Everton da Silva Coelho R.A.:101937

Implementação de um compilador em python para a linguagem C-

São José dos Campos - Brasil

Julho de 2018

Everton da Silva Coelho R.A.:101937

Implementação de um compilador em python para a linguagem C-

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

Docente: Prof. Dr. Luiz Eduardo Galvão Martins

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Julho de 2018

Sumário

1 Introdução

Vivendo na época atual considerada a era da informação, não é difícil encontrar diversas máquinas capazes de tratar grandes quantidades de dados gerados pela interação social, por imagens, sensores, entre outros. Essas máquinas responsáveis pelo armazenamento, processamento e cálculo dos dados são chamadas de computadores. No cerne dos computadores estão os processadores, responsáveis por transformar os dados contidos na máquina em informação para o usuário.

A princípio os computadores eram programados diretamente com o código de máquina, o que é um processo lento e custoso para os programadores e com altas chances de erros. Para contornar essas dificuldades projetou-se uma maneira mais fácil de assimilar a linguagem de máquina, resultando em um compilador. O compilador faz a tradução de uma linguagem textual de fácil entendimento para uma linguagem de máquina, essa ferramenta é necessária para grande parte das aplicações de alto nível atuais.

Com o objetivo de entender o funcionamento foi desenvolvido, no Laboratório de Sistemas Computacionais: Compiladores, um compilador para C- utilizando a linguagem python. O compilador traduz um código em C- para o processador projetado em laboratórios anteriores, atentendo as particularidades do mesmo. Neste relátório são abordados os passos do desemvolvimento que estão organizados da seguinte maneira:

1. 2. O Processador: Neste tópico é apresentado o processador desenvolvido no Laboratório de sistemas computacionais: Arquitetura e Organização de computadores, suas especificações e alguns conceitos básicos para o entendimento do mesmo.
2. 3. Compilador(Fase de Análise): Neste encontra-se como foram desenvolvidas a análise léxica, sintática e semântica do compilador.
3. 4. Compilador(Fase de Síntese): Criação do código intermediário, e passagem para o assembly(objeto) e conversão do assembly para o código binário que será posto no processador.
4. 5. Exemplos Gerados: Códigos exemplos compilados.
5. 6. Conclusão: Considerações finais a respeito do desenvolvimento do projeto.

2 O Processador

2.1 Sistemas computacionais

Um sistema computacional é um conjunto de dispositivos eletrônicos que são utilizados para processar informações, compostos pela união de hardware e software. Um exemplo de sistema computacional é um celular, que possui funções para tirar fotos, gravar vídeos, sons e comunicação com outros dispositivos, ou seja, ele captura os dados dos sensores e transforma em informação, possibilitando a comunicação com o usuário(1).

Focando no Hardware de um sistema computacional, ele consiste em três principais partes:

1. O processador ou CPU: é a unidade responsável pela execução de instruções requisitadas por um sistema, como cálculos aritméticos, lógicos, operações de entrada e saída e manipulação de dados.
2. Memória principal: é o dispositivo capaz de armazenar dados gerais do sistema. Basicamente o processador acessa a memória para buscar informações e depois de processá-las manda sinais para outras partes do sistema realizando atividades pedidas.
3. Módulo de entrada e saída: responsável pela interface entre a máquina e o usuário, ele facilita a comunicação do programador com o sistema computacional.

2.2 *Central Processor Unit* (CPU)

Responsável por executar as instruções, o processador é constituído das seguintes partes:

- ALU (Unidade Lógica Aritmética): realiza cálculos como soma e subtração e exeuta comparações lógicas como *and* e *or*.
- Memória de instruções: armazena as instruções que serão executadas.
- Banco de registradores: memória de acesso rápido que armazena os operandos da instrução;
- Unidade de Controle: controla as outras partes do processador para a execução correta das instruções.
- *Program counter*: Armazena o endereço para a próxima instrução a ser executada(2).

2.3 MIPS

MIPS (*Microprocessor Without Interlocked Pipeline Stages*) é um processador de arquitetura RISC que trabalha de forma monocíclica, ou seja, ele executa uma instrução por ciclo de clock. Devido a isto todas as instruções possuem tempo igual de execução. Como é um processador RISC ele herda algumas de suas características: Quantidade reduzida de instruções, instruções de tamanho fixo, pouca redundância de instruções, data path mais simples(2).

2.4 Tipos de endereçamento

Para acessar os dados o processador precisa saber onde esses dados estão localizados, uma forma de saber o caminho desses dados é definindo um tipo de endereçamento, alguns tipos são listados a seguir:

1. Endereçamento por registrador: o operando é o conteúdo de um registrador especificado na instrução.
2. Endereçamento imediato: o conteúdo do operando está incorporado na instrução.
3. Endereçamento indireto a registrador: o operando está no endereço especificado dentro de um registrador repassado pela instrução(3).

2.5 Verilog

Verilog é uma linguagem de descrição de hardware(Hardware Description Language - HDL) que permite descrever sistemas digitais, analógicos ou híbridos em vários níveis de abstração(4).

O Verilog difere das outras linguagens pela maneira como é executado. Diferente de uma linguagem procedural um projeto Verilog consiste na separação hierárquica de módulos que contém conexões e registradores. Esses módulos contém blocos que são executados em paralelo, há também a possibilidade de executar processos sequenciais dentro de blocos "begin/end"(5).

2.6 FPGA

FPGA ou Field Programmable Gate Array é um tipo de circuito integrado que possui uma lógica programável, pode ser configurado após a sua fabricação, internamente possui um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. É constituído basicamente de três partes:

2.7 Quartus

Quartus prime 16.1 é um software de design produzido pela Altera. Ele permite a análise e síntese de linguagem de descrição de hardware e desenhos, permite que o desenvolvedor compile seus projetos, realize análise de timing, simule a reação de um projeto a diferentes estímulos, e configure o dispositivo de destino. O Quartus inclui uma implementação do VHDL e Verilog para descrição de hardware, edição visual de circuitos lógicos, e simulação de formas de onda(6).

2.8 Processador Desenvolvido

Tomando como base a arquitetura MIPS, o processador utilizado, chamado de ProcessorE, trabalha de forma monocíclica (uma instrução por ciclo de clock) e possui instruções de tamanho fixo e realiza operações principalmente com registradores, de forma que seu endereçamento é direto por registrador. Possui instruções lógicas e aritméticas, de controle, assim como instruções para manipular a memória e de entrada e saída de dados. O formato das instruções pode ser visto na (Tabela 1).

Tabela 1 – Formato das Instruções

F1					
Opcode	RD	RS	RT	-	-
[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
F2					
Opcode	RD	RS	IMT		
[31:26]	[25:21]	[20:16]	[15:0]		
F3					
Opcode	RD	endereço/IMT/IO			
[31:26]	[25:21]	[20:0]			
F4					
Opcode	-				
[31:26]	[25:0]				

Fonte: O Autor

Cada instrução possui o tamanho de 32 bits com 6 bits destinados à opcode, o que possibilita a criação de 64 instruções diferentes, tornando possível a expansão para futuras instruções. Dos 32 bits, blocos com 5 bits são destinados ao endereço dos registradores; os 5 bits possibilitam ter acesso aos 32 registradores de uso geral, pois com 5 bits obtemos 32 endereços distintos.

O processador possui 25 instruções básicas. Essas instruções foram classificadas de acordo com sua finalidade e estão descritas na Tabela 2.

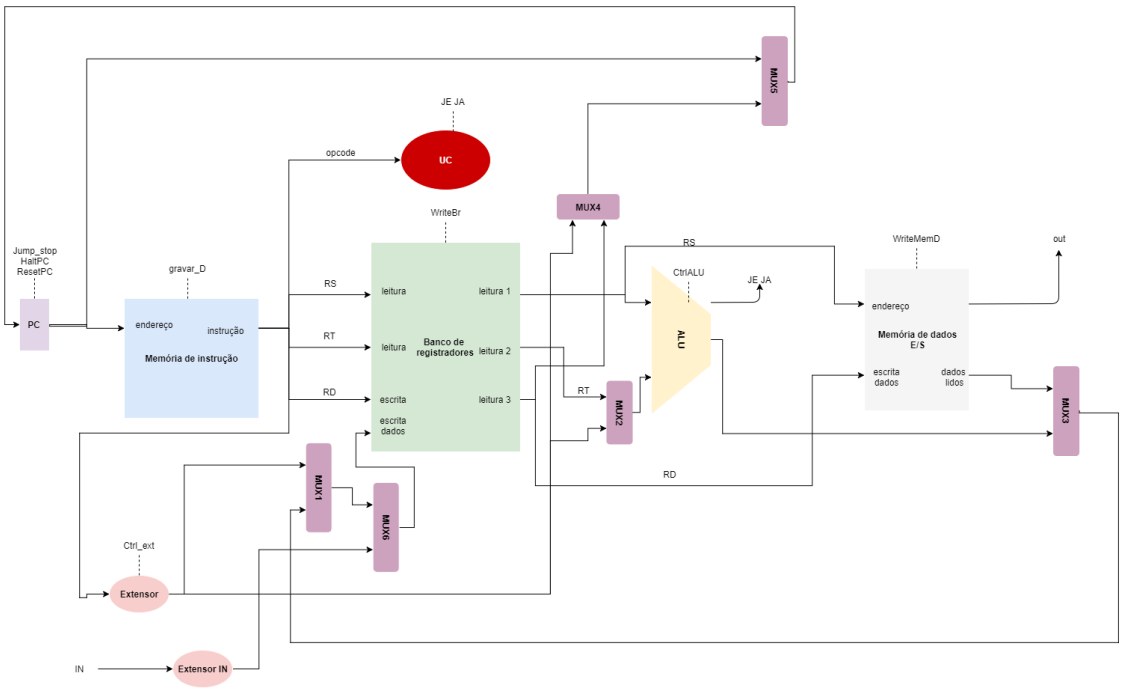
Abaixo, na (Figura 5), está o esquemático do processador referenciado.

Tabela 2 – Conjunto de instruções

Tipo	Instrução	Opcode Dec	Opcode Bin	Operação	Formato
Aritiméticas	ADD	000001	000001	RD <= RS + RT	F1
	ADDI	000002	000010	RD <= RS + IMT	F2
	SUB	000003	000011	RD <= RS - RT	F1
	SUBI	000004	000100	RD <= RS - IMT	F2
	MULT	000005	000101	RD <= RS * RT	F1
Lógicas	AND	000006	000110	RD <= RS and RT	F1
	OR	000007	000111	RD <= RS or RT	F1
	XOR	000008	001000	RD <= RS xor RT	F1
	NOT	000009	001001	RD <= not(RS)	F2
Deslocamento	SHR	000010	001010	RD <= RS »RT	F1
	SHL	000011	001011	RD <= RS «RT	F1
Movimentação de dados	LOAD	000012	001100	RD <= mem_raw[RS]	F2
	LOADI	000013	001101	RD <= IMT	F3
	STORE	000014	001110	mem_raw[RS]<= RD	F2
	MOVE	000015	001111	RD <= RS	F2
Desvios	JMPI	000016	010000	PC <= IMT	F3
	JMP	000017	010001	PC <= RD	F3
	JE	000018	010010	if RS == RT, PC <= RD	F1
	JNE	000019	010011	if RS != RT, PC <= RD	F1
	JA	000020	010100	if RS >RT, PC <= RD	F1
	JNA	000021	010101	if RS =<RT, PC <= RD	F1
Entrada e saída	IN	000022	010110	RD <= entrada de dados	F3
	OUT	000023	010111	saída de dados <= RD	F3
Controle	NOP	000024	011000	nenhuma ação	F4
	HALT	000025	011001	parar processamento	F4

Fonte: O Autor

Figura 1 – Esquematico ProcessorE



Fonte: O autor

3 Compilador

Um compilador é uma aplicação que traduz um programa escrito em uma linguagem para uma outra linguagem geralmente de mais baixo nível(7). O compilador projetado e desenvolvido neste laboratório faz a conversão de um programa em linguagem c- para a linguagem de máquina respectiva ao processador usado. O compilador pode ser dividido em vários blocos que consistem na Fase de Análise: Análise Léxica, Análise Sintática, Análise Semântica, e Fase de Síntese: Geração do código intermediário, Geração do código *Assembly*, Geração do Código Executável.

