;
764 }
```

## Apêndice C - *symbol.h*

```

1  #ifndef ESTRUTURA_TIPOS_INCLUDED
2  #define ESTRUTURA_TIPOS_INCLUDED
3
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7
8      #include <stdlib.h>
9      #include <stdio.h>
10     #include <limits.h>
11     #include <string.h>
12     #include <math.h>
13
14     //#define TAM_LEXEMA 255 // colocar?
15
16     #define MAX_TAM_WORD 255
17
18     enum SymbolType {NOTHING, _REAL, FFUNCTION, _INTEGER, _WORD, AGGREGATED, _VARIABLE};
19
20
21     typedef void* (Function) (void* p, ...);
22
23     typedef struct Squad{
24         struct Symbol *internal_variables; // TODO: Arrumar
25     } Squad;
26
27     typedef struct Data{
28         enum SymbolType type;
29         union{
30             int integer;
31             double real;
32             char *word;
33             Squad squad;
34             Function *Function;
35         } v;
36     } Data;
37
38     typedef struct Symbol{
39         int fake_memory_address;
40         char *lexem;
41         int token_type;
42         Data *data;
43         char to_emit[255]; // limitacao de emissao (se der tempo, melhorar)
44         int line_backpatch;
45         int line_backpatch_risc;
46     } Symbol;
47
48     void createData(Data **data, enum SymbolType type);
49     void _createSymbol(Symbol **symbol, char *lexem, int token_type, Data *data);
50     void createSymbol(Symbol **symbol, char *lexem, int token_type, enum SymbolType type);
51     void destroySymbol(Symbol **symbol);
52     void destroyData(Data **data);
53     void destroySquad(Squad **squad);
54     int checkTypes(enum SymbolType t1, enum SymbolType t2);
55     void upTypes(Symbol *symbol, enum SymbolType t1, enum SymbolType t2);
56     Symbol* applyBinaryOperatorInSymbols(Symbol *s1, char op, Symbol *s2);
57
58     #ifdef __cplusplus
59     }
60     #endif
61
62     #endif /* ESTRUTURA_TIPOS_INCLUDED */

```



## Apêndice D - *symbol.c*

```

1  #include "symbol.h"
2
3  void createData(Data **data, enum SymbolType type){
4      (*data) = (Data*) malloc(sizeof(Data));
5      (*data)->type = type;
6  }
7
8  void _createSymbol(Symbol **symbol, char *lexem, int token_type, Data *data){
9      (*symbol) = (Symbol*) malloc(sizeof(Symbol));
10     if(lexem != NULL){
11         (*symbol)->lexem = strdup(lexem);
12         strcat((*symbol)->lexem, "_key");
13     }
14     else
15         (*symbol)->lexem = NULL;
16     (*symbol)->fake_memory_address = -1;
17     (*symbol)->token_type = token_type;
18     (*symbol)->data = data;
19 }
20
21 void createSymbol(Symbol **symbol, char *lexem, int token_type, enum SymbolType type){
22
23     Data *d;
24     //createData(&d, type);
25     createData(&d, type);
26     _createSymbol(symbol, lexem, token_type, d);
27 }
28
29 void destroySymbol(Symbol **symbol){
30     if(symbol != NULL && *symbol != NULL){
31         Symbol *removedor = (*symbol); // TODO: melhorar isso, dando free nas coisas internas
32         destroyData(&(removedor->data));
33         free(removedor);
34     }
35 }
36
37 void destroyData(Data **data){
38     if(data != NULL && *data != NULL){
39         Data *removedor = (*data);
40         if(removedor->type == _WORD){
41             char *r_word = removedor->v.word;
42             free(r_word);
43         }
44         free(removedor);
45     }
46 }
47
48 void destroySquad(Squad **squad){
49     // TODO
50 }
51
52 int checkTypes(enum SymbolType t1, enum SymbolType t2){
53     if(t1 != t2){
54         if(t1 == _REAL && t2 == _INTEGER)
55             return 1;
56         return 0;
57     }
58     return 1;
59 }
60
61 void upTypes(Symbol *symbol, enum SymbolType t1, enum SymbolType t2){
62     if(t1 != t2){
63         if(t1 == _REAL && t2 == _INTEGER || (t1 == _INTEGER && t2 == _REAL))
64             symbol->data->type = _REAL;

```

```

65     }
66 }
67
68 Symbol* applyBinaryOperatorInSymbols(Symbol *s1, char op, Symbol *s2){
69     if(s1->data != NULL && s2->data != NULL){
70         if(s1->data->type == _INTEGER && s2->data->type == _INTEGER){
71             int a, b;
72             a = s1->data->v.integer;
73             b = s2->data->v.integer;
74
75             switch(op){
76                 case '+': a = a + b; break;
77                 case '-': a = a - b; break;
78                 case '/': if(b == 0) return NULL; a = a / b; break;
79                 case '*': a = a * b; break;
80                 case '%': a = a % b; break;
81                 case '^': a = pow(a, b); break;
82             }
83             s1->data->v.integer = a;
84             return s1;
85         }
86         if(s1->data->type == _INTEGER && s2->data->type == _REAL){
87             int a;
88             double b;
89             a = s1->data->v.integer;
90             b = s2->data->v.real;
91
92             switch(op){
93                 case '+': b = a + b; break;
94                 case '-': b = a - b; break;
95                 case '/': if(b == 0) return NULL; b = (double) a / b; break;
96                 case '*': b = a * b; break;
97                 case '%': return NULL; break;
98                 case '^': b = pow(a, b); break;
99             }
100             s1->data->type = _REAL;
101             s1->data->v.real = b;
102             return s1;
103         }
104         if(s1->data->type == _REAL && s2->data->type == _INTEGER){
105             double a;
106             int b;
107             a = s1->data->v.real;
108             b = s2->data->v.integer;
109
110             switch(op){
111                 case '+': a = a + b; break;
112                 case '-': a = a - b; break;
113                 case '/': if(b == 0) return NULL; a = (double) a / b; break;
114                 case '*': a = a * b; break;
115                 case '%': return NULL; break;
116                 case '^': a = pow(a, b); break;
117             }
118             s1->data->v.real = a;
119             return s1;
120         }
121         if(s1->data->type == _REAL && s2->data->type == _REAL){
122             double a, b;
123             a = s1->data->v.real;
124             b = s2->data->v.real;
125
126             switch(op){
127                 case '+': a = a + b; break;
128                 case '-': a = a - b; break;
129                 case '/': if(b == 0) return NULL; a = (double) a / b; break;
130                 case '*': a = a * b; break;
131                 case '%': return NULL; break;
132                 case '^': a = pow(a, b); break;
133             }

```

