André Filipe Siqueira Tokumoto

Desenvolvimento do Compilador: Produto final

São José dos Campos - Brasil

Julho de 2023

André Filipe Siqueira Tokumoto

Desenvolvimento do Compilador: Produto final

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 2023

Resumo

Nesse trabalho, buscou-se demonstrar as definições e etapas para o planejamento e implementação do compilador c-. No qual o compilador converte um código redigido na linguagem c- e o converte para um código binário, executável no processador MIPS monociclo, desenvolvido em projetos anteriores.

Palavras-chaves: Compilador. Linguagem c-. MIPS.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Na sociedade atual, cada vez mas digital, a computação ganha mais importância a cada dia, sendo assim, estudar e procurar melhorar os processos computacionais e fundamental.

Uma dessas etapas é a da compilação, na qual um algoritmo escrito em alto nível (para que os usuários humanos possam compreender), são traduzidas para a linguagem de maquina, para que possam ser processadas pelo computador.

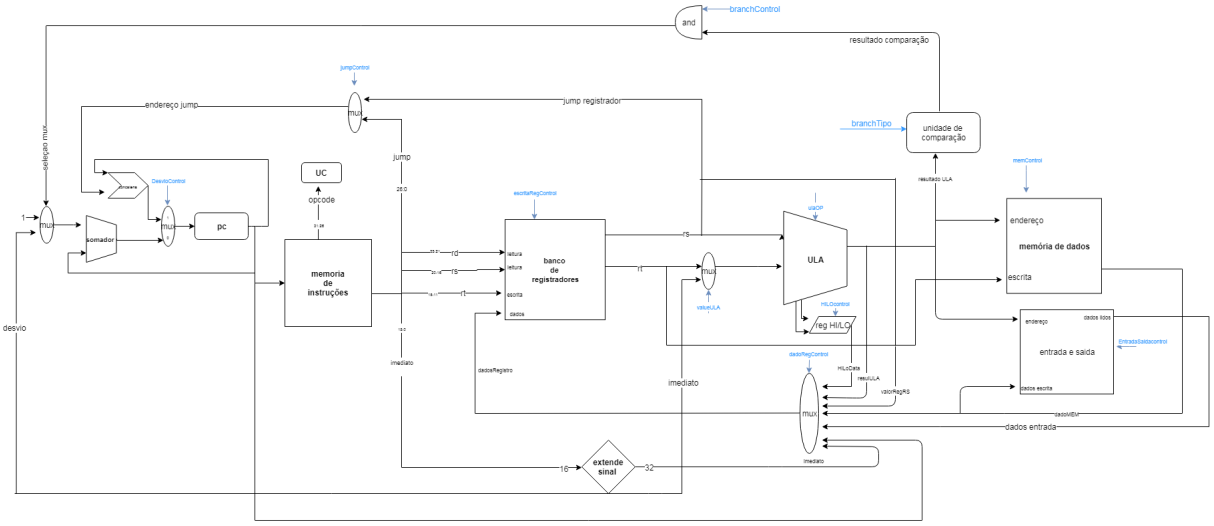
Desse modo, nesse projeto procurou-se estudar e entender e implementar na pratica a etapa de compilação de um algoritmo para uma linguagem de maquina que possa ser processada pela arquitetura já projetada anteriormente.

2 O Processador

2.1 Caminho de dados

Através do conjunto de instruções do formato de instruções e os modos de endereçamento pode ser desenvolvido um esquemático do caminho de dados da arquitetura que possibilite a execução de todas as instruções propostas.

Figura 1 – Esquemático da arquitetura



Fonte: O autor

Para instruções do tipo R, que são as de tipo lógica e aritméticas, nelas os endereços dos registradores onde estão os operandos e o endereço do registrador que ira guardar o resultado são mandadas para o banco de registradores ,e esses por sua vez mandam os valores contidos nesses registradores para a ULA que realiza a operação e manda o resultado de volta para o banco onde é salvo no registrador de destino.

Já para as instruções do tipo J, que são as de desvio incondicional, nela o valor da instrução de desvio é concatenada com os bits mais significativos do PC(registrador que guarda o endereço da próxima instrução) é mandado para o registrador de PC. Caso seja uma instrução do tipo jump and link, o valor do PC atual é mandado para o banco de registradores onde é salvo.

Já para as instruções do tipo I, é mandado para o banco de registradores os endereços de um registrador de operando e o registrador de destino e um imediato que é estendido de 16 para 32 bits e mandado para a ULA para que seja realizada a operação com o imediato e o valor do banco de registradores.

2.2 Explicação dos componentes do processador

2.2.1 ULA

Nessa unidade a cada vez que é recebido um valor de operando , a ULA realiza a operação indicada pelo sinal de controle e manda o resultado pela saida da ULA, e em caso de a operação for de multiplicação ou divisão é mandado o valor de HI e LO para os bancos especiais que os armazenam.

```
1  module ULA(clk,ulaOP,RS,RT, saidaULA , saidaHI,saidaLO);
2
3  input [31:0] RS,RT;
4  input  [4:0]ulaOP;
5  input clk;
6  output reg[31:0] saidaULA,saidaHI,saidaLO;
7
8  parameter soma=5'b00000,subtracao=4'b00001,multiplicacao=4'b00010,divisao=4'b00011,
       restoDivisao=4'b00100,OPor=4'b00101,OPand=4'b00110,OPnot=4'b00111;
9  parameter OPxor=4'b01000,OPnor=4'b01001,OPnand=4'b01010,OPxnor=4'b01011,maior=4'b01110;
10
11 always@(RS,RT)
12 begin
13   case(ulaOP)
14
15     soma: saidaULA = RS + RT;
16
17           subtracao : saidaULA = RS - RT;
18
19
20            multiplicacao:
21            begin
22             {saidaHI,saidaLO} = RS * RT;
23                   saidaULA = saidaLO;
24            end
25
26           divisao : saidaULA = RS / RT;
27
28           restoDivisao : saidaULA = RS % RT;
29
30           OPor : saidaULA = RS | RT;
31
32           OPand : saidaULA = RS & RT;
33
34           OPnot : saidaULA =  ~RS;
35
36           OPxor : saidaULA = RS ^ RT;
37
38           OPnor : saidaULA = ~(RS | RT);
39
40           OPnand : saidaULA = ~(RS & RT);
41
42     OPxnor : saidaULA = ~(RS ^ RT);
43
44   endcase
45 end
46 endmodule
```

2.2.2 Unidade de comparação

Na unidade de comparação a cada subida de clock eh verificado o sinal de controle, caso o sinal seja zero a unidade verifica se o valor recebido é zero, caso afirmativo a saida recebe valor um, caso contrario recebe zero. Se o sinal de controle for um a unidade tera saida um se o valor recebido for diferente de zero.

```
1
2  module unidadeDeComparacao(clk,branchTipo,resultadoSubtracao,resultadoComparacao);
3
4  input [31:0] resultadoSubtracao;
5  input clk;
6  input branchTipo;
7  output reg resultadoComparacao;
8
9  always@(posedge clk)
10   begin
11
12          case(branchTipo)
13
14            1'b0:
15             begin
16               if(resultadoSubtracao == 32'd0)
17                       resultadoComparacao = 1'b1;
18                  else
19                     resultadoComparacao = 1'b0;
20             end
21
22        1'b1:
23              begin
24               if(resultadoSubtracao != 32'd0)
25                       resultadoComparacao = 1'b1;
26                 else
27                       resultadoComparacao = 1'b0;
28              end
29
30
31        default: resultadoComparacao = 1'b0;
32
33          endcase
34
35   end
36
37 endmodule
```

2.2.3 Multiplexador de 7 entradas

Aqui temos a implementação de um multiplexador de 7 entradas que seleciona o valor que sera salvo no banco de registradores.

```
1  module mux6(dadoRegControl,HiLoData,resulULA,valorRegRS,dadoMEM,dadosEntrada,imediato,
       DadosRegistro);
2
3  input [2:0] dadoRegControl;
```

```
4  input [31:0] HiLoData,resulULA,valorRegRS,dadoMEM,dadosEntrada,imediato;
5  output reg [31:0] DadosRegistro;
6
7
8  always @(dadoRegControl)
9  begin
10
11 case(dadoRegControl)
12
13 3'b000:DadosRegistro = HiLoData;
14 3'b001:DadosRegistro=resulULA;
15 3'b010:DadosRegistro=valorRegRS;
16 3'b011:DadosRegistro=dadoMEM;
17 3'b100:DadosRegistro=dadosEntrada;
18 3'b101:DadosRegistro=imediato;
19
20 endcase
21
22 end
23
24 endmodule
```