3.1 Fase de Análise

Na fase se análise utilizou-se o ANTLR (*ANother Tool for Language Recognition*) que é uma ferramenta de leitura e processamento de texto que a partir de uma gramática consegue contruir e analisar arvores sintáticas.

3.1.1 Análise Léxica

Na análise léxica é feita a varredura do código para a separação e verificação de *tokens*. Através de expressões regulares os *tokens* são separados e analisados, retornando se é uma sequência de caracteres válida ou não. Nessa fase é feito o reconhecimento de palavras reservadas(if, else, while), simbolos aritméticos e caracteres especiais. As expressões regulares são postas no mesmo arquivo da gramática(subseção 3.1.2) para que o ANTLR construa a arvore.

Os *tokens* gerados por um programa teste podem ser vistos a seguir:

```
1  /* programa teste */
2  void main(void)
3  {       int x; int y;
4
5      x = 12;
6      y = 15 + x;
7  }
8  /* Tokens gerados:
9  4 : void
10 4 : main
11 4 : (
12 4 : void
13 4 : )
14 5 : {
15 5 : int
16 5 : x
17 5 : ;
18 5 : int
19 5 : y
```

```
5 : ;
7 : x
7 : =
7 : 12
7 : ;
8 : y
8 : =
8 : 15
8 : +
8 : x
8 : ;
9 : }
10 : <EOF>
*/
```

3.1.2 Análise Sintática

Na análise sintática ocorre o processo em que o compilador verifica se o programa analisado obedece as regras da gramática estabelecida. Após recuperar os *tokens* fornecidos pela análise léxica o analisador sintático compara as entradas através de derivações de acordo com a gramática. Cada derivação é uma regra da gramática representada em BNF(Formalismo de Backus-Naur), o ANTLR faz a leitura da gramática no formato BNF e após faz a derivação das regras utilizando os *tokens* de entrada.

Gramática para cminus:

```
grammar cminus;


/* Parser Rules */

 programa
    : (decl+=declaracao)+
    ;


  declaracao
    : var_declaracao
    | fun_declaracao
    ;

  var_declaracao
    : tipo_especificador ID SEMI
    | tipo_especificador ID LSBRACKET NUM RSBRACKET SEMI
    ;

  tipo_especificador
    : INT
    | VOID
    ;

  fun_declaracao
    : tipo_especificador ID LPAREN params RPAREN composto_decl
    ;

  params
```

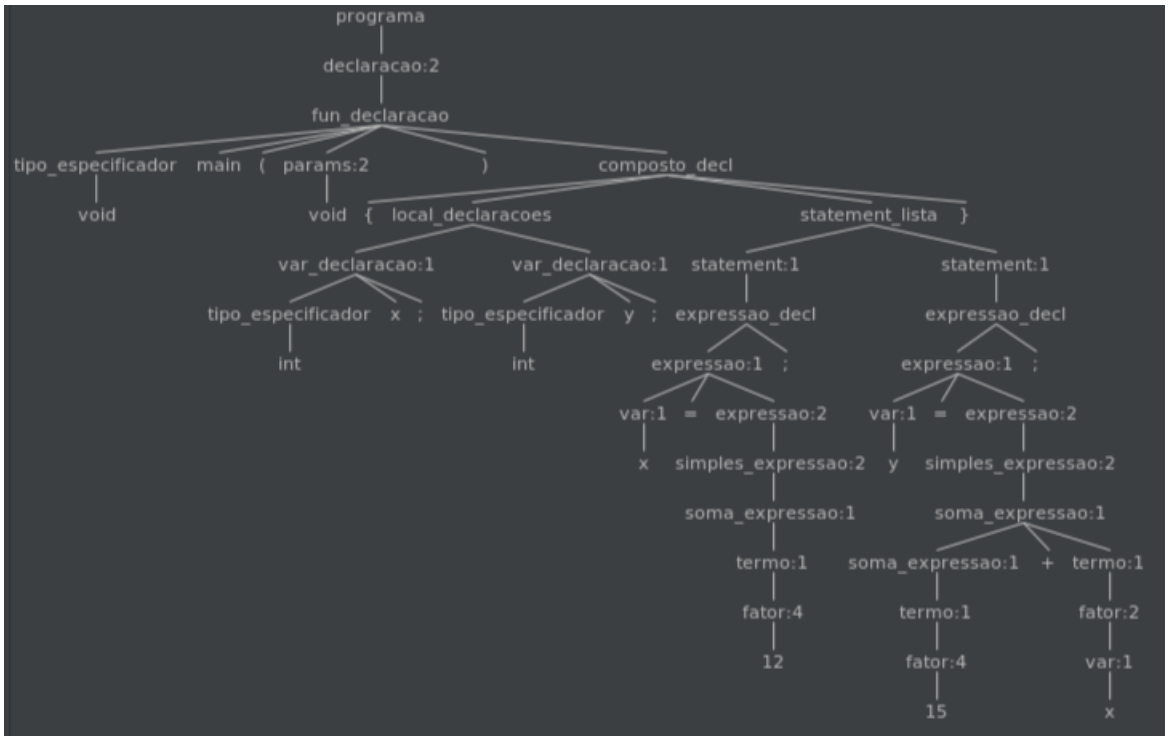
```
31    : param_lista
32    | VOID
33    ;
34
35  param_lista
36    : param_lista COMMA param
37    | param
38    ;
39
40  param
41    : tipo_especificador ID
42    | tipo_especificador ID LSBRACKET RSBRACKET
43    ;
44
45  composto_decl
46    : LCBRACKET (l_decl+=local_declaracoes)* (stm_list+=statement_lista)* RCBRACKET
47    ;
48
49  local_declaracoes
50    : (var_decl+=var_declaracao)+
51    ;
52
53  statement_lista
54    : (stms+=statement)+
55    ;
56
57  statement
58    : expressao_decl
59    | composto_decl
60    | selecao_decl
61    | iteracao_decl
62    | retorno_decl
63    ;
64
65  expressao_decl
66    : expressao? SEMI
67    ;
68
69  selecao_decl
70    : IF LPAREN condicao=expressao RPAREN LCBRACKET corpoIF+=statement* RCBRACKET (ELSE
        LCBRACKET corpoElse+=statement* RCBRACKET)?
71    ;
72
73  iteracao_decl
74    : WHILE LPAREN expressao RPAREN statement
75    ;
76
77  retorno_decl
78    : RETURN SEMI
79    | RETURN expressao SEMI
80    ;
81
82  expressao
83    : var ASSIGN expressao
84    | simples_expressao
85    ;
86
87  var
88    : ID
89    | ID LSBRACKET expressao RSBRACKET
```

```
90      ;
91
92    simples_expressao
93      : esquerda=soma_expressao relacional=(LETHAN| LT| GT| GETHAN| EQ| DF) direita=
            soma_expressao
94      | operacao=soma_expressao
95      ;
96
97    soma_expressao
98      : soma_expressao op=('+'|'-') termo
99      | termo
100     ;
101
102
103   termo
104     : termo op=('/'|'*') fator
105     | fator
106     ;
107
108
109   fator
110     : LPAREN expressao RPAREN
111     | var
112     | ativacao
113     | NUM
114     ;
115
116   ativacao
117     : ID LPAREN (arg_list+=expressao COMMA)* (arg_list+=expressao) RPAREN
118     | ID LPAREN RPAREN
119     ;
120
121     /* Lexer Rules */
122
123       //RESERVED_WORD : 'else' | 'if' | 'int' | 'return' | 'void' | 'while' ;
124       ELSE : 'else' ;
125       IF : 'if' ;
126       INT : 'int' ;
127       RETURN : 'return' ;
128       VOID : 'void' ;
129       WHILE : 'while';
130       LETHAN : '<=' ;
131       GETHAN : '>=' ;
132       ASSIGN : '=' ;
133       EQ : '==' ;
134       DF : '!=' ;
135       LT : '<' ;
136       GT : '>' ;
137       PLUS : '+' ;
138       MINUS : '-' ;
139       TIMES : '*' ;
140       OVER : '/' ;
141       LPAREN : '(';
142       RPAREN : ')';
143       SEMI : ';' ;
144       COMMA : ',' ;
145       LCBRACKET : '{' ;
146       RCBRACKET : '}' ;
147       LSBRACKET : '[';
148       RSBRACKET : ']' ;
```

```
149
150       //tokens {ELSE, IF, INT, RETURN, VOID, WHILE}
151
152       ID : [a-zA-Z]+ ; // match identifiers
153       NUM : [0-9]+ ; // match integers
154
155       BLOCK_COMMENT: '/*' .*? '*/' -> skip
156       ;
157       //LINE_COMMENT: '//' ~[\r\n]* -> skip
158       //;
159       // Whitespace
160       WS : [ \t\r\n\f]+ -> skip ;
```

A maneira que o ANTLR utiliza para representar as derivações é através de uma árvore, conhecida como árvore sintática, como as regras da gramática são recursivas uma árvore se torna uma boa forma para representar essas derivações. Se todas as derivações ocorrerem de forma a chegar num nó folha a arvore é montada e retornada, caso contrário o programa de entrada possui algum erro sintático e esse erro é retornado.