```
134     s1->data->v.real = a;  
135     return s1;  
136 }  
137 return NULL;  
138 }  
139 }
```

## Apêndice E - *hash-table.h*

```

1 // https://gist.github.com/tonious/1377667/d9e4f51f05992f79455756836c9371942d0f0cee
2
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <limits.h>
6 #include <string.h>
7 #include "symbol.h"
8
9 #define MAX_TAM_HASH 500
10
11 struct entry_s {
12     char *key;
13     Symbol *value;
14     struct entry_s *next;
15 };
16
17 typedef struct entry_s entry_t;
18 typedef struct hashtable_s hashtable_t;
19
20 struct hashtable_s {
21     int size;
22     entry_t *first_entry;
23     struct entry_s **table;
24     hashtable_t *previous_hash;
25     hashtable_t *child_hash;
26     hashtable_t *brother_hash;
27 };
28
29 hashtable_t *first_symbol_table; // global
30 hashtable_t *curr_symbol_table; // global
31
32 hashtable_t *ht_create( int size );
33 int ht_hash( hashtable_t *hashtable, char *key );
34 entry_t *ht_newpair( char *key, Symbol *value );
35 void ht_set( hashtable_t *hashtable, char *key, Symbol *value );
36 Symbol *ht_get( hashtable_t *hashtable, char *key, int find_fathers );
37 void ht_print( hashtable_t *hashtable );

```

## Apêndice F - *hash-table.c*

```

1  #include "hash-table.h"
2  extern int *_id_temporario;
3  // Create a new hashtable.
4  hashtable_t *ht_create( int size ) {
5
6      hashtable_t *hashtable = NULL;
7      int i;
8
9      if( size < 1 ) return NULL;
10
11     // Allocate the table itself.
12     if( ( hashtable = malloc( sizeof( hashtable_t ) ) ) == NULL ) {
13         return NULL;
14     }
15
16     // Allocate pointers to the head nodes.
17     if( ( hashtable->table = malloc( sizeof( entry_t * ) * size ) ) == NULL ) {
18         return NULL;
19     }
20     for( i = 0; i < size; i++ ) {
21         hashtable->table[i] == NULL;
22     }
23
24     hashtable->size = size;
25     hashtable->first_entry = NULL;
26     hashtable->previous_hash = NULL;
27     hashtable->brother_hash = NULL;
28     hashtable->child_hash = NULL;
29
30     return hashtable;
31 }
32
33 // Hash a string for a particular hash table.
34 int ht_hash( hashtable_t *hashtable, char *key ) {
35     /*
36     int hashval;
37     int i = 0;
38     //printf("Imakey %s %d\n", key, strlen( key ));
39     // Convert our string to an integer
40     while( hashval < UINT_MAX && i < strlen( key ) ) {
41         hashval = hashval << 4;
42         hashval += key[ i ];
43         i++;
44     }
45     //printf("HASHVAL: %ul %d %d\n", hashval, hashtable->size, hashval % hashtable->size);
46     printf("PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP: %d\n\n", hashval);
47     return hashval % hashtable->size;
48     */
49     unsigned long hash = hashtable->size;
50     int c;
51
52     while (c = *key++) {
53         hash = ((hash << 5) + hash) + c; /* hash * 33 + c */ /*
54     https://stackoverflow.com/questions/2535284/how-can-i-hash-a-string-to-an-int-using-c */
55     }
56
57     return ((unsigned int) hash) % hashtable->size;
58 }
59
60 // Create a key-value pair.
61 entry_t *ht_newpair( char *key, Symbol *value ) {
62     entry_t *newpair;
63
64     if( ( newpair = malloc( sizeof( entry_t ) ) ) == NULL ) {

```

```

65     return NULL;
66 }
67
68 if( ( newpair->key = strdup( key ) ) == NULL ) {
69     return NULL;
70 }
71
72 if( ( newpair->value = value ) == NULL ) {
73     return NULL;
74 }
75
76 newpair->next = NULL;
77
78 return newpair;
79 }
80
81 // Insert a key-value pair into a hash table.
82 void ht_set( hashtable_t *hashtable, char *key, Symbol *value ) {
83     int bin = 0;
84     entry_t *newpair = NULL;
85     entry_t *next = NULL;
86     entry_t *last = NULL;
87
88     bin = ht_hash( hashtable, key );
89     //printf("BBBBBBBBIN INSEEEERT: %d %s\n", bin, key);
90
91     next = hashtable->table[ bin ];
92
93     while( next != NULL && next->key != NULL && strcmp( key, next->key ) > 0 ) {
94         last = next;
95         next = next->next;
96     }
97
98     // There's already a pair. Let's replace that string.
99     if( next != NULL && next->key != NULL && strcmp( key, next->key ) == 0 ) {
100
101         //free( next->value );
102         //printf("TTTTTT\n");
103         destroySymbol(&(next->value));
104         next->value = value;
105
106         // Nope, couldn't find it. Time to grow a pair.
107     } else {
108         //printf("YYYYYYYYYY %d\n", bin);
109         newpair = ht_newpair( key, value );
110         newpair->value->fake_memory_address = (*_id_temporario)++;
111         //newpair->value->fake_memory_address = bin;
112         if(hashtable->first_entry == NULL){
113             hashtable->first_entry = newpair;
114         }
115         // We're at the start of the linked list in this bin.
116         if( next == hashtable->table[ bin ] ) {
117             newpair->next = next;
118             hashtable->table[ bin ] = newpair;
119
120             // We're at the end of the linked list in this bin.
121         } else if ( next == NULL ) {
122             last->next = newpair;
123
124             // We're in the middle of the list.
125         } else {
126             newpair->next = next;
127             last->next = newpair;
128         }
129     }
130 }
131
132 // Retrieve a key-value pair from a hash table.
133 Symbol *ht_get( hashtable_t *hashtable, char *key, int find_fathers ) {

```