2.2.4 Unidade de Controle

Essa é a unidade que controla os sinais de controle do processamento para a execução das instruções. Para isso, a unidade recebe o código de identificação da instrução (opcode), e a partir desse código seleciona os sinais de controle.

```
1  module UnidadeDeControle(opcode,status,ulaOP,valueULA,DesvioControl,jumpControl,
       linkControl,escritaRegControl,branchControl,branchTipo,dadoRegControl,memControl,
       HILOcontrol,entradaSaidaControl);
2
3  input [5:0] opcode;
4  output reg DesvioControl,HILOcontrol,branchControl,branchTipo,jumpControl,
       escritaRegControl,valueULA,linkControl,memControl;//sinal 1 bit
5  output reg status = 0;
6  output reg  [1:0] entradaSaidaControl;//sinal 2 bits
7  output reg  [2:0] dadoRegControl;//sinal de 3 bits
8  output reg  [4:0] ulaOP;//sinal 5 bits
9
10
11
12 //opcode de cada opera ao
13 parameter add=6'b000000,addi=6'b000001,sub=6'b000010,subi=6'b000011,mult=6'b000100,multi
       =6'b000101,div=6'b000110,divi=6'b000111,rdiv=6'b001000;
14 parameter OR=6'b001001,AND=6'b001010,NOT=6'b001011,XOR=6'b001100,NOR=6'b001101,NAND=6'
       b001110,XNOR=6'b001111,LT=6'b010000;
15 parameter jump=6'b010001,jumpR=6'b010010,jal=6'b010011,beq=6'b010100,bne=6'b010101,blt=6'
       b010110;
16 parameter lw=6'b010111,sw=6'b011000;
17 parameter mov=6'b011001,movi=6'b011010,mfhi=6'b011011,mflo=6'b011100;
18 parameter in=6'b011101,out=6'b011110,END=6'b011111,pause=6'b100000;
19
20
21
22
23 always@(*)
```

```
24  begin
25
26    case(opcode)
27
28          add:
29            begin
30
31                    DesvioControl = 1'b0;
32                    branchControl = 1'b0;
33                    branchTipo= 1'b0;
34                    jumpControl= 1'b0;
35                    escritaRegControl= 1'b1;
36                    valueULA= 1'b0;
37                    linkControl = 1'b0;
38                    memControl= 1'b0;
39                    entradaSaidaControl = 2'b00;
40                    dadoRegControl = 3'b001;
41                    ulaOP = 5'b00000;
42                    status=1'b0;
43
44                  end
45
46          addi:
47            begin
48
49                    DesvioControl = 1'b0;
50                    branchControl = 1'b0;
51                    branchTipo= 1'b0;
52                    jumpControl= 1'b0;
53                    escritaRegControl= 1'b1;
54                    valueULA= 1'b1;
55                    linkControl = 1'b0;
56                    memControl= 1'b0;
57                    entradaSaidaControl = 2'b00;
58                    dadoRegControl = 3'b001;
59                    ulaOP = 5'b00000;
60                    status=1'b0;
61
62                  end
63
64          sub:
65            begin
66
67                    DesvioControl = 1'b0;
68                    branchControl = 1'b0;
69                    branchTipo= 1'b0;
70                    jumpControl= 1'b0;
71                    escritaRegControl= 1'b1;
72                    valueULA= 1'b0;
73                    linkControl = 1'b0;
74                    memControl= 1'b0;
75                    entradaSaidaControl = 2'b00;
76                    dadoRegControl = 3'b001;
77                    ulaOP = 5'b00001;
78                    status=1'b0;
79
80                  end
81
82          subi:
83            begin
```

```
84
85                  DesvioControl = 1'b0;
86                  branchControl = 1'b0;
87                  branchTipo= 1'b0;
88                  jumpControl= 1'b0;
89                  escritaRegControl= 1'b1;
90                  valueULA= 1'b1;
91                  linkControl = 1'b0;
92                  memControl= 1'b0;
93                  entradaSaidaControl = 2'b00;
94                  dadoRegControl = 3'b001;
95                  ulaOP = 5'b00001;
96                  status=1'b0;
97
98                end
99
100
101         mult:
102           begin
103
104                 DesvioControl = 1'b0;
105                 branchControl = 1'b0;
106                 branchTipo= 1'b0;
107                 jumpControl= 1'b0;
108                 escritaRegControl= 1'b1;
109                 valueULA= 1'b0;
110                 linkControl = 1'b0;
111                 memControl= 1'b0;
112                 entradaSaidaControl = 2'b00;
113                 dadoRegControl = 3'b001;
114                 ulaOP = 5'b00010;
115                 status=1'b0;
116
117               end
118
119         multi:
120           begin
121
122                 DesvioControl = 1'b0;
123                 branchControl = 1'b0;
124                 branchTipo= 1'b0;
125                 jumpControl= 1'b0;
126                 escritaRegControl= 1'b1;
127                 valueULA= 1'b1;
128                 linkControl = 1'b0;
129                 memControl= 1'b0;
130                 entradaSaidaControl = 2'b00;
131                 dadoRegControl = 3'b001;
132                 ulaOP = 5'b00010;
133                 status=1'b0;
134
135               end
136
137         div:
138           begin
139
140                 DesvioControl = 1'b0;
141                 branchControl = 1'b0;
142                 branchTipo= 1'b0;
143                 jumpControl= 1'b0;
```

```
144                 escritaRegControl= 1'b1;
145                 valueULA= 1'b0;
146                 linkControl = 1'b0;
147                 memControl= 1'b0;
148                 entradaSaidaControl = 2'b00;
149                 dadoRegControl = 3'b001;
150                 ulaOP = 5'b00011;
151                 status=1'b0;
152
153               end
154
155
156       divi:
157         begin
158
159                 DesvioControl = 1'b0;
160                 branchControl = 1'b0;
161                 branchTipo= 1'b0;
162                 jumpControl= 1'b0;
163                 escritaRegControl= 1'b1;
164                 valueULA= 1'b1;
165                 linkControl = 1'b0;
166                 memControl= 1'b0;
167                 entradaSaidaControl = 2'b00;
168                 dadoRegControl = 3'b001;
169                 ulaOP = 5'b00011;
170                 status=1'b0;
171
172               end
173
174   rdiv:
175         begin
176
177                 DesvioControl = 1'b0;
178                 branchControl = 1'b0;
179                 branchTipo= 1'b0;
180                 jumpControl= 1'b0;
181                 escritaRegControl= 1'b1;
182                 valueULA= 1'b0;
183                 linkControl = 1'b0;
184                 memControl= 1'b0;
185                 entradaSaidaControl = 2'b00;
186                 dadoRegControl = 3'b001;
187                 ulaOP = 5'b00100;
188                 status=1'b0;
189
190               end
191
192       OR:
193         begin
194
195                 DesvioControl = 1'b0;
196                 branchControl = 1'b0;
197                 branchTipo= 1'b0;
198                 jumpControl= 1'b0;
199                 escritaRegControl= 1'b1;
200                 valueULA= 1'b0;
201                 linkControl = 1'b0;
202                 memControl= 1'b0;
203                 entradaSaidaControl = 2'b00;
```

```
204                 dadoRegControl = 3'b001;
205                 ulaOP = 5'b00101;
206                 status=1'b0;
207 
208               end
209 
210         AND:
211            begin
212 
213                 DesvioControl = 1'b0;
214                 branchControl = 1'b0;
215                 branchTipo= 1'b0;
216                 jumpControl= 1'b0;
217                 escritaRegControl= 1'b1;
218                 valueULA= 1'b0;
219                 linkControl = 1'b0;
220                 memControl= 1'b0;
221                 entradaSaidaControl = 2'b00;
222                 dadoRegControl = 3'b001;
223                 ulaOP = 5'b00110;
224                 status=1'b0;
225 
226               end
227 
228 
229     NOT:
230            begin
231 
232                 DesvioControl = 1'b0;
233                 branchControl = 1'b0;
234                 branchTipo= 1'b0;
235                 jumpControl= 1'b0;
236                 escritaRegControl= 1'b1;
237                 valueULA= 1'b0;
238                 linkControl = 1'b0;
239                 memControl= 1'b0;
240                 entradaSaidaControl = 2'b00;
241                 dadoRegControl = 3'b001;
242                 ulaOP = 5'b00111;
243                 status=1'b0;
244 
245               end
246 
247 
248     XOR:
249            begin
250 
251                 DesvioControl = 1'b0;
252                 branchControl = 1'b0;
253                 branchTipo= 1'b0;
254                 jumpControl= 1'b0;
255                 escritaRegControl= 1'b1;
256                 valueULA= 1'b0;
257                 linkControl = 1'b0;
258                 memControl= 1'b0;
259                 entradaSaidaControl = 2'b00;
260                 dadoRegControl = 3'b001;
261                 ulaOP = 5'b01000;
262                 status=1'b0;
263 
```

```
264                 end
265
266
267     NOR:
268             begin
269
270                     DesvioControl = 1'b0;
271                     branchControl = 1'b0;
272                     branchTipo= 1'b0;
273                     jumpControl= 1'b0;
274                     escritaRegControl= 1'b1;
275                     valueULA= 1'b0;
276                     linkControl = 1'b0;
277                     memControl= 1'b0;
278                     entradaSaidaControl = 2'b00;
279                     dadoRegControl = 3'b001;
280                     ulaOP = 5'b01001;
281                     status=1'b0;
282
283                 end
284
285
286      NAND:
287             begin
288
289                     DesvioControl = 1'b0;
290                     branchControl = 1'b0;
291                     branchTipo= 1'b0;
292                     jumpControl= 1'b0;
293                     escritaRegControl= 1'b1;
294                     valueULA= 1'b0;
295                     linkControl = 1'b0;
296                     memControl= 1'b0;
297                     entradaSaidaControl = 2'b00;
298                     dadoRegControl = 3'b001;
299                     ulaOP = 5'b01010;
300                     status=1'b0;
301
302                 end
303
304
305      XNOR:
306             begin
307
308                     DesvioControl = 1'b0;
309                     branchControl = 1'b0;
310                     branchTipo= 1'b0;
311                     jumpControl= 1'b0;
312                     escritaRegControl= 1'b1;
313                     valueULA= 1'b0;
314                     linkControl = 1'b0;
315                     memControl= 1'b0;
316                     entradaSaidaControl = 2'b00;
317                     dadoRegControl = 3'b001;
318                     ulaOP = 5'b01011;
319                     status=1'b0;
320
321                 end
322
323      LT:
```

```
324             begin
325
326                     DesvioControl = 1'b0;
327                     branchControl = 1'b0;
328                     branchTipo= 1'b0;
329                     jumpControl= 1'b0;
330                     escritaRegControl= 1'b1;
331                     valueULA= 1'b0;
332                     linkControl = 1'b0;
333                     memControl= 1'b0;
334                     entradaSaidaControl = 2'b00;
335                     dadoRegControl = 3'b001;
336                     ulaOP = 5'b01110;
337                     status=1'b0;
338
339                 end
340
341         jump:
342             begin
343
344                     DesvioControl = 1'b1;
345                     branchControl = 1'b0;
346                     branchTipo= 1'b0;
347                     jumpControl= 1'b0;
348                     escritaRegControl= 1'b0;
349                     valueULA= 1'b0;
350                     linkControl = 1'b0;
351                     memControl= 1'b0;
352                     status=1'b0;
353
354                 end
355
356          jumpR:
357             begin
358
359                     DesvioControl = 1'b1;
360                     branchControl = 1'b0;
361                     branchTipo= 1'b0;
362                     jumpControl= 1'b1;
363                     escritaRegControl= 1'b0;
364                     valueULA= 1'b0;
365                     linkControl = 1'b0;
366                     memControl= 1'b0;
367                     status=1'b0;
368
369                 end
370
371     jal:
372             begin
373
374                     DesvioControl = 1'b1;
375                     branchControl = 1'b0;
376                     branchTipo= 1'b0;
377                     jumpControl= 1'b0;
378                     escritaRegControl= 1'b1;
379                     valueULA= 1'b0;
380                     linkControl = 1'b1;
381                     memControl= 1'b0;
382                     dadoRegControl = 3'b110;
383                     status=1'b0;
```

```
384
385                 end
386
387
388
389     beq:
390             begin
391
392                     DesvioControl = 1'b0;
393                     branchControl = 1'b1;
394                     branchTipo= 1'b0;
395                     jumpControl= 1'b0;
396                     escritaRegControl= 1'b0;
397                     valueULA= 1'b0;
398                     linkControl = 1'b0;
399                     memControl= 1'b0;
400                     entradaSaidaControl = 2'b00;
401                     ulaOP = 5'b00001;
402                     status=1'b0;
403
404                 end
405
406
407      bne:
408             begin
409
410                     DesvioControl = 1'b0;
411                     branchControl = 1'b1;
412                     branchTipo= 1'b1;
413                     jumpControl= 1'b0;
414                     escritaRegControl= 1'b0;
415                     valueULA= 1'b0;
416                     linkControl = 1'b0;
417                     memControl= 1'b0;
418                     entradaSaidaControl = 2'b00;
419                     ulaOP = 5'b00001;
420                     status=1'b0;
421
422                 end
423
424
425       blt:
426             begin
427
428                     DesvioControl = 1'b0;
429                     branchControl = 1'b1;
430                     branchTipo= 1'b1;
431                     jumpControl= 1'b0;
432                     escritaRegControl= 1'b0;
433                     valueULA= 1'b0;
434                     linkControl = 1'b0;
435                     memControl= 1'b0;
436                     entradaSaidaControl = 2'b00;
437                     ulaOP = 5'b01110;
438                     status=1'b0;
439
440                 end
441
442
443      lw:
```

```
444             begin
445
446                     DesvioControl = 1'b0;
447                     branchControl = 1'b0;
448                     jumpControl= 1'b0;
449                     escritaRegControl= 1'b1;
450                     valueULA= 1'b1;
451                     linkControl = 1'b0;
452                     memControl= 1'b0;
453                     entradaSaidaControl = 2'b00;
454                     dadoRegControl = 3'b011;
455                     ulaOP = 5'b00010;
456                     status=1'b0;
457
458                 end
459
460           sw:
461             begin
462
463                     DesvioControl = 1'b0;
464                     branchControl = 1'b0;
465                     jumpControl= 1'b0;
466                     escritaRegControl= 1'b0;
467                     valueULA= 1'b1;
468                     linkControl = 1'b0;
469                     memControl= 1'b1;
470                     entradaSaidaControl = 2'b00;
471                     ulaOP = 5'b00010;
472                     status=1'b0;
473
474                 end
475
476           mov:
477             begin
478
479                     DesvioControl = 1'b0;
480                     branchControl = 1'b0;
481                     branchTipo= 1'b0;
482                     jumpControl= 1'b0;
483                     escritaRegControl= 1'b1;
484                     valueULA= 1'b0;
485                     linkControl = 1'b0;
486                     memControl= 1'b0;
487                     entradaSaidaControl = 2'b00;
488                     dadoRegControl = 3'b010;
489                     status=1'b0;
490
491                 end
492
493
494           movi:
495             begin
496
497                     DesvioControl = 1'b0;
498                     branchControl = 1'b0;
499                     branchTipo= 1'b0;
500                     jumpControl= 1'b0;
501                     escritaRegControl= 1'b1;
502                     valueULA= 1'b0;
503                     linkControl = 1'b0;
```

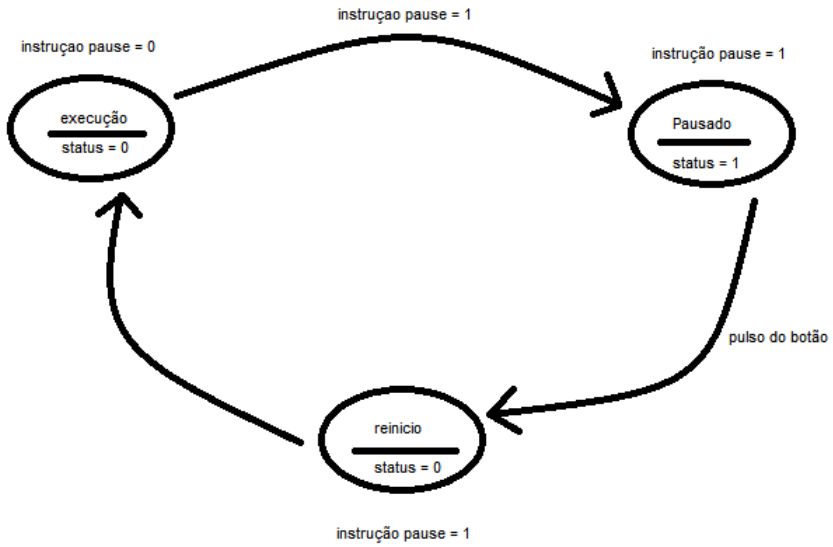
```
504                   memControl= 1'b0;
505                   entradaSaidaControl = 2'b00;
506                   dadoRegControl = 3'b101;
507                   status=1'b0;
508
509                 end
510
511           mfhi:
512             begin
513
514                   DesvioControl = 1'b0;
515                   branchControl = 1'b0;
516                   branchTipo= 1'b0;
517                   jumpControl= 1'b0;
518                   escritaRegControl= 1'b1;
519                   valueULA= 1'b0;
520                   linkControl = 1'b0;
521                   memControl= 1'b0;
522                   entradaSaidaControl = 2'b00;
523                   HILOcontrol = 1'b1;
524                   dadoRegControl = 3'b000;
525                   status=1'b0;
526
527                 end
528
529         mflo:
530             begin
531
532                   DesvioControl = 1'b0;
533                   branchControl = 1'b0;
534                   branchTipo= 1'b0;
535                   jumpControl= 1'b0;
536                   escritaRegControl= 1'b1;
537                   valueULA= 1'b0;
538                   linkControl = 1'b0;
539                   memControl= 1'b0;
540                   entradaSaidaControl = 2'b00;
541                   HILOcontrol = 1'b0;
542                   dadoRegControl = 3'b000;
543                   status=1'b0;
544
545                 end
546
547   in:
548    begin
549
550                   DesvioControl = 1'b0;
551                   branchControl = 1'b0;
552                   branchTipo= 1'b0;
553                   jumpControl= 1'b0;
554                   escritaRegControl= 1'b1;
555                   valueULA= 1'b0;
556                   linkControl = 1'b0;
557                   memControl= 1'b0;
558                   entradaSaidaControl = 2'b10;
559                   dadoRegControl = 3'b100;
560                   status=1'b1;
561                   ulaOP = 5'b00010;
562
563                 end
```

```
564
565   out:
566     begin
567
568                   DesvioControl = 1'b0;
569                   branchControl = 1'b0;
570                   branchTipo= 1'b0;
571                   jumpControl= 1'b0;
572                   escritaRegControl= 1'b0;
573                   valueULA= 1'b1;
574                   linkControl = 1'b0;
575                   memControl= 1'b0;
576                   entradaSaidaControl = 2'b01;
577                   dadoRegControl = 3'b000;
578                   ulaOP = 5'b00010;
579                   status=1'b0;
580
581                 end
582
583
584    pause:
585      begin
586               escritaRegControl= 1'b0;
587               jumpControl= 1'b0;
588               DesvioControl = 1'b0;
589                    branchControl = 1'b0;
590               memControl= 1'b0;
591               entradaSaidaControl = 2'b00;
592                    status=1'b1;
593           end
594
595
596  endcase
597
598
599  end
600
601
602 endmodule
```

2.2.5 Unidade de parada do sistema

Nessa unidade é controlado o estado do processamento (em execução e pausa), para isso a unidade recebe um sinal da unidade de controle que indica qual o estado do sistema para aquela instrução. Caso o sistema esteja pausado, o mesmo deve mudar de estado quando houver um pulso de entrada. Para isso foi utilizado o conceito de maquina de estados de mealy, mostrados no diagrama de estados (figura 2).

Figura 2 – Diagrama de estados da unidade de parada



Fonte: O autor

A partir do diagrama foi implementada a unidade.

```
1  module ParadaSistema(clock,pausa,botaoIN,status);
2
3    input botaoIN,clock;
4    input pausa;
5    output reg status;
6
7    reg [1:0] estadoAtual,estadoFuturo;
8
9    always@(pausa)
10     begin
11
12             case(estadoAtual)
13
14                   2'b00://sistema em execu ao
15                         begin
16
17                                if(pausa)//intru ao para pausa
18                                  begin
19                                    estadoFuturo = 2'b01;
20                                         status = 1'b1;
21                                  end
22
23                                else //mantem estado
24                              begin
25                                      estadoFuturo = 2'b00;
26                                           status = 1'b0;
27                                  end
28                          end
29
30                        2'b01://sistema pausado
31                    begin
```

```
32
33                              if(pausa)//mantem estado
34                                    begin
35                                      estadoFuturo = 2'b01;
36                                           status = 1'b1;
37                                    end
38
39                                   else
40                                     begin
41                                            estadoFuturo = 2'b00;
42                                            status = 1'b1;
43                                          end
44
45                             end
46
47                            2'b10://executa com a instru ao de pausa
48                              begin
49                                 estadoFuturo = 2'b00;
50                            status = 1'b0;
51                    end
52
53                     endcase
54            end
55
56
57            always@(posedge botaoIN or posedge clock)
58             begin
59                 if(botaoIN) estadoAtual = 2'b10;
60                           else estadoAtual = estadoFuturo;
61             end
62
63
64
65
66  endmodule
```

2.2.6 Unidade de entrada e saída de dados

Essa é a implementação da unidade que controla a entrada e saída de dados do sistema.

```
1   module EntradaSaida(botaoIN,endereco,dadosEscrita,DadosLidos,entradaSaidaControl,clk,
        entradaDeDados,unidade,dezena,centena);
2
3   input [31:0] endereco,dadosEscrita;
4   input [3:0]  entradaDeDados;
5   input [1:0]  entradaSaidaControl;
6   input clk,botaoIN;
7
8   reg [31:0] saidaDeDados;
9   output wire [3:0] unidade,dezena,centena;
10
11  output reg [31:0] DadosLidos;
12
13
14  BCD bcd(.binario(saidaDeDados),.unidade(unidade),.dezena(dezena),.centena(centena));
15
16
```

```
17  always@(negedge clk)//saida
18   begin
19
20    if(entradaSaidaControl==2'b01) saidaDeDados = dadosEscrita;
21
22   end
23
24
25
26   always@(posedge clk)//entrada
27    begin
28           if(entradaSaidaControl==2'b10) DadosLidos = {28'b0000000000000000000000000000,
                 entradaDeDados};
29         end
30
31 endmodule
```