Figura 2 – Árvore de análise sintática do programa teste



Fonte: O autor

A árvore sintática também pode ser representada de forma textual e a representação da árvore do código de cálculo do mdc entre dois números é mostrada a seguir:

```
1    rvore  sint tica do programa que calcula mdc:
2
3  Programa:{
4  || Declara  es:{
5  |||| func:{
```

```
|||||| tipo: int id: gcd
|||||| argumentos da fun   o :{
|||||||||| tipo: int  id: u
|||||||||| tipo: int  id: v
|||||| }
|||||| corpo da fun   o :{
|||||||| declaracao if:{
|||||||| condicao:{
|||||||||||| simples expressao:{
|||||||||||| esquerda:{
|||||||||||||||||||| variavel:{
|||||||||||||||||||| id: v
|||||||||||||||||||| }
|||||||||||| }
|||||||||||| ==
|||||||||||| direita:{
|||||||||||||||||| Numero: 0
|||||||||||| }
|||||||||||| }
|||||||| }
|||||||| corpo if:{
|||||||||| retorno:{
|||||||||||||| simples expressao:{
|||||||||||||||||||||| variavel:{
|||||||||||||||||||||| id: u
|||||||||||||||||||||| }
|||||||||||||| }
|||||||||| }
|||||||| }
|||||||| corpo else:{
|||||||||| retorno:{
|||||||||||||| simples expressao:{
|||||||||||||||||||||| chamada de func:{
|||||||||||||||||||||| id: gcd
|||||||||||||||||||||| argumentos:{
|||||||||||||||||||||||||||| simples expressao:{
|||||||||||||||||||||||||||||||||| variavel:{
|||||||||||||||||||||||||||||||||| id: v
|||||||||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||| simples expressao:{
|||||||||||||||||||||||||||||| operacao:{
|||||||||||||||||||||||||||||||||||| variavel:{
|||||||||||||||||||||||||||||||||||| id: u
|||||||||||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||||| -
|||||||||||||||||||||||||||||| operacao:{
|||||||||||||||||||||||||||||||||| operacao:{
|||||||||||||||||||||||||||||||||||||||| variavel:{
|||||||||||||||||||||||||||||||||||||||| id: u
|||||||||||||||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||||||||| /
|||||||||||||||||||||||||||||||||||||||| variavel:{
|||||||||||||||||||||||||||||||||||||||| id: v
|||||||||||||||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||||||||| }
|||||||||||||||||||||||||||||| *
|||||||||||||||||||||||||||||||||||||||| variavel:{
|||||||||||||||||||||||||||||||||||||||| id: v
|||||||||||||||||||||||||||||||||||||||| }
```

```
66  |||||||||||||||||||||||||||||| }
67  |||||||||||||||||||||||||||| }
68  ||||||||||||||||||||||||| }
69  |||||||||||||||||||||| }
70  |||||||||||||||||||||| }
71  ||||||||||||||| }
72  |||||||||| }
73  |||||||| }
74  |||||| }
75  |||| }
76  |||| func:{
77  |||||| tipo: void id: main
78  |||||| argumentos da fun   o :{
79  |||||||| void
80  |||||| }
81  |||||| corpo da fun   o :{
82  |||||||| tipo: int  id: x
83  |||||||| tipo: int  id: y
84  |||||||| Atribuicao:{
85  |||||||||| variavel:{
86  |||||||||| id: x
87  |||||||||| }
88  ||||||||||| simples expressao:{
89  |||||||||||||||||||| chamada de func:{
90  |||||||||||||||||||| id: input
91  |||||||||||||||||||| }
92  ||||||||||| }
93  |||||||| }
94  |||||||| Atribuicao:{
95  |||||||||| variavel:{
96  |||||||||| id: y
97  |||||||||| }
98  ||||||||||| simples expressao:{
99  |||||||||||||||||||| chamada de func:{
100 |||||||||||||||||||| id: input
101 |||||||||||||||||||| }
102 ||||||||||| }
103 |||||||| }
104 |||||||||| simples expressao:{
105 |||||||||||||||||| chamada de func:{
106 |||||||||||||||||| id: output
107 |||||||||||||||||| argumentos:{
108 |||||||||||||||||||||| simples expressao:{
109 |||||||||||||||||||||||||||||| chamada de func:{
110 |||||||||||||||||||||||||||||| id: gcd
111 |||||||||||||||||||||||||||||| argumentos:{
112 |||||||||||||||||||||||||||||||||| simples expressao:{
113 ||||||||||||||||||||||||||||||||||||||||||||| variavel:{
114 ||||||||||||||||||||||||||||||||||||||||||||| id: x
115 ||||||||||||||||||||||||||||||||||||||||||||| }
116 |||||||||||||||||||||||||||||||||| }
117 |||||||||||||||||||||||||||||||||| simples expressao:{
118 ||||||||||||||||||||||||||||||||||||||||||||| variavel:{
119 ||||||||||||||||||||||||||||||||||||||||||||| id: y
120 ||||||||||||||||||||||||||||||||||||||||||||| }
121 |||||||||||||||||||||||||||||||||| }
122 |||||||||||||||||||||||||||||| }
123 |||||||||||||||||||||||||||||| }
124 |||||||||||||||||||||| }
125 |||||||||||||||||| }
```

```
126 |||||||||||||||||| }
127 |||||||||| }
128 |||||| }
129 |||| }
130 || }
131 }
```

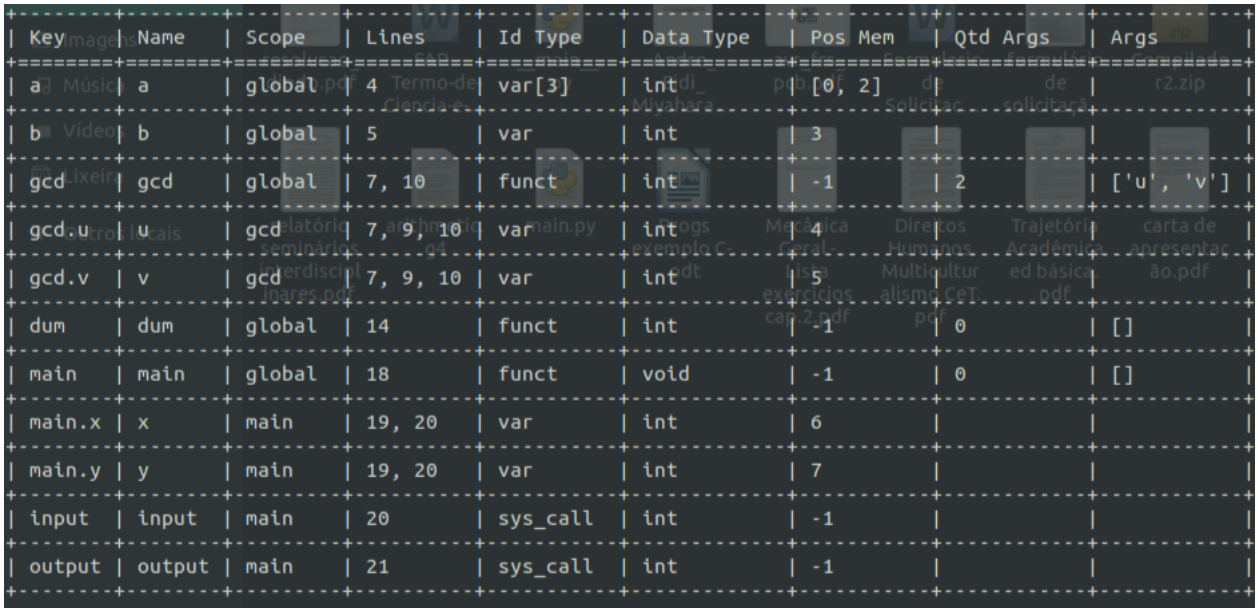
3.1.3 Análise Semântica

Após passar por duas etapas de análise o programa de entrada ainda precissa de uma análise mais específica de erros que as etapas anteriores não conseguem reconhecer. A análise semantica é responsável por reconhecer esses erros residuais, que são erros de significado do código, ou seja, atribuições erradas, variáveis não declaradas entre outros. Os principais erros que devem ser reconhecidos na análise sintática para c- são:

- Reconhecimento de variáveis não declaradas;
- Atribuições invalidas entre tipos de dados;
- Decleração inválida de variavel como *void*;
- Declaração da mesma variável mais de uma vez;
- Chamada de função não declarada;
- Função *main()* não declarada;
- Declaração inválida de uma variável com nome de função.

A análise semântica é realizada percorrendo a árvore gerada na análize sintática de forma recursiva, esse acesso a árvore é feito utilizando um padrão de projeto comportamental chamado de *visitor pattern* que permite criar novas operações sem alterar a classe dos elementos que ele acessa. Durante a visita aos nós da árvore são inseridos elementos em uma tabela *hash*, chamada de tabela de símbolos, que auxilia na detecção dos erros. Essa tabela é acessada sempre que é preciso checar e fazer uma comparação com um parametro que já foi acessado, um exemplo de tabela é mostrado logo abaixo.

Figura 3 – Tabela de símbolos código gcd



Key	Name	Scope	Lines	Id Type	Data Type	Pos Mem	Qtd Args	Args
a	a	global	4	var[3]	int	[0, 2]		
b	b	global	5	var	int	3		
gcd	gcd	global	7, 10	funct	int	-1	2	['u', 'v']
gcd.u	u	gcd	7, 9, 10	var	int	4		
gcd.v	v	gcd	7, 9, 10	var	int	5		
dum	dum	global	14	funct	int	-1	0	[]
main	main	global	18	funct	void	-1	0	[]
main.x	x	main	19, 20	var	int	6		
main.y	y	main	19, 20	var	int	7		
input	input	main	20	sys_call	int	-1		
output	output	main	21	sys_call	int	-1		

Fonte: O autor

3.2 Fase de Síntese

3.2.1 Geração do código intermediário

Para a melhor compreensão da tradução do código em c- para a linguagem de máquina é feita a geração de um código intermediário. O código intermediário neste projeto é montado através de uma lista em que cada elemento dessa lista é uma lista com 4 elementos, de forma que o primeiro elemento é a condição de tratamento para os outros 3 elementos, os outros 3 elementos podendo ser variáveis, *labels* ou registradores temporários utilizados para realizar operações. O intermediário representa as operações na forma do código de 3 endereços e é uma forma de simplificar a lógica do programa de entrada. Nele são adicionadas *labels* e tratamentos para condicionais e saltos.

Para gerar o código intermadiário a árvore sintática é percorrida novamente, através de um *visitor*, porém realizando novas operações a cada nó e adicionando uma nova linha no intermediário quando necessário. Durante o acesso são feitas comparações com os dados da tabela de símbolos de modo a manter a coerência dos dados.

3.2.2 Geração do código *Assembly*

O código assembly é o último passo antes da geração para o código em binário, que irá ser executado no processador. Para a geração do assembly a lista de intermediários é percorrida e traduzida para o conjunto de instruções do processador específico. durante

essa etapa também é feito o acesso a tabela de símbolos para a comparação dos dados e é gerado uma lista contendo as linhas do assembly.

Nessa etapa percebeu-se que modificações na arquitetura do processador e novas instruções precisavam ser adicionadas. As instruções adicionadas foram *push* - que trabalha da mesma forma da instrução *store* -, a instrução *pop* - que é semelhante a instrução *load* -, *jal* - *jump and link* -, e instruções que salvam o resultado lógico de uma comparação em um registrador: EQ - *equal* -, NEQ - *not equal* -, ABV -*above* -, NAB - *not above* -, LT = *less than* -, e NLT - *not less than* -, também foi adicionada uma instrução jei - *jump equal imediate* - que salta o *program count* para o valor do imediato se os valores de dois registradores são iguais. No processador também foi adicionada uma memória dentro da memória de dados para simular pilha e um registrador, *stp*, foi fixado para armazenar o topo desta pilha.

Devido a arquitetura do processador utilizado essa etapa deve ser realizada com cautela para que a lógica não se perca durante o processo de tradução e adaptação para as respectivas instruções do processador.