```

134     int bin = 0;
135     entry_t *pair;
136     //printf("CHAVE: %s\n", key);
137     bin = ht_hash( hashtable, key );
138     //printf("BBBBBBBBBIN GGEEET: %d %s\n", bin, key);
139     //printf("%d\n", bin);
140     // Step through the bin, looking for our value.
141     pair = hashtable->table[ bin ];
142     while( pair != NULL && pair->key != NULL && strcmp( key, pair->key ) > 0 ) {
143         pair = pair->next;
144     }
145
146     // Did we actually find anything?
147     if( pair == NULL || pair->key == NULL || strcmp( key, pair->key ) != 0 ) {
148         /*//printf("AAAAAAA\n");
149         if(pair == NULL) printf("ee\n");
150         else if(pair->key == NULL) printf("ff\n");
151         else if(strcmp( key, pair->key ) != 0) printf("ggg\n");*/
152         //return pair->value;
153         if(hashtable->previous_hash != NULL && find_fathers == 1){
154             return ht_get(hashtable->previous_hash, key, find_fathers );
155         }
156         return NULL;
157     } else {
158         return pair->value;
159     }
160 }
161
162 }
163
164 void ht_print( hashtable_t *hashtable ){
165     if(hashtable == NULL){
166         return;
167     }
168     hashtable_t *st = hashtable;
169     int i = 0;
170     entry_t *pair;
171     printf("\n----- SYMBOL TABLE ----- \n\n");
172     int t_max = 43;
173     int t = t_max - strlen("Lexem");
174     printf("Lexem%*s%*s\n", t <= 0 ? 1 : t, "Data Type", t_max, "Memory Address (fake)");
175     for(i = 0; i < st->size; i++){
176         pair = st->table[i];
177         if(pair != NULL){
178             //char *key = pair->key;
179             Symbol *s = pair->value;
180             //printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
181             char *lexem_copy = strdup(s->lexem);
182             //printf("BBBBBBBBBBBBBBBBBBBBBB\n");
183             lexem_copy[strlen(lexem_copy)-4]='\0';
184             //printf("CCCCCCCCCCCCCCCCCCCC\n");
185             int t = t_max - strlen(lexem_copy);
186             //printf("CCCCCCCCCCCCCCCCCCCC\n");
187             //printf("%s%*s%*d\n", lexem_copy, t <= 0 ? 1 : t, s->data->v.word, t_max, pair->fake_memory_address);
188             // TODO: Trocar essa impressao -> Não temos apenas word agora para imprimir, temos diversos outros valores!
189             // TODO: Trocar essa impressao -> Não temos apenas word agora para imprimir, temos diversos outros valores!
190             // TODO: Trocar essa impressao -> Não temos apenas word agora para imprimir, temos diversos outros valores!
191             // TODO: Trocar essa impressao -> Não temos apenas word agora para imprimir, temos diversos outros valores!
192             //printf("CCCCCCCCCCCCCCCCCCCC\n");
193             free(lexem_copy);
194         }
195     }
196     printf("\n----- \n\n");
197
198     if(hashtable->child_hash != NULL){ // imprime os filhos primeiro
199         ht_print(hashtable->child_hash);
200     }
201     if(hashtable->brother_hash != NULL){
202         ht_print(hashtable->brother_hash);

```

203	}
204	}



## Apêndice G - *custom\_defines.h*

```
1  #ifndef CUSTOM_DEFINES_H_HEADER_  
2  #define CUSTOM_DEFINES_H_HEADER_  
3  
4  #define PRINT_ERROR "----- Erro encontrado na linha %d -----\\n"  
5  #define PRINT(args) printf args ;  
6  #define EQ 0  
7  #define NE 1  
8  #define GE 2  
9  #define LE 3  
10 #define GT 4  
11 #define LT 5  
12 #define AND 0  
13 #define OR 1  
14 #define TRUE 1  
15 #define FALSE 0  
16  
17 #endif
```

## Apêndice H - *y.tab.h* (gerado pelo YACC)

```

1  /* A Bison parser, made by GNU Bison 3.5.1.  */
2
3  /* Bison interface for Yacc-like parsers in C
4
5     Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2020 Free Software Foundation,
6     Inc.
7
8     This program is free software: you can redistribute it and/or modify
9     it under the terms of the GNU General Public License as published by
10    the Free Software Foundation, either version 3 of the License, or
11    (at your option) any later version.
12
13    This program is distributed in the hope that it will be useful,
14    but WITHOUT ANY WARRANTY; without even the implied warranty of
15    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16    GNU General Public License for more details.
17
18    You should have received a copy of the GNU General Public License
19    along with this program. If not, see <http://www.gnu.org/licenses/>.  */
20
21  /* As a special exception, you may create a larger work that contains
22  part or all of the Bison parser skeleton and distribute that work
23  under terms of your choice, so long as that work isn't itself a
24  parser generator using the skeleton or a modified version thereof
25  as a parser skeleton. Alternatively, if you modify or redistribute
26  the parser skeleton itself, you may (at your option) remove this
27  special exception, which will cause the skeleton and the resulting
28  Bison output files to be licensed under the GNU General Public
29  License without this special exception.
30
31  This special exception was added by the Free Software Foundation in
32  version 2.2 of Bison.  */
33
34  /* Undocumented macros, especially those whose name start with YY_,
35  are private implementation details. Do not rely on them.  */
36
37  #ifndef YY_YY_Y_TAB_H_INCLUDED
38  # define YY_YY_Y_TAB_H_INCLUDED
39  /* Debug traces.  */
40  #ifndef YYDEBUG
41  # define YYDEBUG 0
42  #endif
43  #if YYDEBUG
44  extern int yydebug;
45  #endif
46  /* "%code requires" blocks.  */
47  #line 26 "translate.y"
48
49      #include "hash-table.h"
50      #include "montador_lexer.h"
51      void action_math_expression(Symbol *sd, Symbol *s1, char op, Symbol *s2);
52      void action_math_expression_logic(Symbol *sd, Symbol *s1, char op, Symbol *s2);
53      void action_math_expression_rel(Symbol *sd, Symbol *s1, char op, Symbol *s2);
54      void action_math_expression_unary(Symbol *sd, char op, Symbol *s1);
55      void emitBackpatch(int linha_origem, int linha_destino);
56
57
58  #line 59 "y.tab.h"
59
60  /* Token type.  */
61  #ifndef YYTOKENTYPE
62  # define YYTOKENTYPE
63      enum yytokentype
64      {

```