2.2.7 Modulo de decodificação dos displays de 7 segmentos

Aqui é implementado a unidade de codificação dos valores de saída para o display de 7 segmentos do kit.

```
1  module displaySete(entrada,saidas);
2
3   output reg [6:0] saidas;
4   input[3:0] entrada;
5   reg  [6:0] segmentos;
6
7   always @(entrada)
8    begin
9     case(entrada)
10     4'b0000: segmentos=7'b0000001;
11     4'b0001: segmentos=7'b1001111;
12     4'b0010: segmentos=7'b0010010;
13     4'b0011: segmentos=7'b0000110;
14     4'b0100: segmentos=7'b1001100;
15     4'b0101: segmentos=7'b0100100;
16     4'b0110: segmentos=7'b0100000;
17     4'b0111: segmentos=7'b0001111;
18     4'b1000: segmentos=7'b0000000;
19     4'b1001: segmentos=7'b0000100;
20         default: segmentos = 7'b1111111;
21    endcase
22
23     {saidas} = segmentos;
24
25   end
26 endmodule
```

2.2.8 Unidade de processamento

Finalmente nessa etapa todas as unidades são ligadas na unidade de processamento da maneira representada no caminho de dados.

```
1  module CPU(reset,clock,botaoPlaca,entradaDeDadosIO,unidade,dezena,centena);
2
```

```
3  input clock,botaoPlaca,reset;
4  input [3:0] entradaDeDadosIO;
5
6  reg [31:0] concatena;
7  reg [31:0] resulSomador;
8  reg [31:0] imediatoExtendido;
9  reg [31:0] HILOdata;
10 reg [31:0] dadosEscrita;
11 reg [31:0] regHI,regLO;
12 reg [31:0] dadosRegistro;
13 reg [31:0] PC;
14 reg [31:0] operando;
15 wire[3:0] inUnidade,inDezena,inCentena;
16
17
18 wire botaoIN;
19 wire selecaoMuxDesvio;
20 wire parada;
21 wire status;
22 wire [5:0] opcode;
23 wire [4:0] endRD,endRS,endRT;
24 wire [31:0] RS,RT;
25 wire [31:0] resultadoULA;
26 wire [31:0] HI,LO;
27 wire [31:0] dadoMem;
28 wire [31:0] dadosDeEntrada;
29 wire resultComparacao;
30 wire [31:0] jump;
31 wire [31:0] dadosMux6;
32 wire [10:0] imediato;
33
34
35 wire  [1:0] entradaSaidaControl;
36 wire valueULA;
37 wire  DesvioControl,branchControl,branchTipo,jumpControl,linkControl,memControl;
38 wire HILOcontrol,escritaRegControl;//sinal 1 bit
39 wire  [2:0] dadoRegControl;//sinal de 3 bits
40 wire  [4:0] ulaOP;//sinal 5 bits
41
42 output wire [6:0] unidade,dezena,centena;
43
44
45
46
47 //liga ao com pulso botao
48 monostable mon(.clk(clock),.reset(reset),.trigger(botaoPlaca),.pulse(botaoIN));
49
50 //liga ao com memoria de instru oes
51 MEMInstrucoes inst(.pc(PC),.opcode(opcode),.jump(jump),.OUTrs(endRS),.OUTrt(endRT),.OUTrd
       (endRD),.imediato(imediato),.clock(clock));
52
53 //liga ao com unidade de controle
54 UnidadeDeControle UC(.opcode(opcode),.status(status),.ulaOP(ulaOP),.valueULA(valueULA),.
       DesvioControl(DesvioControl),.jumpControl(jumpControl),.linkControl(linkControl),.
       escritaRegControl(escritaRegControl),.branchControl(branchControl),.branchTipo(
       branchTipo),.dadoRegControl(dadoRegControl),.memControl(memControl),.HILOcontrol(
       HILOcontrol),.entradaSaidaControl(entradaSaidaControl));
55
56 //liga ao com  parada de sistema
57 ParadaSistema mest(.clock(clock),.pausa(status),.botaoIN(botaoIN),.status(parada));
```

```
58
59 //liga ao com banco registradores
60 BancoRegistradores BR(.clk(clock),.escritaRegControl(escritaRegControl),.inRS(endRS),.
       inRT(endRT),.inRD(endRD),.dados(dadosMux6),.outRS(RS),.outRT(RT),.linkControl(
       linkControl));
61
62 //liga ao com ULA
63 ULA alu(.ulaOP(ulaOP),.RS(RS),.RT(operando),.saidaULA(resultadoULA),.saidaHI(HI),.saidaLO
       (LO));
64
65 //liga ao com unidade de compara ao
66 unidadeDeComparacao compara(.branchTipo(branchTipo),.resultadoULA(resultadoULA),.
       resultadoComparacao(resultComparacao));
67
68 //ligacao mux6
69 mux6 muxRegistro(.dadoRegControl(dadoRegControl),.HiLoData(HILOdata),.resulULA(
       resultadoULA),.valorRegRS(RS),.dadoMEM(dadoMem),.dadosEntrada(dadosDeEntrada),.
       imediato(imediatoExtendido),.PC(PC),.DadosRegistro(dadosMux6));
70
71 //liga ao com memoria de dados
72 simple_dual_port_ram_single_clock memPrincipal(.data(RT),.read_addr(resultadoULA),.
       write_addr(resultadoULA),.we(memControl),.clk(clock),.q(dadoMem));
73
74 //liga ao com entrada e saida
75 EntradaSaida IO(.botaoIN(botaoIN),.endereco(resultadoULA),.dadosEscrita(dadoMem),.
       DadosLidos(dadosDeEntrada),.entradaSaidaControl(entradaSaidaControl),.clk(clock),.
       entradaDeDados(entradaDeDadosIO),.unidade(inUnidade),.dezena(inDezena),.centena(
       inCentena));
76
77 //liga ao com display
78  displaySete displayUnidade(.entrada(inUnidade),.saidas(unidade));
79  displaySete displayDezena(.entrada(inDezena),.saidas(dezena));
80  displaySete displayCentena(.entrada(inCentena),.saidas(centena));
81
82 assign selecaoMuxDesvio = branchControl & resultComparacao;
83
84
85
86
87 //
       ************************************************************************************************
88  always@(negedge clock) //valores que vao entrar no mux de sele   o da proxima
       instru   o
89
90
91    begin
92
93
94                  if(selecaoMuxDesvio) resulSomador = PC + imediatoExtendido;
95                    else  resulSomador = PC + 32'd1;
96
97                  if(jumpControl) concatena = {PC[31:26],RS[25:0]};//concatenacao para o
                        jump registrador
98                    else concatena = {PC[31:26],jump};//concatenacao para o jump
99
100         end
101
102
103
```

```
104 //
    ***********************************************************************************************

105
106 always@(posedge clock or posedge reset)
107  begin
108
109
110
111               //atualiza  o de pc
112                  if(reset) PC=32'd0;
113
114                       else
115                         begin
116
117                                 if(parada) PC = PC;
118
119                       else
120                               begin
121
122                                 if(DesvioControl) PC <= concatena;  //jump
123                                 else PC = resulSomador;
124
125                             end
126                         end
127
128
129  end
130
131 //
    ***********************************************************************************************

132  always@(*)//extensor de bits
133  begin
134
135     if(imediato[10]==1'b0) imediatoExtendido = {21'b000000000000000000000,imediato};
136     else imediatoExtendido = {21'b111111111111111111111,imediato};
137
138
139  end
140
141 //
    ***********************************************************************************************

142
143  //gerenciamento do banco reservado de Hi e LO
144  always@(HI,LO)//salva dados no banco de hi e lo sempre que houver as opera  es de
        divisao e multiplica  o
145  begin
146     if((ulaOP==5'b00010) | (ulaOP==5'b00011))
147            begin
148          regHI = HI;
149                    regLO = LO;
150                end
151  end
152
153 //
    ***********************************************************************************************

154
```

```
155   always@(*)//sele ao do valor que vai para o mux de sele   o para o banco de
          registradores em mfHI,mfLO
156    begin
157
158
159           if(HILOcontrol) HILOdata = regHI;
160           else  HILOdata = regLO;
161          end
162
163
164  //
         ************************************************************************************************
165
166  always@(*)//selecao de valor que vai para a ula(imediato ou RT)
167   begin
168
169      if(valueULA) operando = imediatoExtendido;
170            else operando = RT;
171
172
173
174   end
175 //
        ************************************************************************************************
176 endmodule
```

2.3 Conjunto de Instruções

2.3.1 Formato das instruções

Para as instruções foram definidos os formatos já existentes na arquitetura MIPS, com algumas modificações.

Figura 3 – Tipos de Instruções

Tipo R :	opcode	reg destino	reg origem	reg origem	funct	-
	6 bits	5 bits	5 bits	5 bits	6 bits	5 bits

Tipo I :	opcode	reg destino	reg origem	imediato/endereço
	6 bits	5 bits	5 bits	16 bits

Tipo J :	opcode	endereço
	6 bits	26 bits

Fonte: O autor

2.3.2 Modos de endereçamento

Quanto ao modo de endereçamento foram definidos os modos : por imediato , por registrador e por deslocamento. Na qual o endereçamento por imediato é utilizado para instruções do tipo J e para algumas do tipo I, já o endereçamento por registrador é utilizado para instruções do tipo R e do tipo I. Enquanto que o endereçamento por deslocamento é empregado para instruções de acesso a memória.

2.3.3 Tabela do conjunto de instruções

Foi definido que para o projeto seria utilizado um conjunto de instruções RISC, *Reduced Instruction Set Computer*, baseado nas instruções da arquitetura MIPS monociclo.

Assim foram desenvolvidas nove instruções para operações aritméticas, sete para operações lógicas, sete instruções de desvio, duas de acesso a memória, duas de transferência de dados, duas para a comunicação com usuário e uma para encerrar a execução.

Tabela 1 – Conjunto de instruções

OPCODE	INSTRUÇÂO	Descrição	Sintaxe	Tipo
Aritmeticas				
000000	add	adição	add rd,rs,rt	r
000001	addi	adição com imediato	addi rd,rs,imediato	r
000010	sub	subtração	sub rd,rs,rt	r
000011	subi	subtração com imediato	subi rd,rs,imediato	r
000100	mult	multiplicação	mult rd,rs,rt	r
000101	multi	multiplicação por imediato	multi rd,rs,imediato	r
000110	div	divisão	div rd,rs,rt	r
000111	divi	divisão por imediato	divi rd,rs,imediato	r
001000	rdiv	resto da divisão	rdiv rd,rs,rt	r
Lógicas				
001001	or	operação lógica or	or rd,rs,rt	i
001010	and	operação lógica and	and rd,rs,rt	i
001011	not	operação lógica not	not rd,rs	i
001100	xor	operação lógica xor	xor rd,rs,rt	i
001101	nor	operação lógica nor	nor rd,rs,rt	i
001110	nand	operação lógica nand	nand rd,rs,rt	i
001111	xnor	operação lógica xnor	xnor rd,rs,rt	i
010000	lt	retorna se um valor eh menor que o outro	lt rd,rs,rt	i
Desvio				
010001	j	desvio incondicional	j ENDEREÇO	j
010010	jr	jump para endereço de registrador	jr rs	j
010011	jal	jump and link	jal ENDEREÇO	j
010100	beq	desvio condicional por igualdade	beq rs,rt,ENDEREÇO	i
010101	bne	desvio condicional por desigualdade	bne rs,rt,ENDEREÇO	i
010110	blt	desvio se é menor que registrador	blt rs,rt,ENDEREÇO	i
Acesso a memória				
010111	lw	carrega da memória	lw s1,CONST(reg)	I
011000	sw	escreve na memória	sw s1,CONST(reg)	I
Transferência				
011001	mov	move dados entre registradores	mov rd,rd	r
011010	movi	atribui valor de imediato para registrador	movi rd,imediato	r
011011	mfhi	move dados do registrador HI para o banco	mfhi rd	r
011100	mflo	move dados do registrador LO para o banco	mflo rd	r
Entrada e Saída				
011101	in	entrada de dados do usuario	in rd	r
011110	out	saida de dados para o usuario	out rd,end_Saida	r
011111	end	encerra execução	end	r
100000	pause	pausa a execução	pause	r

Fonte: O autor

2.4 Organização da memória

2.4.1 Memória de instruções

Para o modulo da memoria de instruções, há como entrada o valor de PC e como saídas os valores de opcode,endereço de jump, endereço dos registradores de operandos e o de destino, além do imediato de 16 bits.

```
1  module MEMInstrucoes(pc,opcode,jump,OUTrs,OUTrt,OUTrd,imediato);
2
3  input [31:0] pc;
4  output reg[5:0] opcode;
5  output reg[4:0] OUTrs,OUTrt,OUTrd;
6  output reg[15:0] imediato;
7  output reg[25:0] jump;
8
9  reg [31:0] instrucao;
10 reg [31:0] memoria[31:0];
11
12
13 always @(pc)
14  begin
15           instrucao = memoria[pc];
16
17           opcode     =   instrucao[31:26];
18           jump       =   instrucao[25:0];
19           OUTrs      =   instrucao[25:21];
20           OUTrt      =   instrucao[20:16];
21           OUTrd      =   instrucao[15:11];
22           imediato   =   instrucao[15:0];
23
24  end
25
26 endmodule
```

2.4.2 Banco de registradores

Nesse modulo a cada subida do clock é enviado pelas saídas outRS e outRT os valores contidos nos registradores selecionados (através da instrução) e a cada descida do clock, caso o sinal de escrita esteja ativo, é salvo o valor da entrada no resgistrador de destino selecionado.

```
1  module BancoRegistradores(clk,escritaRegControl,inRS,inRT,inRD,dados,outRS,outRT);
2
3  input clk,escritaRegControl,linkControl;
4  input [4:0] inRS,inRT,inRD;
5  input [31:0] dados;
6  output wire[31:0]  outRS,outRT;
7
8  reg [31:0] bancoDeRegistradores[31:0];
9
10
```

```
11
12          assign outRS = bancoDeRegistradores[inRS];
13          assign outRT = bancoDeRegistradores[inRT];
14
15
16
17   always@(negedge clk)//toda descida de clock
18   begin
19
20     if(escritaRegControl) bancoDeRegistradores[inRD] = dados;
21
22   end
23
24
25  endmodule
```

2.4.3 Memória de dados

Para o modulo da unidade de dados foi utilizado o modelo de template disponível no quartus,com algumas modificações.

```
1
2  // Quartus Prime Verilog Template
3  // Simple Dual Port RAM with separate read/write addresses and
4  // single read/write clock
5
6  module simple_dual_port_ram_single_clock
7  #(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=32)
8  (
9          input [(DATA_WIDTH-1):0] data,
10         input [(ADDR_WIDTH-1):0] read_addr, write_addr,
11         input we, clk,
12         output reg [(DATA_WIDTH-1):0] q
13 );
14
15         // Declare the RAM variable
16         reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
17
18         always @ (posedge clk)
19         begin
20
21
22                         q <= ram[read_addr];
23
24                 // Read (if read_addr == write_addr, return OLD data).  To return
25                 // NEW data, use = (blocking write) rather than <= (non-blocking write)
26                 // in the write assignment.      NOTE: NEW data may require extra bypass
27                 // logic around the RAM.
28
29         end
30
31         always@(negedge clk)
32          begin
33                 // Write
34            if (we)
35                 ram[write_addr] <= data;
```

```
36
37          end
38
39
40  endmodule
```

3 Compilador: Fase de Análise

A fase de análise é responsável por ler o código fonte e valida-do ou rejeita-lo a partir da gramatica da linguagem na qual ele foi escrito. Essa etapa é divida em três partes: análise léxica, análise sintática e análise semântica, que serão explicadas mais pra frente.