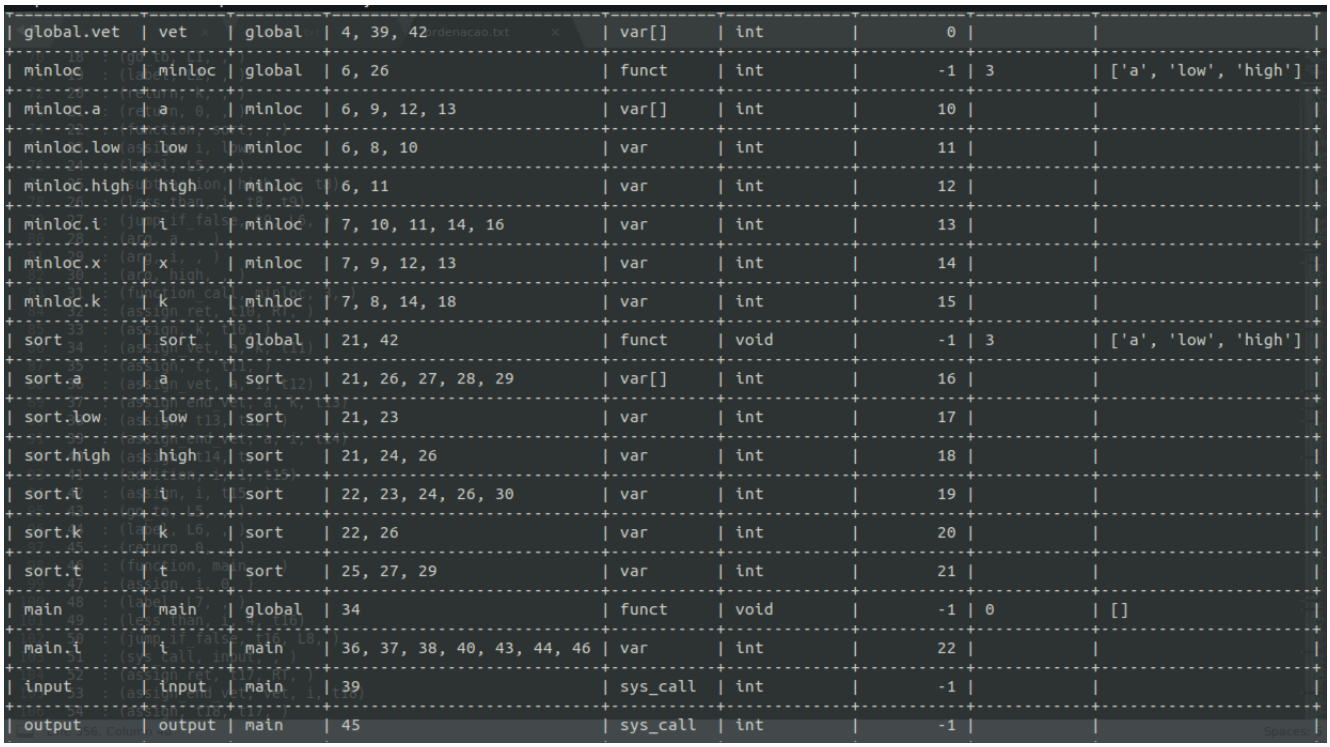
3.2.3 Gerenciamento de memória

A gerenciamento de memória do código é realizado através de duas memórias contidas na memória de dados, sendo que uma representa uma pilha e a outra uma memória estática. As funções são armazenadas na memória estática e a pilha é utilizada para empilhar paramêtros utilizados quando se tem uma chamada de função, sendo assim, permitindo recursões e chamadas de função.

4 Exemplos Gerados

Saídas geradas pelo código sort:

Figura 4 – Tabela de símbolos código sort



global.vet	vet	global	4, 39, 42	var[]	int	0		
minloc	minloc	global	6, 26	funct	int	-1	3	['a', 'low', 'high']
minloc.a	a	minloc	6, 9, 12, 13	var[]	int	10		
minloc.low	low	minloc	6, 8, 10	var	int	11		
minloc.high	high	minloc	6, 11	var	int	12		
minloc.i	i	minloc	7, 10, 11, 14, 16	var	int	13		
minloc.x	x	minloc	7, 9, 12, 13	var	int	14		
minloc.k	k	minloc	7, 8, 14, 18	var	int	15		
sort	sort	global	21, 42	funct	void	-1	3	['a', 'low', 'high']
sort.a	a	sort	21, 26, 27, 28, 29	var[]	int	16		
sort.low	low	sort	21, 23	var	int	17		
sort.high	high	sort	21, 24, 26	var	int	18		
sort.i	i	sort	22, 23, 24, 26, 30	var	int	19		
sort.k	k	sort	22, 26	var	int	20		
sort.t	t	sort	25, 27, 29	var	int	21		
main	main	global	34	funct	void	-1	0	[]
main.i	i	main	36, 37, 38, 40, 43, 44, 46	var	int	22		
input	input	main	39	sys_call	int	-1		
output	output	main	45	sys_call	int	-1		