```

65  WORD_VALUE = 258,
66  NUMBER = 259,
67  REAL_NUMBER = 260,
68  SQUAD_ACCESS_DERREFERENCE = 261,
69  INTEGER = 262,
70  WORD = 263,
71  REAL = 264,
72  SQUAD = 265,
73  ENDSQUAD = 266,
74  VECTOR = 267,
75  IDENTIFIER = 268,
76  ADD_OPERATOR = 269,
77  DIV_OPERATOR = 270,
78  POW_OPERATOR = 271,
79  LOGIC_OPERATOR = 272,
80  NEG_OPERATOR = 273,
81  REL_OPERATOR = 274,
82  WORD_CONCAT_OPERATOR = 275,
83  GIVEUP = 276,
84  BLOCK_BEGIN = 277,
85  BLOCK_END = 278,
86  SAY = 279,
87  LISTEN = 280,
88  IF = 281,
89  ENDIF = 282,
90  ELIF = 283,
91  ENDELIF = 284,
92  FOR = 285,
93  ENDFOR = 286,
94  WHILE = 287,
95  ENDWHILE = 288,
96  TASK = 289,
97  ENDTASK = 290,
98  FAREWELL = 291,
99  JUMPTO = 292,
100 STOP = 293
101 };
102 #endif
103 /* Tokens. */
104 #define WORD_VALUE 258
105 #define NUMBER 259
106 #define REAL_NUMBER 260
107 #define SQUAD_ACCESS_DERREFERENCE 261
108 #define INTEGER 262
109 #define WORD 263
110 #define REAL 264
111 #define SQUAD 265
112 #define ENDSQUAD 266
113 #define VECTOR 267
114 #define IDENTIFIER 268
115 #define ADD_OPERATOR 269
116 #define DIV_OPERATOR 270
117 #define POW_OPERATOR 271
118 #define LOGIC_OPERATOR 272
119 #define NEG_OPERATOR 273
120 #define REL_OPERATOR 274
121 #define WORD_CONCAT_OPERATOR 275
122 #define GIVEUP 276
123 #define BLOCK_BEGIN 277
124 #define BLOCK_END 278
125 #define SAY 279
126 #define LISTEN 280
127 #define IF 281
128 #define ENDIF 282
129 #define ELIF 283
130 #define ENDELIF 284
131 #define FOR 285
132 #define ENDFOR 286
133 #define WHILE 287

```

```

134 #define ENDWHILE 288
135 #define TASK 289
136 #define ENDTASK 290
137 #define FAREWELL 291
138 #define JUMPTO 292
139 #define STOP 293
140
141 /* Value type. */
142 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
143 union YYSTYPE
144 {
145 #line 36 "translate.y"
146
147     Symbol *symbol;
148     hashtable_t *symbol_table;
149     char type_aux[20]; // apenas para print...
150     char op;
151     int type_declaration;
152     int id_temporario;
153
154
155 #line 156 "y.tab.h"
156
157 };
158 typedef union YYSTYPE YYSTYPE;
159 # define YYSTYPE_IS_TRIVIAL 1
160 # define YYSTYPE_IS_DECLARED 1
161 #endif
162
163
164 extern YYSTYPE yylval;
165
166 int yyparse (void);
167
168 #endif /* !YY_Y_Y_TAB_H_INCLUDED */

```

## Apêndice I - *montador\_lexer.h*

```

1  #include <stdio.h>
2
3  //int line_backpatch_risc;
4
5  typedef struct Instrucao{
6      int endereco;
7      char * label;
8      int imediato;
9      int rd;
10     int src1, src2;
11     char * op;
12     int labelint;
13 } Instrucao;
14
15 typedef struct Block{
16     Instrucao insts[50];
17     int num_inst;
18 } Block;
19
20 Block block_mount;
21
22 Block createBlock();
23
24 void imprimeRISCV(Block *b);
25
26 void imprimeLLVM(Block *b);
27
28 void createADD(Block *b, int rd_int, int src1_int, int src2_int);
29 void createSUB(Block *b, int rd_int, int src1_int, int src2_int);
30 void createMUL(Block *b, int rd_int, int src1_int, int src2_int);
31 void createDIV(Block *b, int rd_int, int src1_int, int src2_int);
32
33 void createADDi(Block *b, int rd_int, int src1_int, int imediato);
34 void createSUBi(Block *b, int rd_int, int src1_int, int imediato);
35 void createMULi(Block *b, int rd_int, int src1_int, int imediato);
36 void createDIVi(Block *b, int rd_int, int src1_int, int imediato);
37
38 void createDeclaration(Block *b, int rd_int);
39 //Difícil no RISC: 2 instrucoes
40 //void createIsNotEqual(Block *b, int rd_int, int src1_int, int src2_int);
41
42 void createAnd(Block *b, int rd_int, int src1_int, int src2_int);
43 void createAndi(Block *b, int rd_int, int src1_int, int imediato);
44 void createOr(Block *b, int rd_int, int src1_int, int src2_int);
45 //Difícil no RISC: 2 instrucoes
46 void createNot(Block *b, int rd_int, int src1_int, int src2_int);
47
48 void createSlt(Block *b, int rd_int, int src1_int, int src2_int);
49 void createSgt(Block *b, int rd_int, int src1_int, int src2_int);
50
51 /*void createSltEqual(Block *b, int rd_int, int src1_int, int src2_int);
52 void createSgtEqual(Block *b, int rd_int, int src1_int, int src2_int);
53 */
54
55 void createBeqi(Block *b, int src1_int, int src2_int, char * label);
56 void createBeqEndi(Block *b, int src1_int, int src2_int, int endereco);
57
58 void createLabel(Block *b, char * label);
59 void createLabelInt(Block *b, int labelint);

```

## Apêndice J - *montador\_lexer.c*