Figura 4 – Processo de compilação

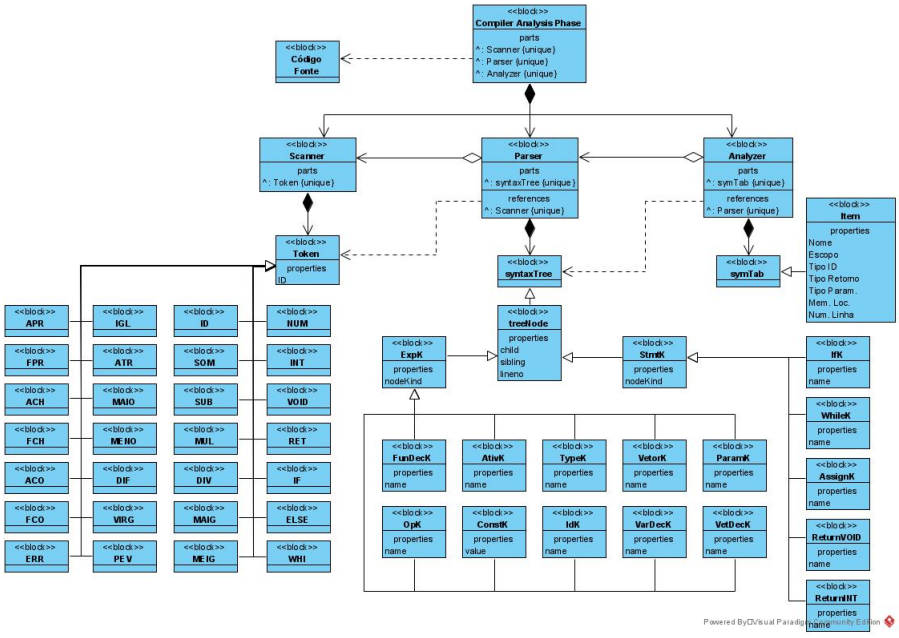
Fonte: Material de aula UFRGS (1)

3.1 Modelagem

Antes do inicio da implementação da fase de análise, a mesma foi modelada em diagramas, apresentados abaixo.

3.1.1 Diagrama de blocos

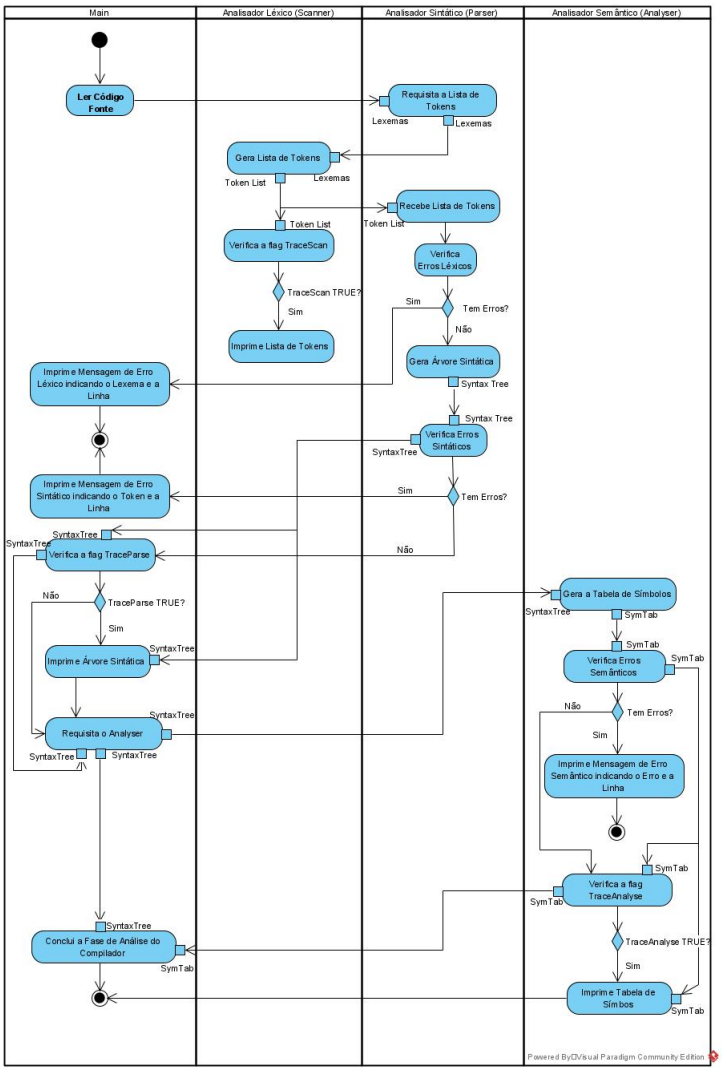
Figura 5 – Diagrama de blocos da fase de análise



Fonte: O autor

3.1.2 Diagrama de atividades

Figura 6 – Diagrama de atividades da fase de análise



Fonte: O autor

3.2 Análise Léxica

A fase de varredura, ou analise léxica da compilação, é responsável por percorrer o código fonte, e conferir a cada lexema "*Marcas(tokens)*", que são palavras que identificam os lexemas no processo (2). Os lexemas reconhecidos são:

- Palavras reservadas: else, if, int, return, void, while
- Símbolos: + - * / < <= > >= == != = ; , [ ] /* */
- Identificadores: lexemas que não são palavras reservadas
- Números

Essa unidade foi desenvolvida a a partir da ferramenta *flex (scanner.l)*, disponibilizada na entrega.

3.3 Análise Sintática

A análise sintática é responsável por determinar se o código fonte respeita todas as *regras gramaticais* da linguagem analisada e montar a arvore sintática que será usada nas etapas seguintes da compilação (2).

Figura 7 – Exemplo de árvore sintática

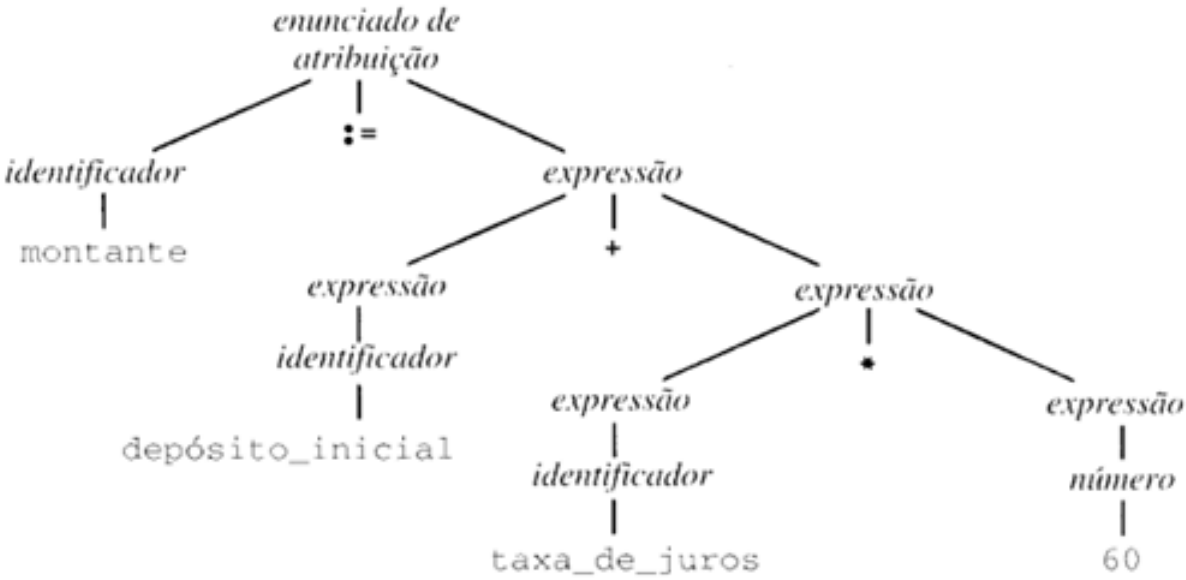


Fig. 1.4 Árvore gramatical para `montante := depósito_inicial + taxa_de_juros * 60`.

Fonte: Compiladores Princípios, Técnicas e Ferramentas (3)

Essa unidade foi desenvolvida a a partir da ferramenta *bison*, disponibilizada na entrega.

3.4 Análise Semântica

A análise semântica é responsável por verificar aspectos relacionados ao significado da instrução, esta é a terceira etapa do processo de compilação, onde são verificadas uma série de regras que não podem ser verificadas nas etapas anteriores (4). Nesse projeto as regras verificadas são:

- Compatibilidade do retorno de uma função

- Múltipla declaração de uma variável
- Ausência da declaração de uma variável
- Ausência da declaração de uma função
- Compatibilidade dos parâmetros de uma função

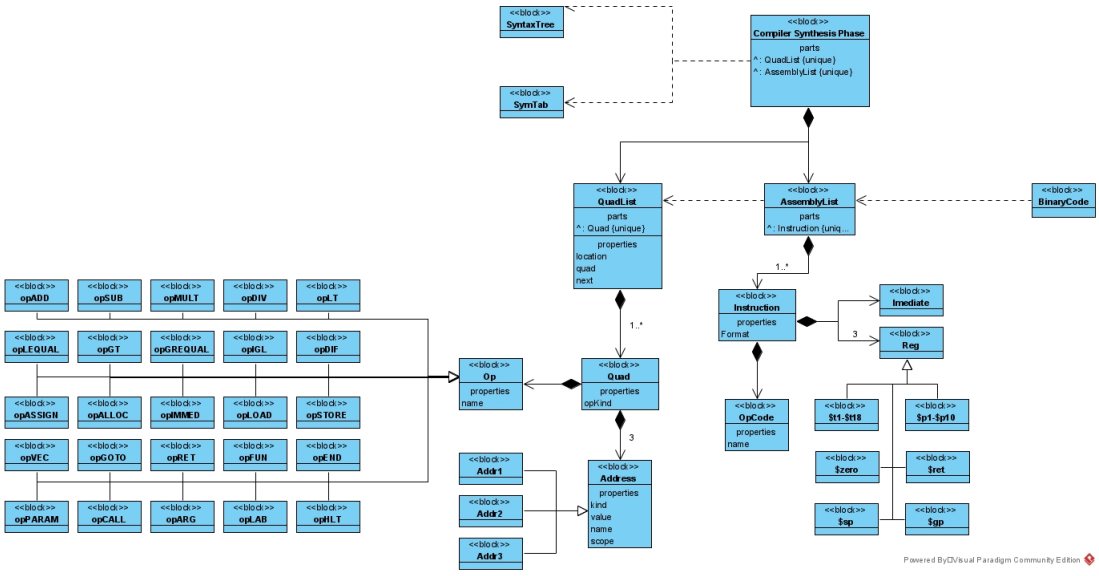
4 Compilador: Fase de Síntese

A fase de síntese é a ultima etapa do processo de compilação, na qual o compilador constrói o código executável na maquina, a partir dos elementos gerados na fase anterior (3).

4.1 Modelagem

4.1.1 Diagrama de blocos

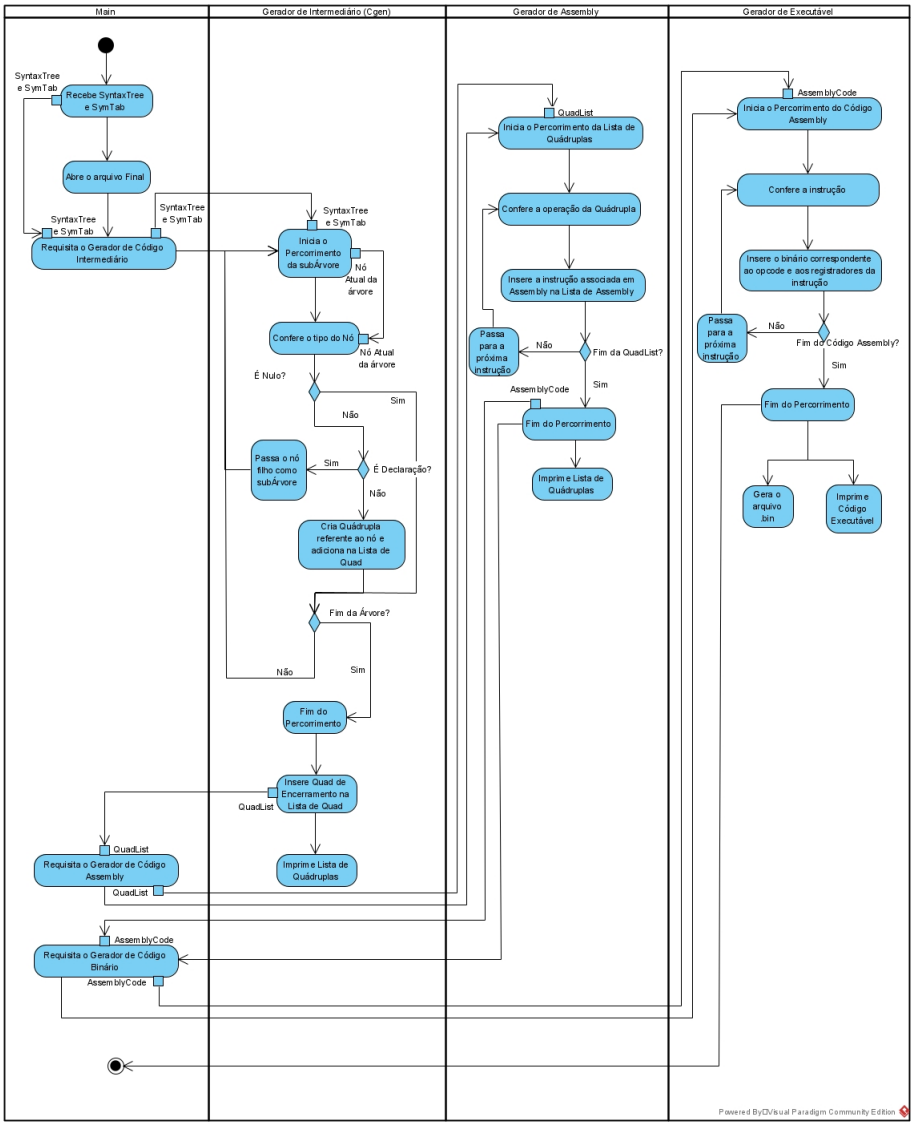
Figura 8 – Diagrama de blocos da fase de síntese



Fonte: O autor

4.1.2 Diagrama de atividades

Figura 9 – Diagrama de atividades da fase de síntese



Fonte: O autor

4.2 Geração de código intermediário

O código intermediário, é uma estrutura construída a partir da arvore de analise sintática, que está mais próxima do código alvo (2).