Fonte: O autor

```
1  /* programa para ordenacao por selecao de
2     uma matriz com dez elementos. */
3
4  int vet[ 10 ];
5
6  int minloc ( int a[], int low, int high )
7  {       int i; int x; int k;
8          k = low;
9          x = a[low];
10         i = low + 1;
11         while (i < high){
12                 if (a[i] < x){
13                         x = a[i];
14                         k = i;
15                 }
16                 i = i + 1;
17         }
18         return k;
19 }
20
21 void sort( int a[], int low, int high)
22 {       int i; int k;
```

```
23          i = low;
24          while (i < high-1){
25                  int t;
26                  k = minloc(a,i,high);
27                  t = a[k];
28                  a[k] = a[i];
29                  a[i] = t;
30                  i = i + 1;
31          }
32  }
33
34  void main(void)
35  {
36          int i;
37          i = 0;
38          while (i < 4){
39                  vet[i] = input();
40                  i = i + 1;
41          }
42          sort(vet,0,4);
43          i = 0;
44          while (i < 4){
45                  output(vet[i]);
46                  i = i + 1;
47          }
48  }
49
50  /*C digo intermedi rio:*/
51
52  0  : (function, minloc, , )
53  1  : (assign, k, low, )
54  2  : (assign_vet, a, low, t1)
55  3  : (assign, x, t1, )
56  4  : (addition, low, 1, t2)
57  5  : (assign, i, t2, )
58  6  : (label, L1, , )
59  7  : (less_than, i, high, t3)
60  8  : (jump_if_false, t3, L2, )
61  9  : (assign_vet, a, i, t4)
62  10  : (less_than, t4, x, t5)
63  11  : (jump_if_false, t5, L3, )
64  12  : (assign_vet, a, i, t6)
65  13  : (assign, x, t6, )
66  14  : (assign, k, i, )
67  15  : (label, L3, , )
68  16  : (addition, i, 1, t7)
69  17  : (assign, i, t7, )
70  18  : (go_to, L1, , )
71  19  : (label, L2, , )
72  20  : (return, k, , )
73  21  : (return, 0, , )
74  22  : (function, sort, , )
75  23  : (assign, i, low, )
76  24  : (label, L5, , )
77  25  : (subtraction, high, 1, t8)
78  26  : (less_than, i, t8, t9)
79  27  : (jump_if_false, t9, L6, )
80  28  : (arg, a, , )
81  29  : (arg, i, , )
82  30  : (arg, high, , )
```

```
31  : (function_call, minloc, 3, )
32  : (assign_ret, t10, RT, )
33  : (assign, k, t10, )
34  : (assign_vet, a, k, t11)
35  : (assign, t, t11, )
36  : (assign_vet, a, i, t12)
37  : (assign_end_vet, a, k, t13)
38  : (assign, t13, t12, )
39  : (assign_end_vet, a, i, t14)
40  : (assign, t14, t, )
41  : (addition, i, 1, t15)
42  : (assign, i, t15, )
43  : (go_to, L5, , )
44  : (label, L6, , )
45  : (return, 0, , )
46  : (function, main, , )
47  : (assign, i, 0, )
48  : (label, L7, , )
49  : (less_than, i, 4, t16)
50  : (jump_if_false, t16, L8, )
51  : (sys_call, input, , )
52  : (assign_ret, t17, RT, )
53  : (assign_end_vet, vet, i, t18)
54  : (assign, t18, t17, )
55  : (addition, i, 1, t19)
56  : (assign, i, t19, )
57  : (go_to, L7, , )
58  : (label, L8, , )
59  : (arg, vet, , )
60  : (arg, 0, , )
61  : (arg, 4, , )
62  : (function_call, sort, 3, )
63  : (assign_ret, t20, RT, )
64  : (assign, i, 0, )
65  : (label, L9, , )
66  : (less_than, i, 4, t21)
67  : (jump_if_false, t21, L10, )
68  : (assign_vet, vet, i, t22)
69  : (arg, t22, , )
70  : (sys_call, output, 1, )
71  : (addition, i, 1, t24)
72  : (assign, i, t24, )
73  : (go_to, L9, , )
74  : (label, L10, , )

/* Assembly: */

     0: loadi $r0 0
     1: loadi $stp 0
     2: loadi $ra 0
     3: jmpi main
minloc:
     4: subi $stp $stp 1
     5: pop $r1 $stp
     6: loadi $rl 10
     7: store $r1 $rl
     8: subi $stp $stp 1
     9: pop $r1 $stp
     10: loadi $rl 11
     11: store $r1 $rl
```

```
143       12: subi $stp $stp 1
144       13: pop $r1 $stp
145       14: loadi $rl 12
146       15: store $r1 $rl
147       16: loadi $r1 15
148       17: loadi $rl 11
149       18: load $r2 $rl
150       19: store $r2 $r1
151       20: loadi $rl 10
152       21: load $r1 $rl
153       22: loadi $rl 11
154       23: load $r2 $rl
155       24: add $r4 $r1 $r2
156       25: load $r3 $r4
157       26: loadi $r1 14
158       27: store $r3 $r1
159       28: loadi $rl 11
160       29: load $r1 $rl
161       30: loadi $r2 1
162       31: add $r3 $r1 $r2
163       32: loadi $r1 13
164       33: store $r3 $r1
165  L1.
166       34: loadi $rl 13
167       35: load $r1 $rl
168       36: loadi $rl 12
169       37: load $r2 $rl
170       38: lt $r3 $r1 $r2
171       39: jei $r0 $r3 L2
172       40: loadi $rl 10
173       41: load $r1 $rl
174       42: loadi $rl 13
175       43: load $r2 $rl
176       44: add $r4 $r1 $r2
177       45: load $r3 $r4
178       46: loadi $rl 14
179       47: load $r1 $rl
180       48: lt $r2 $r3 $r1
181       49: jei $r0 $r2 L3
182       50: loadi $rl 10
183       51: load $r1 $rl
184       52: loadi $rl 13
185       53: load $r2 $rl
186       54: add $r4 $r1 $r2
187       55: load $r3 $r4
188       56: loadi $r1 14
189       57: store $r3 $r1
190       58: loadi $r1 15
191       59: loadi $rl 13
192       60: load $r2 $rl
193       61: store $r2 $r1
194  L3.
195       62: loadi $rl 13
196       63: load $r1 $rl
197       64: loadi $r2 1
198       65: add $r3 $r1 $r2
199       66: loadi $r1 13
200       67: store $r3 $r1
201       68: jmpi L1
202  L2.
```

```
203       69: loadi $r1 15
204       70: load $rt $r1
205       71: jmp $ra
206       72: loadi $rt 0
207       73: jmp $ra
208  sort:
209       74: subi $stp $stp 1
210       75: pop $r1 $stp
211       76: loadi $rl 16
212       77: store $r1 $rl
213       78: subi $stp $stp 1
214       79: pop $r1 $stp
215       80: loadi $rl 17
216       81: store $r1 $rl
217       82: subi $stp $stp 1
218       83: pop $r1 $stp
219       84: loadi $rl 18
220       85: store $r1 $rl
221       86: loadi $r1 19
222       87: loadi $rl 17
223       88: load $r2 $rl
224       89: store $r2 $r1
225  L5.
226       90: loadi $rl 18
227       91: load $r1 $rl
228       92: loadi $r2 1
229       93: sub $r3 $r1 $r2
230       94: loadi $rl 19
231       95: load $r1 $rl
232       96: lt $r2 $r1 $r3
233       97: jei $r0 $r2 L6
234       98: push $ra $stp
235       99: addi $stp $stp 1
236       100: loadi $rl 18
237       101: load $r1 $rl
238       102: push $r1 $stp
239       103: addi $stp $stp 1
240       104: loadi $rl 19
241       105: load $r1 $rl
242       106: push $r1 $stp
243       107: addi $stp $stp 1
244       108: loadi $rl 16
245       109: load $r1 $rl
246       110: push $r1 $stp
247       111: addi $stp $stp 1
248       112: jal minloc
249       113: subi $stp $stp 1
250       114: pop $ra $stp
251       115: move $r1 $rt
252       116: loadi $r2 20
253       117: store $r1 $r2
254       118: loadi $rl 16
255       119: load $r1 $rl
256       120: loadi $rl 20
257       121: load $r2 $rl
258       122: add $r4 $r1 $r2
259       123: load $r3 $r4
260       124: loadi $r1 21
261       125: store $r3 $r1
262       126: loadi $rl 16
```

```
263       127: load $r1 $rl
264       128: loadi $rl 19
265       129: load $r2 $rl
266       130: add $r4 $r1 $r2
267       131: load $r3 $r4
268       132: loadi $rl 16
269       133: load $r1 $rl
270       134: loadi $rl 20
271       135: load $r2 $rl
272       136: add $r4 $r1 $r2
273       137: store $r3 $r4
274       138: loadi $rl 16
275       139: load $r1 $rl
276       140: loadi $rl 19
277       141: load $r2 $rl
278       142: add $r3 $r1 $r2
279       143: loadi $rl 21
280       144: load $r1 $rl
281       145: store $r1 $r3
282       146: loadi $rl 19
283       147: load $r1 $rl
284       148: loadi $r2 1
285       149: add $r3 $r1 $r2
286       150: loadi $r1 19
287       151: store $r3 $r1
288       152: jmpi L5
289  L6.
290       153: loadi $rt 0
291       154: jmp $ra
292  main:
293       155: loadi $r1 22
294       156: loadi $r2 0
295       157: store $r2 $r1
296  L7.
297       158: loadi $rl 22
298       159: load $r1 $rl
299       160: loadi $r2 4
300       161: lt $r3 $r1 $r2
301       162: jei $r0 $r3 L8
302       163: in $rt
303       164: move $r1 $rt
304       165: loadi $r2 0
305       166: loadi $rl 22
306       167: load $r3 $rl
307       168: add $r4 $r2 $r3
308       169: store $r1 $r4
309       170: loadi $rl 22
310       171: load $r1 $rl
311       172: loadi $r2 1
312       173: add $r3 $r1 $r2
313       174: loadi $r1 22
314       175: store $r3 $r1
315       176: jmpi L7
316  L8.
317       177: push $ra $stp
318       178: addi $stp $stp 1
319       179: loadi $r1 4
320       180: push $r1 $stp
321       181: addi $stp $stp 1
322       182: loadi $r1 0
```

```
323       183: push $r1 $stp
324       184: addi $stp $stp 1
325       185: loadi $r1 0
326       186: push $r1 $stp
327       187: addi $stp $stp 1
328       188: jal sort
329       189: subi $stp $stp 1
330       190: pop $ra $stp
331       191: move $r1 $rt
332       192: loadi $r2 22
333       193: loadi $r3 0
334       194: store $r3 $r2
335  L9.
336       195: loadi $rl 22
337       196: load $r2 $rl
338       197: loadi $r3 4
339       198: lt $r4 $r2 $r3
340       199: jei $r0 $r4 L10
341       200: loadi $r2 0
342       201: loadi $rl 22
343       202: load $r3 $rl
344       203: add $r5 $r2 $r3
345       204: load $r4 $r5
346       205: out $r4
347       206: loadi $rl 22
348       207: load $r2 $rl
349       208: loadi $r3 1
350       209: add $r4 $r2 $r3
351       210: loadi $r2 22
352       211: store $r4 $r2
353       212: jmpi L9
354  L10.
355
356  /* conjunto de instru  es em bin rio:*/
357
358  mem_ram[0] = 32'b001101_00000_00000000000000000000; //['loadi', '$r0', '0']
359  mem_ram[1] = 32'b001101_11101_00000000000000000000; //['loadi', '$r29', '0']
360  mem_ram[2] = 32'b001101_11111_00000000000000000000; //['loadi', '$r31', '0']
361  mem_ram[3] = 32'b010000_00000_00000000000010011011; //['jmpi', 'main']
362  mem_ram[4] = 32'b000100_11101_11101_00000000000000001; //['subi', '$r29', '$r29', '1']
363  mem_ram[5] = 32'b100100_00001_11101_00000000000000000; //['pop', '$r1', '$r29']
364  mem_ram[6] = 32'b001101_11110_00000000000000001010; //['loadi', '$r30', 10]
365  mem_ram[7] = 32'b001110_00001_11110_00000000000000000; //['store', '$r1', '$r30']
366  mem_ram[8] = 32'b000100_11101_11101_00000000000000001; //['subi', '$r29', '$r29', '1']
367  mem_ram[9] = 32'b100100_00001_11101_00000000000000000; //['pop', '$r1', '$r29']
368  mem_ram[10] = 32'b001101_11110_00000000000000001011; //['loadi', '$r30', 11]
369  mem_ram[11] = 32'b001110_00001_11110_00000000000000000; //['store', '$r1', '$r30']
370  mem_ram[12] = 32'b000100_11101_11101_00000000000000001; //['subi', '$r29', '$r29', '1']
371  mem_ram[13] = 32'b100100_00001_11101_00000000000000000; //['pop', '$r1', '$r29']
372  mem_ram[14] = 32'b001101_11110_00000000000000001100; //['loadi', '$r30', 12]
373  mem_ram[15] = 32'b001110_00001_11110_00000000000000000; //['store', '$r1', '$r30']
374  mem_ram[16] = 32'b001101_00001_00000000000000001111; //['loadi', '$r1', 15]
375  mem_ram[17] = 32'b001101_11110_00000000000000001011; //['loadi', '$r30', 11]
376  mem_ram[18] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
377  mem_ram[19] = 32'b001110_00010_00001_00000000000000000; //['store', '$r2', '$r1']
378  mem_ram[20] = 32'b001101_11110_00000000000000001010; //['loadi', '$r30', 10]
379  mem_ram[21] = 32'b001100_00001_11110_00000000000000000; //['load', '$r1', '$r30']
380  mem_ram[22] = 32'b001101_11110_00000000000000001011; //['loadi', '$r30', 11]
381  mem_ram[23] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
382  mem_ram[24] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
```

```
383 mem_ram[25] = 32'b001100_00011_00100_0000000000000000; //['load', '$r3', '$r4']
384 mem_ram[26] = 32'b001101_00001_00000000000000000001110; //['loadi', '$r1', 14]
385 mem_ram[27] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
386 mem_ram[28] = 32'b001101_11110_00000000000000000001011; //['loadi', '$r30', 11]
387 mem_ram[29] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
388 mem_ram[30] = 32'b001101_00010_00000000000000000000001; //['loadi', '$r2', '1']
389 mem_ram[31] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
390 mem_ram[32] = 32'b001101_00001_00000000000000000001101; //['loadi', '$r1', 13]
391 mem_ram[33] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
392 mem_ram[34] = 32'b001101_11110_00000000000000000001101; //['loadi', '$r30', 13]
393 mem_ram[35] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
394 mem_ram[36] = 32'b001101_11110_00000000000000000001100; //['loadi', '$r30', 12]
395 mem_ram[37] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
396 mem_ram[38] = 32'b011111_00011_00001_00010_00000000000; //['lt', '$r3', '$r1', '$r2']
397 mem_ram[39] = 32'b100010_00000_00011_0000000001000101; //['jei', '$r0', '$r3', 'L2']
398 mem_ram[40] = 32'b001101_11110_00000000000000000001010; //['loadi', '$r30', 10]
399 mem_ram[41] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
400 mem_ram[42] = 32'b001101_11110_00000000000000000001101; //['loadi', '$r30', 13]
401 mem_ram[43] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
402 mem_ram[44] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
403 mem_ram[45] = 32'b001100_00011_00100_0000000000000000; //['load', '$r3', '$r4']