```

1  #include "montador_lexer.h"
2
3  Block createBlock(){
4      Block b;
5      b.num_inst = 0;
6      return b;
7  }
8
9  void imprimeRISCV(Block *b){
10     printf("RISC\n");
11
12     FILE *risc_file = NULL;
13
14     char buffer[1000];
15
16     risc_file = fopen("cod.asm", "w"); // abrindo para escrita
17     if(risc_file == NULL){
18         printf("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated'\n");
19     }
20
21     for(int i=0; i<b->num_inst;i++){
22         if(b->insts[i].op == "ADD" || b->insts[i].op == "MUL" || b->insts[i].op == "DIV" || b->insts[i].op == "AND" || b->insts[i].op
23 == "OR" || b->insts[i].op == "SLT" || b->insts[i].op == "SGT"){
24             printf("%s t%d, t%d, t%d\n", b->insts[i].op, b->insts[i].rd, b->insts[i].src1, b->insts[i].src2);
25             sprintf(buffer, "%s t%d, t%d, t%d\n", b->insts[i].op, b->insts[i].rd, b->insts[i].src1, b->insts[i].src2);
26             fputs(buffer, risc_file);
27         }
28         if(b->insts[i].op == "SUB"){
29             printf("ADD t%d, t%d, -t%d\n", b->insts[i].rd, b->insts[i].src1, b->insts[i].src2);
30             sprintf(buffer, "ADD t%d, t%d, -t%d\n", b->insts[i].rd, b->insts[i].src1, b->insts[i].src2);
31             fputs(buffer, risc_file);
32         }
33         if(b->insts[i].op == "ADDi" || b->insts[i].op == "MULi" || b->insts[i].op == "DIVi" || b->insts[i].op == "ANDi"){
34             printf("%s t%d, t%d, %d\n", b->insts[i].op, b->insts[i].rd, b->insts[i].src1, b->insts[i].imediato);
35             sprintf(buffer, "%s t%d, t%d, %d\n", b->insts[i].op, b->insts[i].rd, b->insts[i].src1, b->insts[i].imediato);
36             fputs(buffer, risc_file);
37         }
38         if(b->insts[i].op == "SUBi"){
39             printf("ADDi t%d, t%d, %d\n", b->insts[i].rd, b->insts[i].src1, b->insts[i].imediato);
40             sprintf(buffer, "ADDi t%d, t%d, %d\n", b->insts[i].rd, b->insts[i].src1, b->insts[i].imediato);
41             fputs(buffer, risc_file);
42         }
43         if(b->insts[i].op == "BEQi"){
44             printf("BEQi t%d, t%d, %s\n", b->insts[i].src1, b->insts[i].imediato, b->insts[i].label);
45             sprintf(buffer, "BEQi t%d, t%d, %s\n", b->insts[i].src1, b->insts[i].imediato, b->insts[i].label);
46             fputs(buffer, risc_file);
47         }
48         if(b->insts[i].op == "BEQiend"){
49             printf("BEQi t%d, t%d, %d\n", b->insts[i].src1, b->insts[i].imediato, b->insts[i].endereco);
50             sprintf(buffer, "BEQi t%d, t%d, %d\n", b->insts[i].src1, b->insts[i].imediato, b->insts[i].endereco);
51             fputs(buffer, risc_file);
52         }
53         if(b->insts[i].op == "label"){
54             printf("%s :\n", b->insts[i].label);
55             sprintf(buffer, "%s :\n", b->insts[i].label);
56             fputs(buffer, risc_file);
57         }
58         if(b->insts[i].op == "labelint"){
59             printf("%d :\n", b->insts[i].labelint);
60             sprintf(buffer, "%d :\n", b->insts[i].labelint);
61             fputs(buffer, risc_file);
62         }
63     }
64     fclose(risc_file);

```