Tabela 2 – Conjunto de quadruplas

Quadruplas					
Operação	**addr1**	**addr2**	**addr3**	**addr4**	**descrição**
opADD	add	$r1	$r2	$r3	soma entre r1 e r2 (r3 = r1 + r2)
opSUB	sub	$r1	$r2	$r3	subtração entre r1 e r2 (r3 = r1 - r2)
opMULT	mult	$r1	$r2	$r3	multiplicação entre r1 e r2 (r3 = r1 * r2)
opDIV	div	$r1	$r2	$r3	divisão entre r1 e r2 (r3 = r1 / r2)
opLT	bgeq	$r1	$r2	L0	if r1 < r2 não desvia para L0
opLEQUAL	bgt	$r1	$r2	L0	if r1 <= r2 não desvia para L0
opGT	bleq	$r1	$r2	L0	if r1 > r2 não desvia para L0
opGREQUAL	blt	$r1	$r2	L0	if r1 >= r2 não desvia para L0
opIGL	beq	$r1	$r2	L0	if r1 == r2 não desvia para L0
opDIF	bne	$r1	$r2	L0	if r1 != r2 não desvia para L0
opASSING	atrib	$r1	$r2	-	atribui r2 a r1 (r1=r2)
opALLOC	alloc	nome variavel	memLoc	escopo	aloca var na posição memLoc da memória
opIMMED	immed	$r1	imediato	-	coloca o imediato em um reg (r1 = imediato)
opLOAD	load	$r1	nome variavel	memLoc	coloca um valor da mem em um reg (r1 = memLoc)
opSTORE	store	nome variavel	memLoc/posVec	$r1	coloca um valor de um reg na mem (memLoc = r1)
opVEC	vec	$r1	nome variavel	posVec	atribui um vetor a um reg (r1 = var[posVec])
opGOTO	goto	L0	-	-	salta para a posição da label L0 (j L0)
opRET	ret	$r1	-	-	indica o reg que armazena o retorno da função
opFUN	fun	nome função	memLoc	-	aloca fun na posição memLoc da memória
opEND	end	nome função	-	-	indica o fim da função
opPARAM	param	$r1	-	-	indica que r1 é um parametro
opCALL	call	$r1	nome função	qtd parametros	indica que r1 vai armazenar o retorno da função e possui qtd parametros
opARG	arg	nome argumento	memLoc	nome função	aloca o arg na posição memLoc da memória
opLAB	label	L0	-	-	indica uma label (L0)
opHLT	hlt	-	-	-	indica o fim do programa

Fonte: O autor

Primeiro foram definidas as quadruplas e sua estrutura, bem como o protótipo para a geração do código intermediário.

```
1  #ifndef _CGEN_H_
2  #define _CGEN_H_
3  #define nlabel_size 3
4  #define ntemp_size 3
5
6  typedef enum {  opADD, opSUB, opMULT, opDIV, opLT, opLEQUAL, opGT, opGREQUAL, opIGL,
       opDIF, opASSIGN, opALLOC, opIMMED, opLOAD, opSTORE,
7                  opVEC, opGOTO, opRET, opFUN, opEND, opPARAM, opCALL, opARG, opLAB, opHLT
                       } OpKind;
8
9  typedef enum {  Empty, IntConst, String } AddrKind;
10
11 //estrutura do endere amento
12 typedef struct {
13   AddrKind kind;
14   union {
15     int val;
16     struct{
17       char * name;
18       char * scope;
19     }var;
20   }contents;
21 } Address;
22
23 //estrutura das quadruplas
```

```
24 typedef struct {
25   OpKind op;
26   Address addr1, addr2, addr3;
27 } Quad;
28
29 //estrutura das listas de quadruplas
30 typedef struct QuadListRec {
31   int location;
32   Quad quad;
33   struct QuadListRec * next;
34 } * QuadList;
35
36 void codeGen(TreeNode * syntaxTree, char * codefile);
37
38 QuadList getIntermediate();
39
40 #endif
```

Também foram implementados os procedimentos para a geração do código intermediário no arquivo *cgen.c*.

```
1  //procedimento que gera o cod intermediario
2  void codeGen(TreeNode *syntaxTree, char *codefile){
3    char *s = malloc(strlen(codefile) + 7);
4    strcpy(s, "File: ");
5    strcat(s, codefile);
6    emitComment("\nC- Intermediate Code");
7    emitComment(s);
8    empty = addr_createEmpty();
9    cGen(syntaxTree);
10   quad_insert(opHLT, empty, empty, empty);
11   printCode();
12   emitComment("End of execution");
13 }
```

Aqui foi implentada a função principal do modulo de geração de código intermediario

```
1  void quad_insert(OpKind op, Address addr1, Address addr2, Address addr3){//insere na
       lista de quadriplas
2    Quad quad;
3    quad.op = op;
4    quad.addr1 = addr1;
5    quad.addr2 = addr2;
6    quad.addr3 = addr3;
7    QuadList new = (QuadList)malloc(sizeof(struct QuadListRec));
8    new->location = location;
9    new->quad = quad;
10   new->next = NULL;
11   if (head == NULL){//primeira insers o
12     head = new;
13   }
14   else{
15     QuadList q = head;
16     while (q->next != NULL){//procura local de insers o
17       q = q->next;
18     }
19     q->next = new;
20   }
21   location++;//atualiza atual localiza  o
```

```
22 }
23
24 int quad_update(int loc, Address addr1, Address addr2, Address addr3){//atualizar a
       quadrupla na localiza  o pedida
25   QuadList q = head;
26   while (q != NULL){//procura a quadrupla na localiza  o pedida
27     if (q->location == loc)
28       break;
29     q = q->next;
30   }
31   if (q == NULL)//quadrupla n o localizada
32     return 0;
33   else{//atualiza quadrupla
34     q->quad.addr1 = addr1;
35     q->quad.addr2 = addr2;
36     q->quad.addr3 = addr3;
37     return 1;
38   }
39 }
```

Nestas funções ocorrem, respectivamente, a inserção e a alteração de uma quadrupla na lista de quadruplas.

```
1  //gera c digo em um n  de instru  o
2  static void genStmt(TreeNode *tree){
3    TreeNode *p1, *p2, *p3;
4    Address addr1, addr2, addr3;
5    Address aux1, aux2, tempLabel;
6    int loc1, loc2, loc3;
7    char *label;
8    char *temp;
9
10   switch (tree->kind.stmt){
11   case IfK:
12     if (TraceCode)
13       emitComment("-> if");
14     p1 = tree->child[0]; //arg
15     p2 = tree->child[1]; //if true
16     p3 = tree->child[2]; //if false
17     // condicao if
18     cGen(p1);
19     tempLabel = labelAux;
20     // if true
21     cGen(p2);
22     //goes to end
23     loc2 = location;
24     quad_insert(opGOTO, empty, empty, empty); //jump else
25     // end if
26     label = tempLabel.contents.var.name;
27     quad_insert(opLAB,addr_createString(label, escopoAtual), empty, empty);
28     // if false comes to here
29     quad_update(loc1, addr1,addr_createString(label, escopoAtual), empty);
30     // else
31     cGen(p3);
32     if (p3 != NULL){
33       // goes to the end
34       loc3 = location;
35     }
36     label = newLabel();
```

```
37      // final
38      quad_insert(opLAB,addr_createString(label, escopoAtual), empty, empty);
39      quad_update(loc2,addr_createString(label, escopoAtual), empty, empty);
40      if (p3 != NULL)
41        quad_update(loc3,addr_createString(label, escopoAtual), empty, empty);
42      if (TraceCode)
43        emitComment("<- if");
44      break;
45
46    case WhileK:
47      if (TraceCode)
48        emitComment("-> while");
49      p1 = tree->child[0];//arg
50      p2 = tree->child[1];//body
51      // inicio do while
52      label = newLabel();
53      quad_insert(opLAB,addr_createString(label, escopoAtual), empty, empty);
54      // condicao while
55      cGen(p1);
56      tempLabel = labelAux;
57      // while
58      cGen(p2); //body
59      loc3 = location;
60      quad_insert(opGOTO,addr_createString(label, escopoAtual), empty, empty);
61      // final
62      label = tempLabel.contents.var.name;
63      quad_insert(opLAB,addr_createString(label, escopoAtual), empty, empty);
64      //if condition is false comes to here
65      quad_update(loc1, addr1,addr_createString(label, escopoAtual), empty);
66      if (TraceCode)
67        emitComment("<- while");
68      break;
69
70    case AssignK:
71      if (TraceCode)
72        emitComment("-> atrib");
73      p1 = tree->child[0];//arg
74      p2 = tree->child[1];//body
75      // var
76      cGen(p1);
77      addr1 = aux;
78      aux1 = var;
79      aux2 = offset;
80      // exp
81      cGen(p2);
82      addr2 = aux;
83      quad_insert(opASSIGN, addr1, addr2, empty);
84      quad_insert(opSTORE, aux1, aux2, addr1);
85      if (TraceCode)
86        emitComment("<- atrib");
87      break;
88
89    case ReturnINT:
90      if (TraceCode)
91        emitComment("-> returnINT");
92      p1 = tree->child[0];
93      cGen(p1);
94      addr1 = aux;
95      quad_insert(opRET, addr1, empty, empty);
96      if (TraceCode)
```

```
97        emitComment("<- returnINT");
98      break;
99    case ReturnVOID:
100     if (TraceCode)
101       emitComment("-> returnVOID");
102     addr1 = empty;
103     quad_insert(opRET, addr1, empty, empty);
104     if (TraceCode)
105       emitComment("<- returnVOID");
106     break;
107   default:
108     break;
109   }
110 }
111
112 //gera c digo em um n  de express o
113 static void genExp(TreeNode *tree){
114   TreeNode *p1, *p2, *p3;
115   Address addr1, addr2, addr3;
116   int loc1, loc2, loc3;
117   char *label;
118   char *temp;
119   char *s = "";
120
121   switch (tree->kind.exp){
122   case ConstK:
123     if (TraceCode)
124       emitComment("-> Const");
125     addr1 = addr_createIntConst(tree->attr.val);
126     temp = newTemp();
127     aux = addr_createString(temp, escopoAtual);
128     quad_insert(opIMMED, aux, addr1, empty);
129     if (TraceCode)
130       emitComment("<- Const");
131     break;
132
133   case IdK:
134     if (TraceCode)
135       emitComment("-> Id");
136     aux = addr_createString(tree->attr.name, escopoAtual);
137     p1 = tree->child[0];
138     if (p1 != NULL){
139       temp = newTemp();
140       addr1 =addr_createString(temp, escopoAtual);
141       addr2 = aux;
142       cGen(p1);
143       quad_insert(opVEC, addr1, addr2, aux);
144       var = addr2;
145       offset = aux;
146       aux = addr1;
147     }
148     else{
149       posMem = getMemLoc(tree->attr.name,escopoAtual);
150       temp = newTemp();
151       addr1 =addr_createString(temp, escopoAtual);
152       addr3 =addr_createIntConst(posMem);
153       quad_insert(opLOAD, addr1, aux, addr3);
154       var = aux;
155       offset = addr_createIntConst(posMem);;
156       aux = addr1;
```

```
157     }
158     if (TraceCode)
159       emitComment("<- Id");
160     break;
161
162   case FunDeclK:
163     strcpy(escopoAtual,tree->attr.name);
164     posMem = getMemLoc(tree->attr.name,"global");
165     if (TraceCode)
166       emitComment("-> Fun");
167     // if main
168     if (strcmp(tree->attr.name, "main") == 0)
169       mainLocation = location;
170     if ((strcmp(tree->attr.name, "input") != 0) && (strcmp(tree->attr.name, "output") !=
          0) && (strcmp(tree->attr.name, "storeStack") != 0) && (strcmp(tree->attr.name, "
          loadStack") != 0) && (strcmp(tree->attr.name, "storeRegs") != 0) && (strcmp(tree
          ->attr.name, "loadRegs") != 0)){
171       quad_insert(opFUN, addr_createString(tree->attr.name, escopoAtual),
              addr_createIntConst(posMem), empty);
172       // params
173       p1 = tree->child[1];
174       cGen(p1);
175       // dec & expressions
176       p2 = tree->child[2];
177       cGen(p2);
178       quad_insert(opEND, addr_createString(tree->attr.name, escopoAtual), empty, empty);
179       strcpy(escopoAtual, "global");
180     }
181     if (TraceCode)
182       emitComment("<- Fun");
183     break;
184
185   case AtivK:
186     if (TraceCode)
187       emitComment("-> Call");
188     nparams = tree->params;
189     p1 = tree->child[0];
190     while (p1 != NULL){
191       if(p1->kind.exp == IdK){
192         if(getVarType(p1->attr.name,escopoAtual) == VET){
193           temp = newTemp();
194           aux = addr_createString(temp,escopoAtual);
195           quad_insert(opIMMED,aux,addr_createIntConst(getMemLoc(p1->attr.name,escopoAtual
                )),empty);
196
197         }else cGen(p1);
198       }else{
199         cGen(p1);
200       }
201       quad_insert(opPARAM, aux, empty, empty);
202       nparams--;
203       p1 = p1->sibling;
204     }
205     nparams = -1;
206     aux =addr_createString("$ret", escopoAtual);
207     quad_insert(opCALL, aux, addr_createString(tree->attr.name, escopoAtual),
          addr_createIntConst(tree->params));
208
209     if (TraceCode)
210       emitComment("<- Call");
```

```
211     break;
212
213   case VarParamK:
214     posMem = getMemLoc(tree->attr.name,escopoAtual);
215     if (TraceCode)
216       emitComment("-> Param");
217     quad_insert(opARG, addr_createString(tree->attr.name, escopoAtual),
            addr_createIntConst(posMem), addr_createString(escopoAtual,escopoAtual));
218     if (TraceCode)
219       emitComment("<- Param");
220     break;
221
222   case VarDeclK:
223     posMem = getMemLoc(tree->attr.name,escopoAtual);
224     if (TraceCode)
225       emitComment("-> Var");
226     if (posMem != -1){
227       quad_insert(opALLOC, addr_createString(tree->attr.name, escopoAtual),
              addr_createIntConst(1), addr_createString(escopoAtual,escopoAtual));
228     }
229     else{
230       Error = TRUE;
231       return;
232     }
233     if (TraceCode)
234       emitComment("<- Var");
235     break;
236
237   case VetorK:
238     posMem = getMemLoc(tree->attr.name,escopoAtual);
239     if (TraceCode)
240       emitComment("-> Vet");
241     if (posMem != -1){
242       quad_insert(opALLOC, addr_createString(tree->attr.name, escopoAtual),
              addr_createIntConst(tree->child[1]->attr.val), addr_createString(escopoAtual,
              escopoAtual));
243     }
244     else{
245       Error = TRUE;
246       return;
247     }
248     if (TraceCode)
249       emitComment("<- Vet");
250     break;
251
252   case OpK:
253     if (TraceCode)
254       emitComment("-> Op");
255     p1 = tree->child[0];
256     p2 = tree->child[1];
257     cGen(p1);
258     addr1 = aux;
259     cGen(p2);
260     addr2 = aux;
261     switch (tree->attr.op){
262     case SOM:
263       temp = newTemp();
264       aux =addr_createString(temp, escopoAtual);
265       quad_insert(opADD, addr1, addr2, aux);
266       break;
```

```
267     case SUB:
268       temp = newTemp();
269       aux =addr_createString(temp, escopoAtual);
270       quad_insert(opSUB, addr1, addr2, aux);
271       break;
272     case MUL:
273       temp = newTemp();
274       aux =addr_createString(temp, escopoAtual);
275       quad_insert(opMULT, addr1, addr2, aux);
276       break;
277     case DIV:
278       temp = newTemp();
279       aux =addr_createString(temp, escopoAtual);
280       quad_insert(opDIV, addr1, addr2, aux);
281       break;
282     case MENO:
283       labelAux = addr_createString(newLabel(), escopoAtual);
284       quad_insert(opLT, addr1, addr2, labelAux);
285       break;
286     case MEIG:
287       labelAux = addr_createString(newLabel(), escopoAtual);
288       quad_insert(opLEQUAL, addr1, addr2, labelAux);
289       break;
290     case MAIO:
291       labelAux = addr_createString(newLabel(), escopoAtual);
292       quad_insert(opGT, addr1, addr2, labelAux);
293       break;
294     case MAIG:
295       labelAux = addr_createString(newLabel(), escopoAtual);
296       quad_insert(opGREQUAL, addr1, addr2, labelAux);
297       break;
298     case IGL:
299       labelAux = addr_createString(newLabel(), escopoAtual);
300       quad_insert(opIGL, addr1, addr2, labelAux);
301       break;
302     case DIF:
303       labelAux = addr_createString(newLabel(), escopoAtual);
304       quad_insert(opDIF, addr1, addr2, labelAux);
305       break;
306     default:
307       emitComment("BUG: Unknown operator");
308       break;
309     }
310     if (TraceCode)
311       emitComment("<- Op");
312     break;
313 
314   default:
315     break;
316   }
317 }
```

O gerador faz a analise dos tipos e realiza a inserção da quadruplas de acordo com o tipo de instrução.

4.3 Geração de código assembly

"Criada primeiramente em 1955 para o computador IBM650, logo após a introdução de caracteres de texto para instruções computacionais Assembly é considerada a primeira linguagem de programação, sendo uma linguagem de extremo baixo nível que permite uma comunicação mais direta com a arquitetura de cada dispositivo, podendo abranger qualquer dispositivo programável"(5).