404 mem_ram[46] = 32'b001101_11110_00000000000000000001110; //['loadi', '$r30', 14]
405 mem_ram[47] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
406 mem_ram[48] = 32'b011111_00010_00011_00001_00000000000; //['lt', '$r2', '$r3', '$r1']
407 mem_ram[49] = 32'b100010_00000_00010_0000000000111110; //['jei', '$r0', '$r2', 'L3']
408 mem_ram[50] = 32'b001101_11110_00000000000000000001010; //['loadi', '$r30', 10]
409 mem_ram[51] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
410 mem_ram[52] = 32'b001101_11110_00000000000000000001101; //['loadi', '$r30', 13]
411 mem_ram[53] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
412 mem_ram[54] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
413 mem_ram[55] = 32'b001100_00011_00100_0000000000000000; //['load', '$r3', '$r4']
414 mem_ram[56] = 32'b001101_00001_00000000000000000001110; //['loadi', '$r1', 14]
415 mem_ram[57] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
416 mem_ram[58] = 32'b001101_00001_00000000000000000001111; //['loadi', '$r1', 15]
417 mem_ram[59] = 32'b001101_11110_00000000000000000001101; //['loadi', '$r30', 13]
418 mem_ram[60] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
419 mem_ram[61] = 32'b001110_00010_00001_0000000000000000; //['store', '$r2', '$r1']
420 mem_ram[62] = 32'b001101_11110_00000000000000000001101; //['loadi', '$r30', 13]
421 mem_ram[63] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
422 mem_ram[64] = 32'b001101_00010_00000000000000000000001; //['loadi', '$r2', '1']
423 mem_ram[65] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
424 mem_ram[66] = 32'b001101_00001_00000000000000000001101; //['loadi', '$r1', 13]
425 mem_ram[67] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
426 mem_ram[68] = 32'b010000_00000_00000000000000000100010; //['jmpi', 'L1']
427 mem_ram[69] = 32'b001101_00001_00000000000000000001111; //['loadi', '$r1', 15]
428 mem_ram[70] = 32'b001100_11100_00001_0000000000000000; //['load', '$r28', '$r1']
429 mem_ram[71] = 32'b010001_11111_00000000000000000000000; //['jmp', '$r31']
430 mem_ram[72] = 32'b001101_11100_00000000000000000000000; //['loadi', '$r28', '0']
431 mem_ram[73] = 32'b010001_11111_00000000000000000000000; //['jmp', '$r31']
432 mem_ram[74] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
433 mem_ram[75] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
434 mem_ram[76] = 32'b001101_11110_00000000000000000010000; //['loadi', '$r30', 16]
435 mem_ram[77] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
436 mem_ram[78] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
437 mem_ram[79] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
438 mem_ram[80] = 32'b001101_11110_00000000000000000010001; //['loadi', '$r30', 17]
439 mem_ram[81] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
440 mem_ram[82] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
441 mem_ram[83] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
442 mem_ram[84] = 32'b001101_11110_00000000000000000010010; //['loadi', '$r30', 18]
```

```
443 mem_ram[85] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
444 mem_ram[86] = 32'b001101_00001_00000000000000010011; //['loadi', '$r1', 19]
445 mem_ram[87] = 32'b001101_11110_00000000000000010001; //['loadi', '$r30', 17]
446 mem_ram[88] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
447 mem_ram[89] = 32'b001110_00010_00001_0000000000000000; //['store', '$r2', '$r1']
448 mem_ram[90] = 32'b001101_11110_00000000000000010010; //['loadi', '$r30', 18]
449 mem_ram[91] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
450 mem_ram[92] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
451 mem_ram[93] = 32'b000011_00011_00001_00010_00000000000; //['sub', '$r3', '$r1', '$r2']
452 mem_ram[94] = 32'b001101_11110_00000000000000010011; //['loadi', '$r30', 19]
453 mem_ram[95] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
454 mem_ram[96] = 32'b011111_00010_00001_00011_00000000000; //['lt', '$r2', '$r1', '$r3']
455 mem_ram[97] = 32'b100010_00000_00010_0000000010011001; //['jei', '$r0', '$r2', 'L6']
456 mem_ram[98] = 32'b100011_11111_11101_0000000000000000; //['push', '$r31', '$r29']
457 mem_ram[99] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
458 mem_ram[100] = 32'b001101_11110_00000000000000010010; //['loadi', '$r30', 18]
459 mem_ram[101] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
460 mem_ram[102] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
461 mem_ram[103] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
462 mem_ram[104] = 32'b001101_11110_00000000000000010011; //['loadi', '$r30', 19]
463 mem_ram[105] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
464 mem_ram[106] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
465 mem_ram[107] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
466 mem_ram[108] = 32'b001101_11110_00000000000000010000; //['loadi', '$r30', 16]
467 mem_ram[109] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
468 mem_ram[110] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
469 mem_ram[111] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
470 mem_ram[112] = 32'b100001_00000000000000000000000100; //['jal', 'minloc']
471 mem_ram[113] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
472 mem_ram[114] = 32'b100100_11111_11101_0000000000000000; //['pop', '$r31', '$r29']
473 mem_ram[115] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
474 mem_ram[116] = 32'b001101_00010_00000000000000010100; //['loadi', '$r2', 20]
475 mem_ram[117] = 32'b001110_00001_00010_0000000000000000; //['store', '$r1', '$r2']
476 mem_ram[118] = 32'b001101_11110_00000000000000010000; //['loadi', '$r30', 16]
477 mem_ram[119] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
478 mem_ram[120] = 32'b001101_11110_00000000000000010100; //['loadi', '$r30', 20]
479 mem_ram[121] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
480 mem_ram[122] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
481 mem_ram[123] = 32'b001100_00011_00100_0000000000000000; //['load', '$r3', '$r4']
482 mem_ram[124] = 32'b001101_00001_00000000000000010101; //['loadi', '$r1', 21]
483 mem_ram[125] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
484 mem_ram[126] = 32'b001101_11110_00000000000000010000; //['loadi', '$r30', 16]
485 mem_ram[127] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
486 mem_ram[128] = 32'b001101_11110_00000000000000010011; //['loadi', '$r30', 19]
487 mem_ram[129] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
488 mem_ram[130] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
489 mem_ram[131] = 32'b001100_00011_00100_0000000000000000; //['load', '$r3', '$r4']
490 mem_ram[132] = 32'b001101_11110_00000000000000010000; //['loadi', '$r30', 16]
491 mem_ram[133] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
492 mem_ram[134] = 32'b001101_11110_00000000000000010100; //['loadi', '$r30', 20]
493 mem_ram[135] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
494 mem_ram[136] = 32'b000001_00100_00001_00010_00000000000; //['add', '$r4', '$r1', '$r2']
495 mem_ram[137] = 32'b001110_00011_00100_0000000000000000; //['store', '$r3', '$r4']
496 mem_ram[138] = 32'b001101_11110_00000000000000010000; //['loadi', '$r30', 16]
497 mem_ram[139] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
498 mem_ram[140] = 32'b001101_11110_00000000000000010011; //['loadi', '$r30', 19]
499 mem_ram[141] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
500 mem_ram[142] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
501 mem_ram[143] = 32'b001101_11110_00000000000000010101; //['loadi', '$r30', 21]
502 mem_ram[144] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
```

```
503 mem_ram[145] = 32'b001110_00001_00011_0000000000000000; //['store', '$r1', '$r3']
504 mem_ram[146] = 32'b001101_11110_00000000000000010011; //['loadi', '$r30', 19]
505 mem_ram[147] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
506 mem_ram[148] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
507 mem_ram[149] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
508 mem_ram[150] = 32'b001101_00001_00000000000000010011; //['loadi', '$r1', 19]
509 mem_ram[151] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
510 mem_ram[152] = 32'b010000_00000_00000000000001011010; //['jmpi', 'L5']
511 mem_ram[153] = 32'b001101_11100_00000000000000000000; //['loadi', '$r28', '0']
512 mem_ram[154] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
513 mem_ram[155] = 32'b001101_00001_00000000000000010110; //['loadi', '$r1', 22]
514 mem_ram[156] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', '0']
515 mem_ram[157] = 32'b001110_00010_00001_0000000000000000; //['store', '$r2', '$r1']
516 mem_ram[158] = 32'b001101_11110_00000000000000010110; //['loadi', '$r30', 22]
517 mem_ram[159] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
518 mem_ram[160] = 32'b001101_00010_00000000000000000100; //['loadi', '$r2', '4']
519 mem_ram[161] = 32'b011111_00011_00001_00010_00000000000; //['lt', '$r3', '$r1', '$r2']
520 mem_ram[162] = 32'b100010_00000_00011_0000000010110001; //['jei', '$r0', '$r3', 'L8']
521 mem_ram[163] = 32'b010110_11100_00000000000000000000; //['in', '$r28']
522 mem_ram[164] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
523 mem_ram[165] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', 0]
524 mem_ram[166] = 32'b001101_11110_00000000000000010110; //['loadi', '$r30', 22]
525 mem_ram[167] = 32'b001100_00011_11110_0000000000000000; //['load', '$r3', '$r30']
526 mem_ram[168] = 32'b000001_00100_00010_00011_00000000000; //['add', '$r4', '$r2', '$r3']
527 mem_ram[169] = 32'b001110_00001_00100_0000000000000000; //['store', '$r1', '$r4']
528 mem_ram[170] = 32'b001101_11110_00000000000000010110; //['loadi', '$r30', 22]
529 mem_ram[171] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
530 mem_ram[172] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
531 mem_ram[173] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
532 mem_ram[174] = 32'b001101_00001_00000000000000010110; //['loadi', '$r1', 22]
533 mem_ram[175] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
534 mem_ram[176] = 32'b010000_00000_00000000000010011110; //['jmpi', 'L7']
535 mem_ram[177] = 32'b100011_11111_11101_0000000000000000; //['push', '$r31', '$r29']
536 mem_ram[178] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
537 mem_ram[179] = 32'b001101_00001_00000000000000000100; //['loadi', '$r1', '4']
538 mem_ram[180] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
539 mem_ram[181] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
540 mem_ram[182] = 32'b001101_00001_00000000000000000000; //['loadi', '$r1', '0']
541 mem_ram[183] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
542 mem_ram[184] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
543 mem_ram[185] = 32'b001101_00001_00000000000000000000; //['loadi', '$r1', 0]
544 mem_ram[186] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
545 mem_ram[187] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
546 mem_ram[188] = 32'b100001_00000000000000000001001010; //['jal', 'sort']
547 mem_ram[189] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
548 mem_ram[190] = 32'b100100_11111_11101_0000000000000000; //['pop', '$r31', '$r29']
549 mem_ram[191] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
550 mem_ram[192] = 32'b001101_00010_00000000000000010110; //['loadi', '$r2', 22]
551 mem_ram[193] = 32'b001101_00011_00000000000000000000; //['loadi', '$r3', '0']
552 mem_ram[194] = 32'b001110_00011_00010_0000000000000000; //['store', '$r3', '$r2']
553 mem_ram[195] = 32'b001101_11110_00000000000000010110; //['loadi', '$r30', 22]
554 mem_ram[196] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
555 mem_ram[197] = 32'b001101_00011_00000000000000000100; //['loadi', '$r3', '4']
556 mem_ram[198] = 32'b011111_00100_00010_00011_00000000000; //['lt', '$r4', '$r2', '$r3']
557 mem_ram[199] = 32'b100010_00000_00100_0000000011010101; //['jei', '$r0', '$r4', 'L10']
558 mem_ram[200] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', 0]
559 mem_ram[201] = 32'b001101_11110_00000000000000010110; //['loadi', '$r30', 22]
560 mem_ram[202] = 32'b001100_00011_11110_0000000000000000; //['load', '$r3', '$r30']
561 mem_ram[203] = 32'b000001_00101_00010_00011_00000000000; //['add', '$r5', '$r2', '$r3']
562 mem_ram[204] = 32'b001100_00100_00101_0000000000000000; //['load', '$r4', '$r5']
```

```
563 mem_ram[205] = 32'b010111_00100_00000000000000000000; //['out', '$r4']
564 mem_ram[206] = 32'b001101_11110_00000000000000000110; //['loadi', '$r30', 22]
565 mem_ram[207] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
566 mem_ram[208] = 32'b001101_00011_00000000000000000001; //['loadi', '$r3', '1']
567 mem_ram[209] = 32'b000001_00100_00010_00011_00000000000; //['add', '$r4', '$r2', '$r3']
568 mem_ram[210] = 32'b001101_00010_00000000000000000110; //['loadi', '$r2', 22]
569 mem_ram[211] = 32'b001110_00100_00010_00000000000000000; //['store', '$r4', '$r2']
570 mem_ram[212] = 32'b010000_00000_00000000000011000011; //['jmpi', 'L9']
571 mem_ram[213] = 32'b011001_00000000000000000000000000; // halt
```