```

65 }
66
67 void imprimeLLVM(Block *b){
68     FILE *llvm_file = NULL;
69
70     llvm_file = fopen("cod.ll", "w"); // abrindo para escrita
71     if(llvm_file == NULL){
72         printf("Nao foi possivel abrir para escrita o arquivo 'arq_three_address_code_generated'\n");
73     }
74
75     char * target_dalila = "\"x86_64-unknown-linux-gnu\"";
76     char * so_dalila = "!1 = !{!\"clang version 11.0.0 (Fedora 11.0.0-2.fc33)\"} ";
77     char * target_daniel = "\"x86_64-pc-linux-gnu\"";
78     char * so_daniel = "!1 = !{!\"clang version 10.0.0-4ubuntu1 \"}";
79
80     char * attributes0 = "attributes #0 = { noline nounwind optnone uwtable \"correctly-rounded-divide-sqrt-fp-math\"=\"false\"
81 \"disable-tail-calls\"=\"false\" \"frame-pointer\"=\"all\" \"less-precise-fpmad\"=\"false\" \"min-legal-vector-width\"=\"0\"
82 \"no-infs-fp-math\"=\"false\" \"no-jump-tables\"=\"false\" \"no-nans-fp-math\"=\"false\" \"no-signed-zeros-fp-math\"=\"false\"
83 \"no-trapping-math\"=\"true\" \"stack-protector-buffer-size\"=\"8\" \"target-cpu\"=\"x86-64\"
84 \"target-features\"=\"+cx8,+fxsr,+mmx,+sse,+sse2,+x87\" \"unsafe-fp-math\"=\"false\" \"use-soft-float\"=\"false\" }";
85
86     char * attributes1 = "attributes #1 = { \"correctly-rounded-divide-sqrt-fp-math\"=\"false\" \"disable-tail-calls\"=\"false\"
87 \"frame-pointer\"=\"all\" \"less-precise-fpmad\"=\"false\" \"no-infs-fp-math\"=\"false\" \"no-nans-fp-math\"=\"false\"
88 \"no-signed-zeros-fp-math\"=\"false\" \"no-trapping-math\"=\"true\" \"stack-protector-buffer-size\"=\"8\"
89 \"target-cpu\"=\"x86-64\" \"target-features\"=\"+cx8,+fxsr,+mmx,+sse,+sse2,+x87\" \"unsafe-fp-math\"=\"false\"
90 \"use-soft-float\"=\"false\" }";
91
92
93     char * flags = "!llvm.module.flags = !{!0}";
94     char * ident = "!llvm.ident = !{!1}";
95     char * wchar_size = "!0 = !{i32 1, !\"wchar_size\", i32 4}";
96     char * codigo = "define dso_local i32 @main() #0 {";
97     char * codigo_ret = "    ret i32 0\n";
98
99     char * declare = "declare dso_local i32 @printf(i8*, ...) #1";
100
101     char buffer[1000];
102     printf("\ntarget triple = %s\n\n",target_dalila);
103     sprintf(buffer, "\ntarget triple = %s\n\n",target_dalila);
104     fputs(buffer, llvm_file);
105
106
107     printf("@.str = private unnamed_addr constant [3 x i8] c\"%%d\\0A\\", align 1\n\n");
108     sprintf(buffer, "@.str = private unnamed_addr constant [3 x i8] c\"%%d\\0A\\", align 1\n\n");
109     fputs(buffer, llvm_file);
110
111     printf("%s\n",codigo);
112     sprintf(buffer, "%s", codigo);
113     fputs(buffer, llvm_file);
114
115     int t[100];
116     int declara[100];
117     int k = 1;
118
119     printf("    %%1 = alloca i32, align 4\n");
120     sprintf(buffer, "    %%1 = alloca i32, align 4\n");
121     fputs(buffer, llvm_file);
122
123     printf("    store i32 0, i32* %%1, align 4\n");
124     sprintf(buffer, "    store i32 0, i32* %%1, align 4\n");
125     fputs(buffer, llvm_file);
126     k++;
127
128
129     //TODO: imprimir instrucoes
130     for(int i=0; i<b->num_inst;i++){
131         if(b->insts[i].op == "DEC"){
132             t[b->insts[i].rd] = k;
133             declara[b->insts[i].rd] = k;

```

```

134     printf(" %%%d = alloca i32, align 4\n",k);
135     sprintf(buffer, " %%%d = alloca i32, align 4\n",k);
136     fputs(buffer, llvm_file);
137
138     printf(" store i32 0, i32* %%%d, align 4\n",k);
139     sprintf(buffer, " store i32 0, i32* %%%d, align 4\n",k);
140     fputs(buffer, llvm_file);
141
142     k++;
143 }
144 if(b->insts[i].op == "ADD"){ // %6 = add nsw i32 %4, %5
145     printf(" %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
146     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
147     fputs(buffer, llvm_file);
148     k++;
149
150     printf(" %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src2]);
151     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src2]);
152     fputs(buffer, llvm_file);
153     k++;
154
155     t[b->insts[i].rd] = k;
156
157     printf(" %%%d = add nsw i32 %%%d, %%%d\n", k, k-1, k-2);
158     sprintf(buffer, " %%%d = add nsw i32 %%%d, %%%d\n", k, k-1, k-2);
159     fputs(buffer, llvm_file);
160
161     printf(" store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
162     sprintf(buffer, " store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
163     fputs(buffer, llvm_file);
164     k++;
165 }
166 if(b->insts[i].op == "ADDi"){ // %4 = add nsw i32 %3, 99
167     printf(" %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
168     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
169     fputs(buffer, llvm_file);
170     k++;
171
172     t[b->insts[i].rd] = k;
173
174     printf(" %%%d = add nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
175     sprintf(buffer, " %%%d = add nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
176     fputs(buffer, llvm_file);
177
178     printf(" store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
179     sprintf(buffer, " store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
180     fputs(buffer, llvm_file);
181     k++;
182 }
183 if(b->insts[i].op == "SUB"){ // %6 = add nsw i32 %4, %5
184     printf(" %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
185     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src1]);
186     fputs(buffer, llvm_file);
187     k++;
188     printf(" %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src2]);
189     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n",k, declara[b->insts[i].src2]);
190     fputs(buffer, llvm_file);
191     k++;
192
193     t[b->insts[i].rd] = k;
194
195     printf(" %%%d = sub nsw i32 %%%d, %%%d\n", k, k-2, k-1);
196     sprintf(buffer, " %%%d = sub nsw i32 %%%d, %%%d\n", k, k-2, k-1);
197     fputs(buffer, llvm_file);
198     printf(" store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
199     sprintf(buffer, " store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
200     fputs(buffer, llvm_file);
201     k++;
202 }

```



```

203 if(b->insts[i].op == "SUBi"){ // %4 = add nsw i32 %3, 99
204     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
205     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
206     fputs(buffer, llvm_file);
207     k++;
208
209     t[b->insts[i].rd] = k;
210
211     printf(" %%%d = sub nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
212     sprintf(buffer, " %%%d = sub nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
213     fputs(buffer, llvm_file);
214     printf(" store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
215     sprintf(buffer, " store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
216     fputs(buffer, llvm_file);
217     k++;
218 }
219 if(b->insts[i].op == "DIV"){ // %6 = add nsw i32 %4, %5
220     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
221     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
222     fputs(buffer, llvm_file);
223     k++;
224     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src2]);
225     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src2]);
226     fputs(buffer, llvm_file);
227     k++;
228
229     t[b->insts[i].rd] = k;
230
231     printf(" %%%d = sdiv i32 %%%d, %%%d\n", k, k-2, k-1);
232     sprintf(buffer, " %%%d = sdiv i32 %%%d, %%%d\n", k, k-2, k-1);
233     fputs(buffer, llvm_file);
234     printf(" store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
235     sprintf(buffer, " store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
236     fputs(buffer, llvm_file);
237     k++;
238 }
239 if(b->insts[i].op == "DIVi"){ // %4 = add nsw i32 %3, 99
240     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
241     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
242     fputs(buffer, llvm_file);
243     k++;
244
245     t[b->insts[i].rd] = k;
246
247     printf(" %%%d = sdiv i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
248     sprintf(buffer, " %%%d = sdiv i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
249     fputs(buffer, llvm_file);
250     printf(" store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
251     sprintf(buffer, " store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
252     fputs(buffer, llvm_file);
253     k++;
254 }
255 if(b->insts[i].op == "MUL"){ // %6 = add nsw i32 %4, %5
256     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
257     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
258     fputs(buffer, llvm_file);
259     k++;
260     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src2]);
261     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src2]);
262     fputs(buffer, llvm_file);
263     k++;
264
265     t[b->insts[i].rd] = k;
266
267     printf(" %%%d = mul nsw i32 %%%d, %%%d\n", k, k-1, k-2);
268     sprintf(buffer, " %%%d = mul nsw i32 %%%d, %%%d\n", k, k-1, k-2);
269     fputs(buffer, llvm_file);
270     printf(" store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);
271     sprintf(buffer, " store i32 %%%d, i32* %%%d, align 4\n", k, declara[b->insts[i].rd]);

```