Por ser a linguagem mais próxima da linguagem de maquina, gerar o código assembly é a ultima etapa de campilação antes da geração da linguagem executável.

```
1  void insertFun (char * id) {
2      FunList new = (FunList) malloc(sizeof(struct FunListRec));
3      new->id = (char *) malloc(strlen(id) * sizeof(char));
4      strcpy(new->id, id);
5      new->size = 0;
6      new->memloc = curmemloc;
7      new->next = NULL;
8      if (funlisthead == NULL) {
9          funlisthead = new;
10     }
11     else {
12         FunList f = funlisthead;
13         while (f->next != NULL) f = f->next;
14         f->next = new;
15     }
16     nscopes ++;
17 }
18
19 void insertVar (char * scope, char * id, int size, VarKind kind) {
20     FunList f = funlisthead;
21     if(scope == NULL){
22     if(kind == 1 )
23     scope= f->id;
24     else
25     scope= f->next->id;
26     }
27     while (f != NULL && strcmp(f->id, scope) != 0)  f = f->next;
28       if (f == NULL) {
29         insertFun(scope);
30         f = funlisthead;
31         while (f != NULL && strcmp(f->id, scope) != 0 )
32         f = f->next;
33     }
34     VarList new = (VarList) malloc(sizeof(struct VarListRec));
35     new->id = (char *) malloc(strlen(id) * sizeof(char));
36     strcpy(new->id, id);
37     new->size = size;
38     new->memloc = f->size;
39     curmemloc = curmemloc + size;
40     new->kind = kind;
41     new->next = NULL;
42     if (f->vars == NULL) {
43         f->vars = new;
44     }
45     else {
46         VarList v = f->vars;
```

```
47          while (v->next != NULL) v = v->next;
48          v->next = new;
49      }
50      f->size = f->size + size;
51  }
```

Nas funções insertFunc e insertVar são, respectivamente, inseridas na lista de funções e variáveis, as mesmas quando são lidas do código intermediário.

```
1  void insertLabel (char * label) {
2      AssemblyCode new = (AssemblyCode) malloc(sizeof(struct AssemblyCodeRec));
3      new->lineno = line;
4      new->kind = lbl;
5      new->line.label = (char *) malloc(strlen(label) * sizeof(char));
6      strcpy(new->line.label, label);
7      new->next = NULL;
8      if (codehead == NULL) {
9          codehead = new;
10     }
11     else {
12         AssemblyCode a = codehead;
13         while (a->next != NULL) a = a->next;
14         a->next = new;
15     }
16 }
```

Essa é a função responsável por inserir as 'Lambels' no código assembly.

```
1  //insere a instru  o na lista de instu  es de acordo com seu formato
2  void insertInstruction (InstrFormat format, InstrKind opcode, Reg reg1, Reg reg2, Reg
       reg3, int im, char * imlbl){
3      Instruction i;
4      i.format = format;
5      i.opcode = opcode;
6      i.reg1 = reg1;
7      i.reg2 = reg2;
8      i.reg3 = reg3;
9      i.im = im;
10     if (imlbl != NULL) {//insere lambel
11         i.imlbl = (char *) malloc(strlen(imlbl) * sizeof(char));
12         strcpy(i.imlbl, imlbl);
13     }
14     AssemblyCode new = (AssemblyCode) malloc(sizeof(struct AssemblyCodeRec));
15     new->lineno = line;
16     new->kind = instr;
17     new->line.instruction = i;
18     new->next = NULL;
19     if (codehead == NULL) {//primeira instru  o
20         codehead = new;
21     }
22     else {
23         AssemblyCode a = codehead;
24         while (a->next != NULL) a = a->next;
25         a->next = new;
26     }
27     line ++;
28 }
```

Essa é a função do *assembler* que faz a inserção das instruções no código.

```
void instructionFormatR (InstrKind opcode, Reg reg1, Reg reg2, Reg reg3) { // Tipo R
    insertInstruction(formatR, opcode, reg1, reg2, reg3, 0, NULL);
}

void instructionFormatJ (InstrKind opcode, int im, char * imlbl) { // Tipo J
    insertInstruction(formatJ, opcode, $zero, $zero, $zero, im, imlbl);
}

void instructionFormatI (InstrKind opcode, Reg reg1, Reg reg2, int im, char * imlbl) { // Tipo I
    insertInstruction(format_I, opcode, reg1, reg2, $zero, im, imlbl);
}

void instructionFormatO (InstrKind opcode, Reg reg1, int im, char * imlbl) { // Tipo O
    insertInstruction(formatO, opcode, reg1, $zero, $zero, im, imlbl);
}
```

A chamada para a função de inserção é feita de acordo com o formato da instrução.

```
Reg getParamReg () {
        return (Reg) nregtemp + 1 + curparam;
}

Reg getArgReg () {

        return (Reg) nregtemp + curarg + 1;
}

Reg getReg (char * regName) {

        for (int i = 0; i < nregisters; i ++) {
            if (strcmp(regName, regNames[i]) == 0) return (Reg) i;
        }

    return $zero;
}
```

funções que atribuem um registrador para uso no código.

```
int getLabelLine (char * label) {
    AssemblyCode a = codehead;
    while (a->next != NULL) {
        if (a->kind == lbl && strcmp(a->line.label, label) == 0) return a->lineno;
        a = a->next;
    }
    return -1;
}
```

função que retorna a linha de uma label.

```
void generateInstruction (QuadList l) {
    Quad q;
    Address a1, a2, a3;
    int aux;
    VarKind v;

    while (l != NULL) {
        q = l->quad;
        a1 = q.addr1;
        a2 = q.addr2;
```

```
11          a3 = q.addr3;
12          FunList g = funlisthead;
13          switch (q.op) {
14              case opADD:
15                  instructionFormatR(add, getReg(a3.contents.var.name), getReg(a1.contents.
                        var.name), getReg(a2.contents.var.name));
16                  break;
17
18              case opSUB:
19                  instructionFormatR(sub, getReg(a3.contents.var.name), getReg(a1.contents.
                        var.name), getReg(a2.contents.var.name));
20                  break;
21
22              case opMULT:
23                  instructionFormatR(mult, getReg(a3.contents.var.name), getReg(a1.contents
                        .var.name), getReg(a2.contents.var.name));
24                  break;
25
26              case opDIV:
27                  instructionFormatR(divi, getReg(a3.contents.var.name), getReg(a1.contents
                        .var.name), getReg(a2.contents.var.name));
28                  break;
29
30              case opLT:
31                  instructionFormatI(bgeq, getReg(a1.contents.var.name), getReg(a2.contents
                        .var.name), -1, a3.contents.var.name);
32                  break;
33
34              case opLEQUAL:
35                  instructionFormatI(bgt, getReg(a1.contents.var.name), getReg(a2.contents.
                        var.name), -1, a3.contents.var.name);
36                  break;
37
38              case opGT:
39                  instructionFormatI(bleq, getReg(a1.contents.var.name), getReg(a2.contents
                        .var.name), -1, a3.contents.var.name);
40                  break;
41
42              case opGREQUAL:
43                  instructionFormatI(blt, getReg(a1.contents.var.name), getReg(a2.contents.
                        var.name), -1, a3.contents.var.name);
44                  break;
45
46              case opIGL:
47                  instructionFormatI(bne, getReg(a1.contents.var.name), getReg(a2.contents.
                        var.name), -1, a3.contents.var.name);
48                  break;
49
50              case opDIF:
51                  instructionFormatI(beq, getReg(a1.contents.var.name), getReg(a2.contents.
                        var.name), -1, a3.contents.var.name);
52                  break;
53
54              case opASSIGN:
55                  //instructionFormatR(add, getReg(a1.contents.var.name), $zero, getReg(a2.
                        contents.var.name));
56                  instructionFormatI(mov, getReg(a1.contents.var.name), getReg(a2.contents.
                        var.name), 0, NULL);
57                  break;
58
```

```
59              case opALLOC:
60                  if (a2.contents.val == 1)
61                   insertVar(a3.contents.var.name, a1.contents.var.name,  a2.contents.val,
                         simple);
62                  else insertVar(a3.contents.var.name, a1.contents.var.name,  a2.contents.
                        val, vector);
63                  break;
64
65              case opIMMED:
66                  //instructionFormatI(addi,getReg(a1.contents.var.name),$zero, a2.contents
                        .val, NULL);
67                   instructionFormatI(movi, getReg(a1.contents.var.name), $zero, a2.
                         contents.val, NULL);
68                  break;
69
70              case opLOAD:
71                  aux = getMemLoc(a2.contents.var.name, a2.contents.var.scope);
72                  if (aux == -1){
73                          aux = getMemLoc(a2.contents.var.name, "global");
74                          instructionFormatI(lw, getReg(a1.contents.var.name), $bp, aux,
                                NULL);
75                  }
76                  else{
77                          instructionFormatI(lw, getReg(a1.contents.var.name), $sp, aux,
                                NULL);
78                  }
79                  break;
80
81              case opSTORE:
82                  aux = getMemLoc(a1.contents.var.name, a1.contents.var.scope);
83                  if (aux == -1) {
84                          aux = getMemLoc(a1.contents.var.name, "global");
85                          v = getVarKind(a1.contents.var.name, "global");
86                          if(v == vector)
87                              instructionFormatI(sw, getReg(a3.contents.var.name), getReg(
                                    a2.contents.var.name), aux, NULL);//inv
88                          else
89                              instructionFormatI(sw, getReg(a3.contents.var.name), $bp, aux
                                    , NULL);
90
91                  }
92                  else{
93
94                          v = getVarKind(a1.contents.var.name, a1.contents.var.scope);
95                          if(v == vector)
96                          instructionFormatI(sw, getReg(a3.contents.var.name),getReg(a2.
                                contents.var.name), aux, NULL);
97                              //instructionFormatI(sw, getReg(a2.contents.var.name), getReg
                                    (a3.contents.var.name), aux, NULL);
98                          else
99                              instructionFormatI(sw, getReg(a3.contents.var.name), $sp, aux
                                    , NULL);
100
101                 }
102                 break;
103
104             case opVEC:
105                 aux = getMemLoc(a2.contents.var.name, a2.contents.var.scope);
106                 if (aux == -1)
107                     v = getVarKind(a2.contents.var.name, "global");
```

```
108                 else
109                     v = getVarKind(a2.contents.var.name, a2.contents.var.scope);
110                 if (v == vector) {
111                     if (aux == -1) { // caso o vetor seja global
112 
113                             aux = getMemLoc(a2.contents.var.name, "global");
114                             //instructionFormatR(add, $bp, getReg(a3.contents.var.name),
                                    getReg(a3.contents.var.name));
115                             instructionFormatR(add,getReg(a3.contents.var.name), $bp,
                                    getReg(a3.contents.var.name));
116                     }
117                     else{ // caso seja um vetor local
118 
119                             //instructionFormatR(add, $sp, getReg(a3.contents.var.name),
                                    getReg(a3.contents.var.name));
120                             instructionFormatR(add,getReg(a3.contents.var.name), $sp,
                                    getReg(a3.contents.var.name));
121                     }
122                     instructionFormatI(lw, getReg(a3.contents.var.name), getReg(a1.
                            contents.var.name),aux, NULL);
123                 }
124                 else{
125                         instructionFormatI(lw,$sp,getReg(a1.contents.var.name),getMemLoc(
                                a2.contents.var.name,a2.contents.var.scope),NULL);
126                         instructionFormatR(add, getReg(a3.contents.var.name), getReg(a1.
                                contents.var.name),getReg(a3.contents.var.name));
127                         instructionFormatI(lw,getReg(a3.contents.var.name),getReg(a1.
                                contents.var.name),0,NULL);
128                 }
129                 break;
130 
131             case opGOTO:
132                 instructionFormatJ(j, -1, a1.contents.var.name);
133                 break;
134 
135             case opRET:
136                 if (a1.kind == String) instructionFormatR(add, $ret, $zero, getReg(a1.
                        contents.var.name));
137                 else
138                     //instructionFormatR(add, $ret, $zero, a1.contents.val);
139                     instructionFormatI(mov, $ret, a1.contents.val, 0, NULL);
140                 if(strcmp(a1.contents.var.scope, "main") != 0){
141                     aux = getFunSize(a1.contents.var.scope);
142                     instructionFormatJ(jra, -1, NULL);
143                 }
144                 else
145                     instructionFormatJ(j, -1, "end");
146                 break;
147 
148             case opFUN:
149                 if (jmpmain == 0) {
150                     instructionFormatJ(j, -1, "main");
151                     jmpmain = 1;
152                 }
153                /* if(strcmp(a1.contents.var.scope, "main") != 0){
154                     instructionFormatI(movi, $bp, $zero, curmemloc, NULL);
155                     instructionFormatI(mov, $sp, $bp, curmemloc, NULL);
156                 }*/
157 
158                 insertLabel(a1.contents.var.name);
```

```
159                 insertFun(a1.contents.var.name);
160                 curarg = 0;
161                 break;
162 
163             case opEND:
164                 if (strcmp(a1.contents.var.name, "main") == 0)
165                     instructionFormatJ(j, -1, "end");
166                 else{
167                     aux = getFunSize(a1.contents.var.name);
168                     instructionFormatJ(jra, 0, NULL);
169                 }
170                 break;
171 
172             case opPARAM:
173                 //instructionFormatR(add, getParamReg(), $zero, getReg(a1.contents.var.
                        name));
174                 instructionFormatI(mov, getParamReg(), getReg(a1.contents.var.name), 0,
                        NULL);
175                 curparam = (curparam+1)%nregparam;
176                 break;
177 
178             case opCALL://chada de fun   o
179                 if (strcmp(a2.contents.var.name, "input") == 0) {//entrada de dados
180                     instructionFormatO(in, getReg(a1.contents.var.name), 0, NULL);
181                 }
182                 else if (strcmp(a2.contents.var.name, "output") == 0) {//saida de dados
183                     instructionFormatO(out, getArgReg(), 0, NULL);
184                     instructionFormatR(pause, $zero, $zero, $zero);
185                 }
186                 else{//outros tipos de fun   o
187                         aux = getFunSize(a1.contents.var.scope);
188                        /* if(strcmp(a1.contents.var.scope, "main") == 0){
189                             instructionFormatI(movi, $bp, $zero, curmemloc, NULL);
190                             instructionFormatI(mov, $bp, $sp, curmemloc, NULL);
191                         } */
192                         instructionFormatI(addi,$sp,$sp,aux,NULL);
193                         instructionFormatJ(jal, -1, a2.contents.var.name);
194                         instructionFormatI(subi,$sp, $sp, aux, NULL);
195                 }
196                 narg = a3.contents.val;
197                 curparam = 0;
198                 break;
199 
200             case opARG:
201                 insertVar(a3.contents.var.name, a1.contents.var.name, 1, checkType(l));
202                 FunList f = funlisthead;
203                     instructionFormatI(sw, getArgReg(), $sp,getMemLoc(a1.contents.var.
                            name,a1.contents.var.scope), NULL);
204 
205                 curarg ++;
206                 break;
207 
208             case opLAB:
209                 insertLabel(a1.contents.var.name);
210                 break;
211 
212             case opHLT:
213                 insertLabel("end");
214                 instructionFormatR(end, $zero, $zero, $zero);
215                 break;
```

```
216
217             default:
218                 break;
219         }
220         l = l->next;
221     }
222 }
```

Função que faz a chamada de função para a inserção da instrução de acordo com seu formato.

```
1  void printAssembly () {
2      AssemblyCode a = codehead;
3      printf("\nC digo Assembly C-\n");
4      while (a != NULL) {
5          if (a->kind == instr) {
6              if (a->line.instruction.format == formatR) {
7                  if(a->line.instruction.opcode == end)
8                      printf("%d:\t%s\n",    a->lineno, InstrNames[a->line.instruction.
                           opcode]);
9                  else
10                     printf("%d:\t%s %s, %s, %s\n",    a->lineno, InstrNames[a->line.
                           instruction.opcode], regNames[a->line.instruction.reg1],
11                                                 regNames[a->line.instruction.reg2],
                                                       regNames[a->line.instruction.reg3]);
12             }
13             else if (a->line.instruction.format == format_I) {
14                     printf("%d:\t%s %s, %s, %d\n",    a->lineno, InstrNames[a->line.
                           instruction.opcode], regNames[a->line.instruction.reg1],
15                                                 regNames[a->line.instruction.reg2], a->
                                                       line.instruction.im);
16
17             }
18             else if (a->line.instruction.format == formatO) {
19                 printf("%d:\t%s %s\n",    a->lineno, InstrNames[a->line.instruction.
                       opcode], regNames[a->line.instruction.reg1]);
20             }
21             else {
22                 if (a->line.instruction.opcode == jra)
23                     printf("%d:\t%s\n",    a->lineno, InstrNames[a->line.instruction.
                           opcode]);
24                 else
25                     printf("%d:\t%s %d\n",    a->lineno, InstrNames[a->line.instruction.
                           opcode], a->line.instruction.im);
26             }
27         }
28         else
29             printf(".%s\n", a->line.label);
30         a = a->next;
31     }
32 }
```

Função que 'imprime' na tela o código assembly gerado.