Saídas geradas pelo código do mdc:

```
1  /* Um programa para calcular o mdc
2     segundo o algoritmo de Euclides. */
3
4  int gcd (int u, int v)
5  {
6          if (v == 0){ return u; }
7          else{ return gcd(v,u-u/v*v);}
8
9          /* u-u/v*v == u mod v */
10 }
11
12 void main(void)
13 {        int x; int y;
14         x = input(); y = input();
15         output(gcd(x,y));
16 }
17
18 /*C digo intermedi rio:*/
19
20 0  : (function, gcd, , )
21 1  : (equal_to, v, 0, t1)
22 2  : (jump_if_false, t1, L1, )
23 3  : (return, u, , )
24 4  : (go_to, L2, , )
25 5  : (label, L1, , )
26 6  : (arg, v, , )
27 7  : (division, u, v, t2)
28 8  : (multiplication, t2, v, t3)
29 9  : (subtraction, u, t3, t4)
30 10  : (arg, t4, , )
31 11  : (function_call, gcd, 2, )
32 12  : (assign_ret, t5, RT, )
33 13  : (return, t5, , )
34 14  : (label, L2, , )
35 15  : (return, 0, , )
36 16  : (function, main, , )
37 17  : (sys_call, input, , )
38 18  : (assign_ret, t6, RT, )
39 19  : (assign, x, t6, )
40 20  : (sys_call, input, , )
41 21  : (assign_ret, t7, RT, )
42 22  : (assign, y, t7, )
43 23  : (arg, x, , )
44 24  : (arg, y, , )
45 25  : (function_call, gcd, 2, )
46 26  : (assign_ret, t8, RT, )
47 27  : (arg, t8, , )
48 28  : (sys_call, output, 1, )
```

```
49
50  /* Assembly: */
51
52        0: loadi $r0 0
53        1: loadi $stp 0
54        2: loadi $ra 0
55        3: jmpi main
56  gcd:
57        4: subi $stp $stp 1
58        5: pop $r1 $stp
59        6: loadi $rl 0
60        7: store $r1 $rl
61        8: subi $stp $stp 1
62        9: pop $r1 $stp
63        10: loadi $rl 1
64        11: store $r1 $rl
65        12: loadi $rl 1
66        13: load $r1 $rl
67        14: loadi $r2 0
68        15: eq $r3 $r1 $r2
69        16: jei $r0 $r3 L1
70        17: loadi $r1 0
71        18: load $rt $r1
72        19: jmp $ra
73        20: jmpi L2
74  L1.
75        21: loadi $rl 0
76        22: load $r1 $rl
77        23: loadi $rl 1
78        24: load $r2 $rl
79        25: div $r3 $r1 $r2
80        26: loadi $rl 1
81        27: load $r1 $rl
82        28: mult $r2 $r3 $r1
83        29: loadi $rl 0
84        30: load $r1 $rl
85        31: sub $r3 $r1 $r2
86        32: push $ra $stp
87        33: addi $stp $stp 1
88        34: loadi $rl 0
89        35: load $r1 $rl
90        36: push $r1 $stp
91        37: addi $stp $stp 1
92        38: loadi $rl 1
93        39: load $r1 $rl
94        40: push $r1 $stp
95        41: addi $stp $stp 1
96        42: push $r3 $stp
97        43: addi $stp $stp 1
98        44: loadi $rl 1
99        45: load $r1 $rl
100       46: push $r1 $stp
101       47: addi $stp $stp 1
102       48: jal gcd
103       49: subi $stp $stp 1
104       50: pop $r1 $stp
105       51: loadi $rl 1
106       52: store $r1 $rl
107       53: subi $stp $stp 1
108       54: pop $r1 $stp
```

```
109       55: loadi $rl 0
110       56: store $r1 $rl
111       57: subi $stp $stp 1
112       58: pop $ra $stp
113       59: move $r1 $rt
114       60: move $rt $r1
115       61: jmp $ra
116 L2.
117       62: loadi $rt 0
118       63: jmp $ra
119 main:
120       64: in $rt
121       65: move $r1 $rt
122       66: loadi $r2 2
123       67: store $r1 $r2
124       68: in $rt
125       69: move $r1 $rt
126       70: loadi $r2 3
127       71: store $r1 $r2
128       72: push $ra $stp
129       73: addi $stp $stp 1
130       74: loadi $rl 3
131       75: load $r1 $rl
132       76: push $r1 $stp
133       77: addi $stp $stp 1
134       78: loadi $rl 2
135       79: load $r1 $rl
136       80: push $r1 $stp
137       81: addi $stp $stp 1
138       82: jal gcd
139       83: subi $stp $stp 1
140       84: pop $ra $stp
141       85: move $r1 $rt
142       86: out $r1
143
144 /* conjunto de instru  es em bin rio:*/
145
146 mem_ram[0] = 32'b001101_00000_00000000000000000000; //['loadi', '$r0', '0']
147 mem_ram[1] = 32'b001101_11101_00000000000000000000; //['loadi', '$r29', '0']
148 mem_ram[2] = 32'b001101_11111_00000000000000000000; //['loadi', '$r31', '0']
149 mem_ram[3] = 32'b010000_00000_00000000000001000000; //['jmpi', 'main']
150 mem_ram[4] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
151 mem_ram[5] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
152 mem_ram[6] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
153 mem_ram[7] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
154 mem_ram[8] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
155 mem_ram[9] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
156 mem_ram[10] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
157 mem_ram[11] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
158 mem_ram[12] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
159 mem_ram[13] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
160 mem_ram[14] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', '0']
161 mem_ram[15] = 32'b011011_00011_00001_00010_00000000000; //['eq', '$r3', '$r1', '$r2']
162 mem_ram[16] = 32'b100010_00000_00011_0000000000010101; //['jei', '$r0', '$r3', 'L1']
163 mem_ram[17] = 32'b001101_00001_00000000000000000000; //['loadi', '$r1', 0]
164 mem_ram[18] = 32'b001100_11100_00001_0000000000000000; //['load', '$r28', '$r1']
165 mem_ram[19] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
166 mem_ram[20] = 32'b010000_00000_00000000000000111110; //['jmpi', 'L2']
167 mem_ram[21] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
168 mem_ram[22] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
```

```
169 mem_ram[23] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
170 mem_ram[24] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
171 mem_ram[25] = 32'b011010_00011_00001_00010_00000000000; //['div', '$r3', '$r1', '$r2']
172 mem_ram[26] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
173 mem_ram[27] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
174 mem_ram[28] = 32'b000101_00010_00011_00001_00000000000; //['mult', '$r2', '$r3', '$r1']
175 mem_ram[29] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
176 mem_ram[30] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
177 mem_ram[31] = 32'b000011_00011_00001_00010_00000000000; //['sub', '$r3', '$r1', '$r2']
178 mem_ram[32] = 32'b100011_11111_11101_0000000000000000; //['push', '$r31', '$r29']
179 mem_ram[33] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
180 mem_ram[34] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
181 mem_ram[35] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
182 mem_ram[36] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
183 mem_ram[37] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
184 mem_ram[38] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
185 mem_ram[39] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
186 mem_ram[40] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
187 mem_ram[41] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
188 mem_ram[42] = 32'b100011_00011_11101_0000000000000000; //['push', '$r3', '$r29']
189 mem_ram[43] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
190 mem_ram[44] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
191 mem_ram[45] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
192 mem_ram[46] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
193 mem_ram[47] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
194 mem_ram[48] = 32'b100001_00000000000000000000000100; //['jal', 'gcd']
195 mem_ram[49] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
196 mem_ram[50] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
197 mem_ram[51] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
198 mem_ram[52] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
199 mem_ram[53] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
200 mem_ram[54] = 32'b100100_00001_11101_0000000000000000; //['pop', '$r1', '$r29']
201 mem_ram[55] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
202 mem_ram[56] = 32'b001110_00001_11110_0000000000000000; //['store', '$r1', '$r30']
203 mem_ram[57] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
204 mem_ram[58] = 32'b100100_11111_11101_0000000000000000; //['pop', '$r31', '$r29']
205 mem_ram[59] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
206 mem_ram[60] = 32'b001111_11100_00001_0000000000000000; //['move', '$r28', '$r1']
207 mem_ram[61] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
208 mem_ram[62] = 32'b001101_11100_00000000000000000000; //['loadi', '$r28', '0']
209 mem_ram[63] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
210 mem_ram[64] = 32'b010110_11100_00000000000000000000; //['in', '$r28']
211 mem_ram[65] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
212 mem_ram[66] = 32'b001101_00010_00000000000000000010; //['loadi', '$r2', 2]
213 mem_ram[67] = 32'b001110_00001_00010_0000000000000000; //['store', '$r1', '$r2']
214 mem_ram[68] = 32'b010110_11100_00000000000000000000; //['in', '$r28']
215 mem_ram[69] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
216 mem_ram[70] = 32'b001101_00010_00000000000000000011; //['loadi', '$r2', 3]
217 mem_ram[71] = 32'b001110_00001_00010_0000000000000000; //['store', '$r1', '$r2']
218 mem_ram[72] = 32'b100011_11111_11101_0000000000000000; //['push', '$r31', '$r29']
219 mem_ram[73] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
220 mem_ram[74] = 32'b001101_11110_00000000000000000011; //['loadi', '$r30', 3]
221 mem_ram[75] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
222 mem_ram[76] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
223 mem_ram[77] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
224 mem_ram[78] = 32'b001101_11110_00000000000000000010; //['loadi', '$r30', 2]
225 mem_ram[79] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
226 mem_ram[80] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
227 mem_ram[81] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
228 mem_ram[82] = 32'b100001_00000000000000000000000100; //['jal', 'gcd']
```

```
mem_ram[83] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
mem_ram[84] = 32'b100100_11111_11101_0000000000000000; //['pop', '$r31', '$r29']
mem_ram[85] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
mem_ram[86] = 32'b010111_00001_00000000000000000000; //['out', '$r1']
```