```

272     fputs(buffer, llvm_file);
273     k++;
274 }
275 if(b->insts[i].op == "MULi"){ // %4 = add nsw i32 %3, 99
276     printf(" %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
277     sprintf(buffer, " %%%d = load i32, i32* %%%d, align 4\n", k, declara[b->insts[i].src1]);
278     fputs(buffer, llvm_file);
279     k++;
280
281     t[b->insts[i].rd] = k;
282
283     printf(" %%%d = mul nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
284     sprintf(buffer, " %%%d = mul nsw i32 %%%d, %d\n", k, k-1, b->insts[i].imediato);
285     fputs(buffer, llvm_file);
286     printf(" store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
287     sprintf(buffer, " store i32 %d, i32* %%%d, align 4\n", b->insts[i].imediato, declara[b->insts[i].rd]);
288     fputs(buffer, llvm_file);
289     k++;
290 }
291 }
292
293 printf(" %%%d = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 %%%d)\n",
294 k, k-1);
295 sprintf(buffer, " %%%d = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32
296 %%%d)\n", k, k-1);
297 fputs(buffer, llvm_file);
298
299 printf("%s\n", codigo_ret);
300 sprintf(buffer, "%s\n", codigo_ret);
301 fputs(buffer, llvm_file);
302
303 printf("\n%s\n", declare);
304 sprintf(buffer, "\n%s\n", declare);
305 fputs(buffer, llvm_file);
306
307 printf("\n%s\n", attributes0);
308 sprintf(buffer, "\n%s\n", attributes0);
309 fputs(buffer, llvm_file);
310
311 printf("\n%s\n", attributes1);
312 sprintf(buffer, "\n%s\n", attributes1);
313 fputs(buffer, llvm_file);
314
315 printf("%s\n", flags);
316 sprintf(buffer, "%s\n", flags);
317 fputs(buffer, llvm_file);
318
319 printf("%s\n", ident);
320 sprintf(buffer, "%s\n", ident);
321 fputs(buffer, llvm_file);
322
323 printf("%s\n", wchar_size);
324 sprintf(buffer, "%s\n", wchar_size);
325 fputs(buffer, llvm_file);
326
327 printf("%s\n", so_dalila);
328 sprintf(buffer, "%s\n", so_dalila);
329 fputs(buffer, llvm_file);
330
331 fclose(llvm_file);
332 }
333
334 void createDeclaration(Block *b, int rd_int){
335     if(b->num_inst < 50){
336         b->insts[b->num_inst].op = "DEC";
337         b->insts[b->num_inst].rd = rd_int;
338         b->num_inst++;
339     } else {
340         printf("You have achived the maximum size of the instruction set!");

```

```

341     }
342 }
343 //and rd, a, b
344 void createAnd(Block *b, int rd_int, int src1_int, int src2_int){
345     if(b->num_inst < 50){
346         b->insts[b->num_inst].op = "AND";
347         b->insts[b->num_inst].rd = rd_int;
348         b->insts[b->num_inst].src1 = src1_int;
349         b->insts[b->num_inst].src2 = src2_int;
350         b->num_inst++;
351     } else {
352         printf("You have achived the maximum size of the instruction set!");
353     }
354 }
355
356 void createADD(Block *b, int rd_int, int src1_int, int src2_int){
357     if(b->num_inst < 50){
358         b->insts[b->num_inst].op = "ADD";
359         b->insts[b->num_inst].rd = rd_int;
360         b->insts[b->num_inst].src1 = src1_int;
361         b->insts[b->num_inst].src2 = src2_int;
362         b->num_inst++;
363     } else {
364         printf("You have achived the maximum size of the instruction set!");
365     }
366 }
367
368 void createSUB(Block *b, int rd_int, int src1_int, int src2_int){
369     if(b->num_inst < 50){
370         b->insts[b->num_inst].op = "SUB";
371         b->insts[b->num_inst].rd = rd_int;
372         b->insts[b->num_inst].src1 = src1_int;
373         b->insts[b->num_inst].src2 = src2_int;
374         b->num_inst++;
375     } else {
376         printf("You have achived the maximum size of the instruction set!");
377     }
378 }
379
380 void createMUL(Block *b, int rd_int, int src1_int, int src2_int){
381     if(b->num_inst < 50){
382         b->insts[b->num_inst].op = "MUL";
383         b->insts[b->num_inst].rd = rd_int;
384         b->insts[b->num_inst].src1 = src1_int;
385         b->insts[b->num_inst].src2 = src2_int;
386         b->num_inst++;
387     } else {
388         printf("You have achived the maximum size of the instruction set!");
389     }
390 }
391
392 void createDIV(Block *b, int rd_int, int src1_int, int src2_int){
393     if(b->num_inst < 50){
394         b->insts[b->num_inst].op = "DIV";
395         b->insts[b->num_inst].rd = rd_int;
396         b->insts[b->num_inst].src1 = src1_int;
397         b->insts[b->num_inst].src2 = src2_int;
398         b->num_inst++;
399     } else {
400         printf("You have achived the maximum size of the instruction set!");
401     }
402 }
403
404 void createADDi(Block *b, int rd_int, int src1_int, int immediato){
405     if(b->num_inst < 50){
406         b->insts[b->num_inst].op = "ADDi";
407         b->insts[b->num_inst].rd = rd_int;
408         b->insts[b->num_inst].src1 = src1_int;
409         b->insts[b->num_inst].imediato = immediato;

```