4.4 Geração de código binário

Para o algoritmo possa ser executado no processador alvo, é necessário que as instruções estejam no formato de numero binário de 32 bits cada instrução. Desse modo, o gerador de código binário faz a tradução do código assembly para o código binário executável no processador.

Primeiramente foi definido os equivalentes binários dos endereços de registradores e os identificadores de instruções (opcode), implementado no arquivo *binary.h*.

```
1  //defini  es de opcode em binario
2  #define OP_ADD   "6'b000000"
3  #define OP_ADDI  "6'b000001"
4  #define OP_SUB   "6'b000010"
5  #define OP_SUBI  "6'b000011"
6  #define OP_MULT  "6'b000100"
7  #define OP_MULTI "6'b000101"
8  #define OP_DIV   "6'b000110"
9  #define OP_DIVI  "6'b000111"
10 #define OP_RDIV  "6'b001000"
11 #define OP_OR    "6'b001001"
12 #define OP_AND   "6'b001010"
13 #define OP_NOT   "6'b001011"
14 #define OP_XOR   "6'b001100"
15 #define OP_NOR   "6'b001101"
16 #define OP_NAND  "6'b001110"
17 #define OP_XNOR  "6'b001111"
18 #define OP_LT    "6'b010000"
19 #define OP_J     "6'b010001"
20 #define OP_JR    "6'b010010"
21 #define OP_JAL   "6'b010011"
22 #define OP_BEQ   "6'b010100"
23 #define OP_BNE   "6'b010101"
24 #define OP_BLT   "6'b010110"
25 #define OP_LW    "6'b010111"
26 #define OP_SW    "6'b011000"
27 #define OP_MOV   "6'b011001"
28 #define OP_MOVI  "6'b011010"
29 #define OP_MFHI  "6'b011011"
30 #define OP_MFLO  "6'b011100"
31 #define OP_IN    "6'b011101"
32 #define OP_OUT   "6'b011110"
33 #define OP_END   "6'b011111"
34 #define OP_PAUSE "6'b100000"
35
36
37 //defini  es de registradores
38 #define RZERO   "5'b00000"
39 #define RT1     "5'b00001"
40 #define RT2     "5'b00010"
41 #define RT3     "5'b00011"
42 #define RT4     "5'b00100"
43 #define RT5     "5'b00101"
44 #define RT6     "5'b00110"
45 #define RT7     "5'b00111"
46 #define RT8     "5'b01000"
47 #define RT9     "5'b01001"
48 #define RT10    "5'b01010"
```

```
49 #define RT11    "5'b01011"
50 #define RT12    "5'b01100"
51 #define RP1     "5'b01101"
52 #define RP2     "5'b01110"
53 #define RP3     "5'b01111"
54 #define RP4     "5'b10001"
55 #define RP5     "5'b10010"
56 #define RP6     "5'b10011"
57 #define RS1     "5'b10100"
58 #define RS2     "5'b10101"
59 #define RS3     "5'b10110"
60 #define RS4     "5'b10111"
61 #define RS5     "5'b11000"
62 #define RS6     "5'b11001"
63 #define RSP     "5'b11010"
64 #define RBP     "5'b11011"
65 #define RA      "5'b11100"
66 #define RRET    "5'b11111"
67
68 //prototipo gera   o binario
69 void generateBinary (AssemblyCode a);
```

Em seguida foi desenvolvido o gerador propriamente dito, na qual para cada elemento de um comando em assembly, seu equivalente binário foi transcrito, de acordo com o formato da instrução.

```
1  #include "globals.h"
2  #include "symtab.h"
3  #include "cgen.h"
4  #include "assembly.h"
5  #include "binary.h"
6
7  char *get_register(Reg r){
8
9     switch (r)
10    {
11     case $zero:
12      return RZERO;
13     break;
14
15    case $t1:
16      return RT1;
17     break;
18
19    case $t2:
20      return RT2;
21     break;
22
23    case $t3:
24      return RT3;
25     break;
26
27    case $t4:
28      return RT4;
29     break;
30
31     case $t5:
32      return RT5;
33     break;
```

```
34
35    case $t6:
36      return RT6;
37     break;
38
39    case $t7:
40      return RT7;
41     break;
42
43    case $t8:
44      return RT8;
45     break;
46
47    case $t9:
48      return RT9;
49     break;
50
51    case $t10:
52      return RT10;
53     break;
54
55    case $t11:
56      return RT11;
57     break;
58
59    case $t12:
60      return RT12;
61     break;
62
63    case $p1:
64      return RP1;
65     break;
66
67    case $p2:
68      return RP2;
69     break;
70
71    case $p3:
72      return RP3;
73     break;
74
75    case $p4:
76      return RP4;
77     break;
78
79    case $p5:
80      return RP5;
81     break;
82
83    case $p6:
84      return RP6;
85     break;
86
87    case $s1:
88      return RS1;
89     break;
90
91    case $s2:
92      return RS2;
93     break;
```

```
94
95      case $s3:
96        return RS3;
97       break;
98
99      case $s4:
100       return RS4;
101      break;
102
103     case $s5:
104       return RS5;
105      break;
106
107     case $s6:
108       return RS6;
109      break;
110
111     case $sp:
112       return RSP;
113      break;
114
115     case $bp:
116       return RBP;
117      break;
118
119     case $ra:
120       return RA;
121     break;
122
123     case $ret:
124       return RRET;
125     break;
126
127     default:
128      break;
129     }
130 }
131
132 char *get_op(InstrKind tipo){
133
134     switch (tipo)
135     {
136      case add:
137        return OP_ADD;
138      break;
139
140      case sub:
141        return OP_SUB;
142      break;
143
144      case mult:
145        return OP_MULT;
146      break;
147
148      case divi:
149        return OP_DIV;
150      break;
151
152      case or:
153        return OP_OR;
```

```
154     break;
155
156     case xor:
157       return OP_XOR;
158     break;
159
160
161     case lw:
162       return OP_LW;
163     break;
164
165     case sw:
166       return OP_SW;
167     break;
168
169     case in:
170       return OP_IN;
171     break;
172
173     case out:
174       return OP_OUT;
175     break;
176
177     case addi:
178       return OP_ADDI;
179     break;
180
181     case subi:
182       return OP_SUBI;
183     break;
184
185     case multi:
186       return OP_MULTI;
187     break;
188
189     case divim:
190       return OP_DIVI;
191     break;
192
193     case andi:
194       return OP_ANDI;
195     break;
196
197     case ori:
198       return OP_ORI;
199     break;
200
201     case beq:
202       return OP_BEQ;
203     break;
204
205     case bne:
206       return OP_BNE;
207     break;
208
209     case blt:
210       return OP_BLT;
211     break;
212
213     case bgt:
```

```
214       return OP_BGT;
215     break;
216
217     case bleq:
218       return OP_BLEQ;
219     break;
220
221     case bgeq:
222       return OP_BGEQ;
223     break;
224
225     case j:
226       return OP_J;
227     break;
228
229     case jra:
230       return OP_JR;
231     break;
232
233     case jr:
234       return OP_JR;
235     break;
236
237     case mov:
238       return OP_MOV;
239     break;
240
241     case movi:
242       return OP_MOVI;
243     break;
244
245     case end:
246       return OP_END;
247     break;
248
249     case pause:
250       return OP_PAUSE;
251     break;
252
253     default:
254     break;
255    }
256
257 }
258
259 char * assumbly_para_binario (Instruction i) {
260
261     char * bin = (char *) malloc(200 * sizeof(char));
262
263
264     if (i.format == formatR) {
265             sprintf(bin, "%s,%s,%s,%s,%s", get_op(i.opcode), get_register(i.reg1),
                    get_register(i.reg2), get_register(i.reg3), "11'd0");
266     }
267     else if (i.format == format_I) {
268         if(i.opcode == lw)
269             sprintf(bin, "%s,%s,%s,16'd%i", get_op(i.opcode), get_register(i.reg1),
                    get_register(i.reg2),i.im);
270         else if(i.opcode == sw)
271             sprintf(bin, "%s,%s,%s,%s,11'd%i", get_op(i.opcode),RZERO, get_register(i.
```

```
                reg2), get_register(i.reg1), i.im);
272         else
273             sprintf(bin, "%s,%s,%s,16'd%i", get_op(i.opcode), get_register(i.reg1),
                    get_register(i.reg2), i.im);
274     }
275     else if (i.format == format0) {
276         if (i.opcode == in)
277             sprintf(bin, "%s,%s,%s", OP_IN, get_register(i.reg1),"21'd0");
278         if (i.opcode == out)
279             sprintf(bin, "%s,%s,%s,%s", OP_OUT, RZERO, get_register(i.reg1), "16'd0");
280     }
281     else {
282         if (i.opcode == jra )
283             sprintf(bin, "%s,%s,%s",OP_JR,RA,"21'd0");
284         else if (i.opcode == jr )
285             sprintf(bin, "%s,%s,%s", OP_JR ,get_register(i.reg1),"21'd0");
286         else if (i.opcode == jal )
287             sprintf(bin, "%s,%s,21'd%i", OP_JAL, RA, i.im);
288         else
289             sprintf(bin, "%s,26'd%i", get_op(i.opcode),i.im);
290 
291     }
292     return bin;
293 }
294 
295 void generateBinary (AssemblyCode head) {
296     AssemblyCode a = head;
297     FILE * c = code;
298     char * bin;
299 
300     printf("\nC- Binary Code\n");
301 
302     while (a != NULL) {
303         if (a->kind == instr) {
304 
305             fprintf(c, "memoria[32'd%d] ={", a->lineno);
306             printf("memoria[32'd%d] =" , a->lineno);
307             bin = assumbly_para_binario(a->line.instruction);
308             fprintf(c, "%s};\n",bin);
309             printf("%s;\n", bin);
310         }
311         else {
312             fprintf(c, "//%s\n", a->line.label);
313             printf("//.%s\n", a->line.label);
314         }
315         a = a->next;
316     }
317 }
```

5 Exemplos

5.1 Sort

5.1.1 Código Fonte

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

5.1.2 Código Intermediário

```
(alloc, vet, 10, global )
(fun, minloc, 0, - )
(arg, low, 2, minloc )
(arg, high, 3, minloc )
(alloc, i, 1, minloc )
(alloc, x, 1, minloc )
(alloc, k, 1, minloc )
(load, $t1, k, 6 )
(load, $t2, low, 2 )
(atrib, $t1, $t2, - )
(store, k, 6, $t1 )
(load, $t3, x, 5 )
(load, $t5, low, 2 )
(vec, $t4, a, $t5 )
(atrib, $t3, $t4, - )
(store, x, 5, $t3 )
(load, $t6, i, 4 )
(load, $t7, low, 2 )
(immed, $t8, 1, - )
(add, $t7, $t8, $t9 )
(atrib, $t6, $t9, - )
(store, i, 4, $t6 )
(lab, L0, -, - )
(load, $t10, i, 4 )
(load, $t11, high, 3 )
(bgeq, $t10, $t11, L1 )
(load, $t1, i, 4 )
(vec, $t12, a, $t1 )
(load, $t2, x, 5 )
(bgeq, $t12, $t2, L2 )
(load, $t3, x, 5 )
(load, $t5, i, 4 )
(vec, $t4, a, $t5 )
(atrib, $t3, $t4, - )
(store, x, 5, $t3 )
(load, $t6, k, 6 )
(load, $t7, i, 4 )
(atrib, $t6, $t7, - )
(store, k, 6, $t6 )
(goto, L3, -, - )
(lab, L2, -, - )
(lab, L3, -, - )
(load, $t8, i, 4 )
(load, $t9, i, 4 )
(immed, $t10, 1, - )
(add, $t9, $t10, $t11 )
(atrib, $t8, $t11, - )
(store, i, 4, $t8 )
(goto, L0, -, - )
(lab, L1, -, - )
(load, $t12, k, 6 )
(ret, $t12, -, - )
(end, minloc, -, - )
(fun, sort, 0, - )
(arg, low, 2, sort )
(arg, high, 3, sort )
(alloc, i, 1, sort )
(alloc, k, 1, sort )
```

```
(load, $t1, i, 4 )
(load, $t2, low, 2 )
(atrib, $t1, $t2, - )
(store, i, 4, $t1 )
(lab, L4, -, - )
(load, $t3, i, 4 )
(load, $t4, high, 3 )
(immed, $t5, 1, - )
(sub, $t4, $t5, $t6 )
(bgeq, $t3, $t6, L5 )
(alloc, t, 1, sort )
(load, $t7, k, 5 )
(immed, $t8, 1, - )
(param, $t8, -, - )
(load, $t9, i, 4 )
(param, $t9, -, - )
(load, $t10, high, 3 )
(param, $t10, -, - )
(call, $ret, minloc, 3 )
(atrib, $t7, $ret, - )
(store, k, 5, $t7 )
(load, $t11, t, 6 )
(load, $t1, k, 5 )
(vec, $t12, a, $t1 )
(atrib, $t11, $t12, - )
(store, t, 6, $t11 )
(load, $t3, k, 5 )
(vec, $t2, a, $t3 )
(load, $t5, i, 4 )
(vec, $t4, a, $t5 )
(atrib, $t2, $t4, - )
(store, a, $t3, $t2 )
(load, $t7, i, 4 )
(vec, $t6, a, $t7 )
(load, $t8, t, 6 )
(atrib, $t6, $t8, - )
(store, a, $t7, $t6 )
(load, $t9, i, 4 )
(load, $t10, i, 4 )
(immed, $t11, 1, - )
(add, $t10, $t11, $t12 )
(atrib, $t9, $t12, - )
(store, i, 4, $t9 )
(goto, L4, -, - )
(lab, L5, -, - )
(end, sort, -, - )
(fun, main, 0, - )
(alloc, i, 1, main )
(load, $t1, i, 1 )
(immed, $t2, 0, - )
(atrib, $t1, $t2, - )
(store, i, 1, $t1 )
(lab, L6, -, - )
(load, $t3, i, 1 )
(immed, $t4, 10, - )
(bgeq, $t3, $t4, L7 )
(load, $t6, i, 1 )
(vec, $t5, vet, $t6 )
(call, $ret, input, 0 )
(atrib, $t5, $ret, - )
```

```
(store, vet, $t6, $t5 )
(load, $t7, i, 1 )
(load, $t8, i, 1 )
(immed, $t9, 1, - )
(add, $t8, $t9, $t10 )
(atrib, $t7, $t10, - )
(store, i, 1, $t7 )
(goto, L6, -, - )
(lab, L7, -, - )
(load, $t11, vet, -1 )
(param, $t11, -, - )
(immed, $t12, 0, - )
(param, $t12, -, - )
(immed, $t1, 10, - )
(param, $t1, -, - )
(call, $ret, sort, 3 )
(load, $t2, i, 1 )
(immed, $t3, 0, - )
(atrib, $t2, $t3, - )
(store, i, 1, $t2 )
(lab, L8, -, - )
(load, $t4, i, 1 )
(immed, $t5, 10, - )
(bgeq, $t4, $t5, L9 )
(load, $t7, i, 1 )
(vec, $t6, vet, $t7 )
(param, $t6, -, - )
(call, $ret, output, 1 )
(load, $t8, i, 1 )
(load, $t9, i, 1 )
(immed, $t10, 1, - )
(add, $t9, $t10, $t11 )
(atrib, $t8, $t11, - )
(store, i, 1, $t8 )
(goto, L8, -, - )
(lab, L9, -, - )
(end, main, -, - )
(end, -, -, - )
```