Saídas geradas pelo código fibonacci:

Figura 5 – Tabela de símbolos código fibonacci

Key	Name	Scope	Lines	Id Type	Data Type	Pos Mem	Qtd Args	Args
fibonacci	fibonacci	global	1	funct	int	-1	1	['n']
fibonacci.n	n	fibonacci	1, 3, 4, 9	var	int	0		
fibonacci.a	a	fibonacci	2, 6, 11, 12	var	int	1		
fibonacci.b	b	fibonacci	2, 7, 10, 11, 15	var	int	2		
fibonacci.c	c	fibonacci	2, 10, 12	var	int	3		
fibonacci.i	i	fibonacci	2, 8, 9, 13	var	int	4		
main	main	global	21	funct	void	-1	0	[]
main.x	x	main	22, 23	var	int	5		
input	input	main	23	sys_call	int	-1		
output	output	main	24	sys_call	int	-1		

Fonte: O autor

```
int fibonacci(int n){
    int a; int b; int c; int i;
    if (n <= 0){ return 0;}
    else{ if (n == 1){ return 1;}
    else {
        a = 0;
        b = 1;
        i = 1;
        while (i < n){
            c = b;
            b = b + a;
            a = c;
            i = i + 1;
        }
        return b;
     }
    }
}


void main(void)
{       int x;
        x = input();
        output(fibonacci(x));
}

/*C digo intermedi rio:*/

0  : (function, fibonacci, , )
1  : (less_than_equal_to, n, 0, t1)
2  : (jump_if_false, t1, L1, )
3  : (return, 0, , )
```

```
4  : (go_to, L2, , )
5  : (label, L1, , )
6  : (equal_to, n, 1, t2)
7  : (jump_if_false, t2, L3, )
8  : (return, 1, , )
9  : (go_to, L4, , )
10  : (label, L3, , )
11  : (assign, a, 0, )
12  : (assign, b, 1, )
13  : (assign, i, 1, )
14  : (label, L5, , )
15  : (less_than, i, n, t3)
16  : (jump_if_false, t3, L6, )
17  : (assign, c, b, )
18  : (addition, b, a, t4)
19  : (assign, b, t4, )
20  : (assign, a, c, )
21  : (addition, i, 1, t5)
22  : (assign, i, t5, )
23  : (go_to, L5, , )
24  : (label, L6, , )
25  : (return, b, , )
26  : (label, L4, , )
27  : (label, L2, , )
28  : (return, 0, , )
29  : (function, main, , )
30  : (sys_call, input, , )
31  : (assign_ret, t6, RT, )
32  : (assign, x, t6, )
33  : (arg, x, , )
34  : (function_call, fibonacci, 1, )
35  : (assign_ret, t7, RT, )
36  : (arg, t7, , )
37  : (sys_call, output, 1, )

/* Assembly: */

     0: loadi $r0 0
     1: loadi $stp 0
     2: loadi $ra 0
     3: jmpi main
fibonacci:
     4: subi $stp $stp 1
     5: pop $r1 $stp
     6: loadi $rl 0
     7: store $r1 $rl
     8: loadi $rl 0
     9: load $r1 $rl
     10: loadi $r2 0
     11: nab $r3 $r1 $r2
     12: jei $r0 $r3 L1
     13: loadi $rt 0
     14: jmp $ra
     15: jmpi L2
L1.
     16: loadi $rl 0
     17: load $r1 $rl
     18: loadi $r2 1
     19: eq $r3 $r1 $r2
     20: jei $r0 $r3 L3
```

```
93        21: loadi $rt 1
94        22: jmp $ra
95        23: jmpi L4
96  L3.
97        24: loadi $r1 1
98        25: loadi $r2 0
99        26: store $r2 $r1
100       27: loadi $r1 2
101       28: loadi $r2 1
102       29: store $r2 $r1
103       30: loadi $r1 4
104       31: loadi $r2 1
105       32: store $r2 $r1
106 L5.
107       33: loadi $rl 4
108       34: load $r1 $rl
109       35: loadi $rl 0
110       36: load $r2 $rl
111       37: lt $r3 $r1 $r2
112       38: jei $r0 $r3 L6
113       39: loadi $r1 3
114       40: loadi $rl 2
115       41: load $r2 $rl
116       42: store $r2 $r1
117       43: loadi $rl 2
118       44: load $r1 $rl
119       45: loadi $rl 1
120       46: load $r2 $rl
121       47: add $r3 $r1 $r2
122       48: loadi $r1 2
123       49: store $r3 $r1
124       50: loadi $r1 1
125       51: loadi $rl 3
126       52: load $r2 $rl
127       53: store $r2 $r1
128       54: loadi $rl 4
129       55: load $r1 $rl
130       56: loadi $r2 1
131       57: add $r3 $r1 $r2
132       58: loadi $r1 4
133       59: store $r3 $r1
134       60: jmpi L5
135 L6.
136       61: loadi $r1 2
137       62: load $rt $r1
138       63: jmp $ra
139 L4.
140 L2.
141       64: loadi $rt 0
142       65: jmp $ra
143 main:
144       66: in $rt
145       67: move $r1 $rt
146       68: loadi $r2 5
147       69: store $r1 $r2
148       70: push $ra $stp
149       71: addi $stp $stp 1
150       72: loadi $rl 5
151       73: load $r1 $rl
152       74: push $r1 $stp
```

```
153      75: addi $stp $stp 1
154      76: jal fibonacci
155      77: subi $stp $stp 1
156      78: pop $ra $stp
157      79: move $r1 $rt
158      80: out $r1
159
160 /* conjunto de instru  es em bin rio:*/
161
162 mem_ram[0] = 32'b001101_00000_00000000000000000000; //['loadi', '$r0', '0']
163 mem_ram[1] = 32'b001101_11101_00000000000000000000; //['loadi', '$r29', '0']
164 mem_ram[2] = 32'b001101_11111_00000000000000000000; //['loadi', '$r31', '0']
165 mem_ram[3] = 32'b010000_00000_00000000000001000010; //['jmpi', 'main']
166 mem_ram[4] = 32'b000100_11101_11101_00000000000000001; //['subi', '$r29', '$r29', '1']
167 mem_ram[5] = 32'b100100_00001_11101_00000000000000000; //['pop', '$r1', '$r29']
168 mem_ram[6] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
169 mem_ram[7] = 32'b001110_00001_11110_00000000000000000; //['store', '$r1', '$r30']
170 mem_ram[8] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
171 mem_ram[9] = 32'b001100_00001_11110_00000000000000000; //['load', '$r1', '$r30']
172 mem_ram[10] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', '0']
173 mem_ram[11] = 32'b011110_00011_00001_00010_00000000000; //['nab', '$r3', '$r1', '$r2']
174 mem_ram[12] = 32'b100010_00000_00011_00000000000010000; //['jei', '$r0', '$r3', 'L1']
175 mem_ram[13] = 32'b001101_11100_00000000000000000000; //['loadi', '$r28', '0']
176 mem_ram[14] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
177 mem_ram[15] = 32'b010000_00000_00000000000001000000; //['jmpi', 'L2']
178 mem_ram[16] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
179 mem_ram[17] = 32'b001100_00001_11110_00000000000000000; //['load', '$r1', '$r30']
180 mem_ram[18] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
181 mem_ram[19] = 32'b011011_00011_00001_00010_00000000000; //['eq', '$r3', '$r1', '$r2']
182 mem_ram[20] = 32'b100010_00000_00011_00000000000011000; //['jei', '$r0', '$r3', 'L3']
183 mem_ram[21] = 32'b001101_11100_00000000000000000001; //['loadi', '$r28', '1']
184 mem_ram[22] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
185 mem_ram[23] = 32'b010000_00000_00000000000001000000; //['jmpi', 'L4']
186 mem_ram[24] = 32'b001101_00001_00000000000000000001; //['loadi', '$r1', 1]
187 mem_ram[25] = 32'b001101_00010_00000000000000000000; //['loadi', '$r2', '0']
188 mem_ram[26] = 32'b001110_00010_00001_00000000000000000; //['store', '$r2', '$r1']
189 mem_ram[27] = 32'b001101_00001_00000000000000000010; //['loadi', '$r1', 2]
190 mem_ram[28] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
191 mem_ram[29] = 32'b001110_00010_00001_00000000000000000; //['store', '$r2', '$r1']
192 mem_ram[30] = 32'b001101_00001_00000000000000000100; //['loadi', '$r1', 4]
193 mem_ram[31] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
194 mem_ram[32] = 32'b001110_00010_00001_00000000000000000; //['store', '$r2', '$r1']
195 mem_ram[33] = 32'b001101_11110_00000000000000000100; //['loadi', '$r30', 4]
196 mem_ram[34] = 32'b001100_00001_11110_00000000000000000; //['load', '$r1', '$r30']
197 mem_ram[35] = 32'b001101_11110_00000000000000000000; //['loadi', '$r30', 0]
198 mem_ram[36] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
199 mem_ram[37] = 32'b011111_00011_00001_00010_00000000000; //['lt', '$r3', '$r1', '$r2']
200 mem_ram[38] = 32'b100010_00000_00011_00000000000111101; //['jei', '$r0', '$r3', 'L6']
201 mem_ram[39] = 32'b001101_00001_00000000000000000011; //['loadi', '$r1', 3]
202 mem_ram[40] = 32'b001101_11110_00000000000000000010; //['loadi', '$r30', 2]
203 mem_ram[41] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
204 mem_ram[42] = 32'b001110_00010_00001_00000000000000000; //['store', '$r2', '$r1']
205 mem_ram[43] = 32'b001101_11110_00000000000000000010; //['loadi', '$r30', 2]
206 mem_ram[44] = 32'b001100_00001_11110_00000000000000000; //['load', '$r1', '$r30']
207 mem_ram[45] = 32'b001101_11110_00000000000000000001; //['loadi', '$r30', 1]
208 mem_ram[46] = 32'b001100_00010_11110_00000000000000000; //['load', '$r2', '$r30']
209 mem_ram[47] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
210 mem_ram[48] = 32'b001101_00001_00000000000000000010; //['loadi', '$r1', 2]
211 mem_ram[49] = 32'b001110_00011_00001_00000000000000000; //['store', '$r3', '$r1']
212 mem_ram[50] = 32'b001101_00001_00000000000000000001; //['loadi', '$r1', 1]
```

```
213 mem_ram[51] = 32'b001101_11110_00000000000000000011; //['loadi', '$r30', 3]
214 mem_ram[52] = 32'b001100_00010_11110_0000000000000000; //['load', '$r2', '$r30']
215 mem_ram[53] = 32'b001110_00010_00001_0000000000000000; //['store', '$r2', '$r1']
216 mem_ram[54] = 32'b001101_11110_00000000000000000100; //['loadi', '$r30', 4]
217 mem_ram[55] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
218 mem_ram[56] = 32'b001101_00010_00000000000000000001; //['loadi', '$r2', '1']
219 mem_ram[57] = 32'b000001_00011_00001_00010_00000000000; //['add', '$r3', '$r1', '$r2']
220 mem_ram[58] = 32'b001101_00001_00000000000000000100; //['loadi', '$r1', 4]
221 mem_ram[59] = 32'b001110_00011_00001_0000000000000000; //['store', '$r3', '$r1']
222 mem_ram[60] = 32'b010000_00000_00000000000000100001; //['jmpi', 'L5']
223 mem_ram[61] = 32'b001101_00001_00000000000000000010; //['loadi', '$r1', 2]
224 mem_ram[62] = 32'b001100_11100_00001_0000000000000000; //['load', '$r28', '$r1']
225 mem_ram[63] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
226 mem_ram[64] = 32'b001101_11100_00000000000000000000; //['loadi', '$r28', '0']
227 mem_ram[65] = 32'b010001_11111_00000000000000000000; //['jmp', '$r31']
228 mem_ram[66] = 32'b010110_11100_00000000000000000000; //['in', '$r28']
229 mem_ram[67] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
230 mem_ram[68] = 32'b001101_00010_00000000000000000101; //['loadi', '$r2', 5]
231 mem_ram[69] = 32'b001110_00001_00010_0000000000000000; //['store', '$r1', '$r2']
232 mem_ram[70] = 32'b100011_11111_11101_0000000000000000; //['push', '$r31', '$r29']
233 mem_ram[71] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
234 mem_ram[72] = 32'b001101_11110_00000000000000000101; //['loadi', '$r30', 5]
235 mem_ram[73] = 32'b001100_00001_11110_0000000000000000; //['load', '$r1', '$r30']
236 mem_ram[74] = 32'b100011_00001_11101_0000000000000000; //['push', '$r1', '$r29']
237 mem_ram[75] = 32'b000010_11101_11101_0000000000000001; //['addi', '$r29', '$r29', '1']
238 mem_ram[76] = 32'b100001_00000000000000000000000100; //['jal', 'fibonacci']
239 mem_ram[77] = 32'b000100_11101_11101_0000000000000001; //['subi', '$r29', '$r29', '1']
240 mem_ram[78] = 32'b100100_11111_11101_0000000000000000; //['pop', '$r31', '$r29']
241 mem_ram[79] = 32'b001111_00001_11100_0000000000000000; //['move', '$r1', '$r28']
242 mem_ram[80] = 32'b010111_00001_00000000000000000000; //['out', '$r1']
243 mem_ram[81] = 32'b011001_00000000000000000000000000; // halt
```

5 Conclusão

Grandes dificuldades foram emfrentadas durante o desemvolvimento do projeto. A que teve mais impacto foram as enfrentadas durante a criação do assembly, pois, soluções custosas foram necessárias como adição de novas instruções no processador e adição de pilha para tornar a recursão possível. Porém a experiência da construção de um compilador faz com que se consiga entender de uma forma bem mais profunda o funcionamento de um sitema computacional.

Referências

1 O que é um sistema computacional. Acessado em 05/04/2017. Disponível em: <https://www.portaleducacao.com.br/conteudo/artigos/informatica/o-que-e-um-sistema-computacional/46697>. Citado na página 4.

2 PATTERSON, D. A.; HENNESY, J. L. *Computer Organization and Design*. 5th edition. ed. Waltham/MA, EUA: Morgan Kaufmann, 2007. Citado 2 vezes nas páginas 4 e 5.

3 ENDEREçAMENTO de memória. Acessado em 06/04/2017. Disponível em: <http://usuarios.upf.br/~appel/arquiI/endereca.pdf>. Citado na página 5.

4 INTRODUçãO a verilog. Acessado em 05/04/2017. Disponível em: <http://www.asic-world.com/verilog/intro1.html#Introduction>. Citado na página 5.

5 VERILOG. Acessado em 07/04/2017. Disponível em: <https://pt.wikipedia.org/wiki/Verilog>. Citado na página 5.

6 ALTERA quartus. Acessado em 07/04/2017. Disponível em: <https://en.wikipedia.org/wiki/Altera_Quartus>. Citado na página 6.

7 LOUDEN, K. C. *Compiladores-Princípios e Práticas*. [S.l.]: Cengage Learning Editores, 2004. Citado na página 8.