```

410     b->num_inst++;
411 } else {
412     printf("You have achived the maximum size of the instruction set!");
413 }
414 }
415
416 void createSUBi(Block *b, int rd_int, int src1_int, int imediato){
417     if(b->num_inst < 50){
418         b->insts[b->num_inst].op = "SUBi";
419         b->insts[b->num_inst].rd = rd_int;
420         b->insts[b->num_inst].src1 = src1_int;
421         b->insts[b->num_inst].imediato = imediato;
422         b->num_inst++;
423     } else {
424         printf("You have achived the maximum size of the instruction set!");
425     }
426 }
427
428 void createMULi(Block *b, int rd_int, int src1_int, int imediato){
429     if(b->num_inst < 50){
430         b->insts[b->num_inst].op = "MULi";
431         b->insts[b->num_inst].rd = rd_int;
432         b->insts[b->num_inst].src1 = src1_int;
433         b->insts[b->num_inst].imediato = imediato;
434         b->num_inst++;
435     } else {
436         printf("You have achived the maximum size of the instruction set!");
437     }
438 }
439
440 void createDIVi(Block *b, int rd_int, int src1_int, int imediato){
441     if(b->num_inst < 50){
442         b->insts[b->num_inst].op = "DIVi";
443         b->insts[b->num_inst].rd = rd_int;
444         b->insts[b->num_inst].src1 = src1_int;
445         b->insts[b->num_inst].imediato = imediato;
446         b->num_inst++;
447     } else {
448         printf("You have achived the maximum size of the instruction set!");
449     }
450 }
451
452 void createAndi(Block *b, int rd_int, int src1_int, int imediato){
453     if(b->num_inst < 50){
454         b->insts[b->num_inst].op = "AND";
455         b->insts[b->num_inst].rd = rd_int;
456         b->insts[b->num_inst].src1 = src1_int;
457         b->insts[b->num_inst].imediato = imediato;
458         b->num_inst++;
459     } else {
460         printf("You have achived the maximum size of the instruction set!");
461     }
462 }
463
464 void createOri(Block *b, int rd_int, int src1_int, int src2_int){
465     if(b->num_inst < 50){
466         b->insts[b->num_inst].op = "OR";
467         b->insts[b->num_inst].rd = rd_int;
468         b->insts[b->num_inst].src1 = src1_int;
469         b->insts[b->num_inst].src2 = src2_int;
470         b->num_inst++;
471     } else {
472         printf("You have achived the maximum size of the instruction set!");
473     }
474 }
475
476 //Difícil no RISC: 2 instrucoes
477 void createNot(Block *b, int rd_int, int src1_int, int src2_int){
478     if(b->num_inst < 50){

```

```

479     b->insts[b->num_inst].op = "NOT";
480     b->insts[b->num_inst].rd = rd_int;
481     b->insts[b->num_inst].src1 = src1_int;
482     b->insts[b->num_inst].src2 = src2_int;
483     b->num_inst++;
484 } else {
485     printf("You have achived the maximum size of the instruction set!");
486 }
487 }
488
489 void createSlt(Block *b, int rd_int, int src1_int, int src2_int){
490     if(b->num_inst < 50){
491         b->insts[b->num_inst].op = "SLT";
492         b->insts[b->num_inst].rd = rd_int;
493         b->insts[b->num_inst].src1 = src1_int;
494         b->insts[b->num_inst].src2 = src2_int;
495         b->num_inst++;
496     } else {
497         printf("You have achived the maximum size of the instruction set!");
498     }
499 }
500
501 void createSgt(Block *b, int rd_int, int src1_int, int src2_int){
502     if(b->num_inst < 50){
503         b->insts[b->num_inst].op = "SGT";
504         b->insts[b->num_inst].rd = rd_int;
505         b->insts[b->num_inst].src1 = src1_int;
506         b->insts[b->num_inst].src2 = src2_int;
507         b->num_inst++;
508     } else {
509         printf("You have achived the maximum size of the instruction set!");
510     }
511 }
512
513
514 void createBeqi(Block *b, int src1_int, int boolean, char * label){
515     if(b->num_inst < 50){
516         b->insts[b->num_inst].op = "BEQi";
517         b->insts[b->num_inst].src1 = src1_int;
518         b->insts[b->num_inst].imediato = boolean;
519         b->insts[b->num_inst].label = label;
520         b->num_inst++;
521     } else {
522         printf("You have achived the maximum size of the instruction set!");
523     }
524 }
525
526 void createBeqEndi(Block *b, int src1_int, int boolean, int endereco){
527     if(b->num_inst < 50){
528         b->insts[b->num_inst].op = "BEQiend";
529         b->insts[b->num_inst].src1 = src1_int;
530         b->insts[b->num_inst].imediato = boolean;
531         b->insts[b->num_inst].endereco = endereco;
532         b->num_inst++;
533     } else {
534         printf("You have achived the maximum size of the instruction set!");
535     }
536 }
537
538 void createLabel(Block *b, char * label){
539     if(b->num_inst < 50){
540         b->insts[b->num_inst].op = "label";
541         b->insts[b->num_inst].label = label;
542         b->num_inst++;
543     } else {
544         printf("You have achived the maximum size of the instruction set!");
545     }
546 }
547

```

```
548 void createLabelInt(Block *b, int labelint){
549     if(b->num_inst < 50){
550         b->insts[b->num_inst].op = "labelint";
551         b->insts[b->num_inst].labelint = labelint;
552         b->num_inst++;
553     } else {
554         printf("You have achived the maximum size of the instruction set!");
555     }
556 }
```