5.1.3 Código Assembly

```
0:      j 107
.minloc
1:      sw $p1, $sp, 2
2:      sw $p2, $sp, 3
3:      lw $t1, $sp, 6
4:      lw $t2, $sp, 2
5:      mov $t1, $t2, 0
6:      sw $t1, $sp, 6
7:      lw $t3, $sp, 5
8:      lw $t5, $sp, 2
9:      lw $sp, $t4, 1
10:     add $t5, $t4, $t5
11:     lw $t5, $t4, 0
12:     mov $t3, $t4, 0
13:     sw $t3, $sp, 5
14:     lw $t6, $sp, 4
15:     lw $t7, $sp, 2
```

```
16:     movi $t8, $zero, 1
17:     add $t9, $t7, $t8
18:     mov $t6, $t9, 0
19:     sw $t6, $sp, 4
.L0
20:     lw $t10, $sp, 4
21:     lw $t11, $sp, 3
22:     bgeq $t10, $t11, 48
23:     lw $t1, $sp, 4
24:     lw $sp, $t12, 1
25:     add $t1, $t12, $t1
26:     lw $t1, $t12, 0
27:     lw $t2, $sp, 5
28:     bgeq $t12, $t2, 41
29:     lw $t3, $sp, 5
30:     lw $t5, $sp, 4
31:     lw $sp, $t4, 1
32:     add $t5, $t4, $t5
33:     lw $t5, $t4, 0
34:     mov $t3, $t4, 0
35:     sw $t3, $sp, 5
36:     lw $t6, $sp, 6
37:     lw $t7, $sp, 4
38:     mov $t6, $t7, 0
39:     sw $t6, $sp, 6
40:     j 41
.L2
.L3
41:     lw $t8, $sp, 4
42:     lw $t9, $sp, 4
43:     movi $t10, $zero, 1
44:     add $t11, $t9, $t10
45:     mov $t8, $t11, 0
46:     sw $t8, $sp, 4
47:     j 20
.L1
48:     lw $t12, $sp, 6
49:     add $ret, $zero, $t12
50:     jra
51:     jra
.sort
52:     sw $p1, $sp, 2
53:     sw $p2, $sp, 3
54:     lw $t1, $sp, 4
55:     lw $t2, $sp, 2
56:     mov $t1, $t2, 0
57:     sw $t1, $sp, 4
.L4
58:     lw $t3, $sp, 4
59:     lw $t4, $sp, 3
60:     movi $t5, $zero, 1
61:     sub $t6, $t4, $t5
62:     bgeq $t3, $t6, 106
63:     lw $t7, $sp, 5
64:     movi $t8, $zero, 1
65:     mov $p1, $t8, 0
66:     lw $t9, $sp, 4
67:     mov $p2, $t9, 0
68:     lw $t10, $sp, 3
69:     mov $p3, $t10, 0
```

```
70:     addi $sp, $sp, 5
71:     jal 1
72:     subi $sp, $sp, 5
73:     mov $t7, $ret, 0
74:     sw $t7, $sp, 5
75:     lw $t11, $sp, 6
76:     lw $t1, $sp, 5
77:     lw $sp, $t12, 1
78:     add $t1, $t12, $t1
79:     lw $t1, $t12, 0
80:     mov $t11, $t12, 0
81:     sw $t11, $sp, 6
82:     lw $t3, $sp, 5
83:     lw $sp, $t2, 1
84:     add $t3, $t2, $t3
85:     lw $t3, $t2, 0
86:     lw $t5, $sp, 4
87:     lw $sp, $t4, 1
88:     add $t5, $t4, $t5
89:     lw $t5, $t4, 0
90:     mov $t2, $t4, 0
91:     sw $t2, $sp, 1
92:     lw $t7, $sp, 4
93:     lw $sp, $t6, 1
94:     add $t7, $t6, $t7
95:     lw $t7, $t6, 0
96:     lw $t8, $sp, 6
97:     mov $t6, $t8, 0
98:     sw $t6, $sp, 1
99:     lw $t9, $sp, 4
100:    lw $t10, $sp, 4
101:    movi $t11, $zero, 1
102:    add $t12, $t10, $t11
103:    mov $t9, $t12, 0
104:    sw $t9, $sp, 4
105:    j 58
.L5
106:    jra
.main
107:    lw $t1, $sp, 1
108:    movi $t2, $zero, 0
109:    mov $t1, $t2, 0
110:    sw $t1, $sp, 1
.L6
111:    lw $t3, $sp, 1
112:    movi $t4, $zero, 3
113:    bgeq $t3, $t4, 127
114:    lw $t6, $sp, 1
115:    add $t6, $bp, $t6
116:    lw $t6, $t5, 0
117:    in $ret
118:    mov $t5, $ret, 0
119:    sw $t5, $t6, 0
120:    lw $t7, $sp, 1
121:    lw $t8, $sp, 1
122:    movi $t9, $zero, 1
123:    add $t10, $t8, $t9
124:    mov $t7, $t10, 0
125:    sw $t7, $sp, 1
126:    j 111
```

```
.L7
127:    lw $t11, $bp, 0
128:    mov $p1, $t11, 0
129:    movi $t12, $zero, 0
130:    mov $p2, $t12, 0
131:    movi $t1, $zero, 3
132:    mov $p3, $t1, 0
133:    addi $sp, $sp, 1
134:    jal 52
135:    subi $sp, $sp, 1
136:    lw $t2, $sp, 1
137:    movi $t3, $zero, 0
138:    mov $t2, $t3, 0
139:    sw $t2, $sp, 1
.L8
140:    lw $t4, $sp, 1
141:    movi $t5, $zero, 3
142:    bgeq $t4, $t5, 156
143:    lw $t7, $sp, 1
144:    add $t7, $bp, $t7
145:    lw $t7, $t6, 0
146:    mov $p1, $t6, 0
147:    out $p1
148:    pause $zero, $zero, $zero
149:    lw $t8, $sp, 1
150:    lw $t9, $sp, 1
151:    movi $t10, $zero, 1
152:    add $t11, $t9, $t10
153:    mov $t8, $t11, 0
154:    sw $t8, $sp, 1
155:    j 140
.L9
156:    j 157
.end
157:    end
```

5.1.4 Código executável

```
memoria[32'd0] =6'b010001,26'd107;
//.minloc
memoria[32'd1] =6'b011000,5'b00000,5'b11010,5'b01101,11'd2;
memoria[32'd2] =6'b011000,5'b00000,5'b11010,5'b01110,11'd3;
memoria[32'd3] =6'b010111,5'b00001,5'b11010,16'd6;
memoria[32'd4] =6'b010111,5'b00010,5'b11010,16'd2;
memoria[32'd5] =6'b011001,5'b00001,5'b00010,16'd0;
memoria[32'd6] =6'b011000,5'b00000,5'b11010,5'b00001,11'd6;
memoria[32'd7] =6'b010111,5'b00011,5'b11010,16'd5;
memoria[32'd8] =6'b010111,5'b00101,5'b11010,16'd2;
memoria[32'd9] =6'b010111,5'b11010,5'b00100,16'd1;
memoria[32'd10] =6'b000000,5'b00101,5'b00100,5'b00101,11'd0;
memoria[32'd11] =6'b010111,5'b00101,5'b00100,16'd0;
memoria[32'd12] =6'b011001,5'b00011,5'b00100,16'd0;
memoria[32'd13] =6'b011000,5'b00000,5'b11010,5'b00011,11'd5;
memoria[32'd14] =6'b010111,5'b00110,5'b11010,16'd4;
memoria[32'd15] =6'b010111,5'b00111,5'b11010,16'd2;
memoria[32'd16] =6'b011010,5'b01000,5'b00000,16'd1;
memoria[32'd17] =6'b000000,5'b01001,5'b00111,5'b01000,11'd0;
memoria[32'd18] =6'b011001,5'b00110,5'b01001,16'd0;
```

```
21 memoria[32'd19] =6'b011000,5'b00000,5'b11010,5'b00110,11'd4;
22 //.L0
23 memoria[32'd20] =6'b010111,5'b01010,5'b11010,16'd4;
24 memoria[32'd21] =6'b010111,5'b01011,5'b11010,16'd3;
25 memoria[32'd22] =6'b100010,5'b01010,5'b01011,16'd48;
26 memoria[32'd23] =6'b010111,5'b00001,5'b11010,16'd4;
27 memoria[32'd24] =6'b010111,5'b11010,5'b01100,16'd1;
28 memoria[32'd25] =6'b000000,5'b00001,5'b01100,5'b00001,11'd0;
29 memoria[32'd26] =6'b010111,5'b00001,5'b01100,16'd0;
30 memoria[32'd27] =6'b010111,5'b00010,5'b11010,16'd5;
31 memoria[32'd28] =6'b100010,5'b01100,5'b00010,16'd41;
32 memoria[32'd29] =6'b010111,5'b00011,5'b11010,16'd5;
33 memoria[32'd30] =6'b010111,5'b00101,5'b11010,16'd4;
34 memoria[32'd31] =6'b010111,5'b11010,5'b00100,16'd1;
35 memoria[32'd32] =6'b000000,5'b00101,5'b00100,5'b00101,11'd0;
36 memoria[32'd33] =6'b010111,5'b00101,5'b00100,16'd0;
37 memoria[32'd34] =6'b011001,5'b00011,5'b00100,16'd0;
38 memoria[32'd35] =6'b011000,5'b00000,5'b11010,5'b00011,11'd5;
39 memoria[32'd36] =6'b010111,5'b00110,5'b11010,16'd6;
40 memoria[32'd37] =6'b010111,5'b00111,5'b11010,16'd4;
41 memoria[32'd38] =6'b011001,5'b00110,5'b00111,16'd0;
42 memoria[32'd39] =6'b011000,5'b00000,5'b11010,5'b00110,11'd6;
43 memoria[32'd40] =6'b010001,26'd41;
44 //.L2
45 //.L3
46 memoria[32'd41] =6'b010111,5'b01000,5'b11010,16'd4;
47 memoria[32'd42] =6'b010111,5'b01001,5'b11010,16'd4;
48 memoria[32'd43] =6'b011010,5'b01010,5'b00000,16'd1;
49 memoria[32'd44] =6'b000000,5'b01011,5'b01001,5'b01010,11'd0;
50 memoria[32'd45] =6'b011001,5'b01000,5'b01011,16'd0;
51 memoria[32'd46] =6'b011000,5'b00000,5'b11010,5'b01000,11'd4;
52 memoria[32'd47] =6'b010001,26'd20;
53 //.L1
54 memoria[32'd48] =6'b010111,5'b01100,5'b11010,16'd6;
55 memoria[32'd49] =6'b000000,5'b11111,5'b00000,5'b01100,11'd0;
56 memoria[32'd50] =6'b010010,5'b11100,21'd0;
57 memoria[32'd51] =6'b010010,5'b11100,21'd0;
58 //.sort
59 memoria[32'd52] =6'b011000,5'b00000,5'b11010,5'b01101,11'd2;
60 memoria[32'd53] =6'b011000,5'b00000,5'b11010,5'b01110,11'd3;
61 memoria[32'd54] =6'b010111,5'b00001,5'b11010,16'd4;
62 memoria[32'd55] =6'b010111,5'b00010,5'b11010,16'd2;
63 memoria[32'd56] =6'b011001,5'b00001,5'b00010,16'd0;
64 memoria[32'd57] =6'b011000,5'b00000,5'b11010,5'b00001,11'd4;
65 //.L4
66 memoria[32'd58] =6'b010111,5'b00011,5'b11010,16'd4;
67 memoria[32'd59] =6'b010111,5'b00100,5'b11010,16'd3;
68 memoria[32'd60] =6'b011010,5'b00101,5'b00000,16'd1;
69 memoria[32'd61] =6'b000010,5'b00110,5'b00100,5'b00101,11'd0;
70 memoria[32'd62] =6'b100010,5'b00011,5'b00110,16'd106;
71 memoria[32'd63] =6'b010111,5'b00111,5'b11010,16'd5;
72 memoria[32'd64] =6'b011010,5'b01000,5'b00000,16'd1;
73 memoria[32'd65] =6'b011001,5'b01101,5'b01000,16'd0;
74 memoria[32'd66] =6'b010111,5'b01001,5'b11010,16'd4;
75 memoria[32'd67] =6'b011001,5'b01110,5'b01001,16'd0;
76 memoria[32'd68] =6'b010111,5'b01010,5'b11010,16'd3;
77 memoria[32'd69] =6'b011001,5'b01111,5'b01010,16'd0;
78 memoria[32'd70] =6'b000001,5'b11010,5'b11010,16'd5;
79 memoria[32'd71] =6'b010011,5'b11100,21'd1;
80 memoria[32'd72] =6'b000011,5'b11010,5'b11010,16'd5;
```

```
memoria[32'd73] =6'b011001,5'b00111,5'b11111,16'd0;
memoria[32'd74] =6'b011000,5'b00000,5'b11010,5'b00111,11'd5;
memoria[32'd75] =6'b010111,5'b01011,5'b11010,16'd6;
memoria[32'd76] =6'b010111,5'b00001,5'b11010,16'd5;
memoria[32'd77] =6'b010111,5'b11010,5'b01100,16'd1;
memoria[32'd78] =6'b000000,5'b00001,5'b01100,5'b00001,11'd0;
memoria[32'd79] =6'b010111,5'b00001,5'b01100,16'd0;
memoria[32'd80] =6'b011001,5'b01011,5'b01100,16'd0;
memoria[32'd81] =6'b011000,5'b00000,5'b11010,5'b01011,11'd6;
memoria[32'd82] =6'b010111,5'b00011,5'b11010,16'd5;
memoria[32'd83] =6'b010111,5'b11010,5'b00010,16'd1;
memoria[32'd84] =6'b000000,5'b00011,5'b00010,5'b00011,11'd0;
memoria[32'd85] =6'b010111,5'b00011,5'b00010,16'd0;
memoria[32'd86] =6'b010111,5'b00101,5'b11010,16'd4;
memoria[32'd87] =6'b010111,5'b11010,5'b00100,16'd1;
memoria[32'd88] =6'b000000,5'b00101,5'b00100,5'b00101,11'd0;
memoria[32'd89] =6'b010111,5'b00101,5'b00100,16'd0;
memoria[32'd90] =6'b011001,5'b00010,5'b00100,16'd0;
memoria[32'd91] =6'b011000,5'b00000,5'b11010,5'b00010,11'd1;
memoria[32'd92] =6'b010111,5'b00111,5'b11010,16'd4;
memoria[32'd93] =6'b010111,5'b11010,5'b00110,16'd1;
memoria[32'd94] =6'b000000,5'b00111,5'b00110,5'b00111,11'd0;
memoria[32'd95] =6'b010111,5'b00111,5'b00110,16'd0;
memoria[32'd96] =6'b010111,5'b01000,5'b11010,16'd6;
memoria[32'd97] =6'b011001,5'b00110,5'b01000,16'd0;
memoria[32'd98] =6'b011000,5'b00000,5'b11010,5'b00110,11'd1;
memoria[32'd99] =6'b010111,5'b01001,5'b11010,16'd4;
memoria[32'd100] =6'b010111,5'b01010,5'b11010,16'd4;
memoria[32'd101] =6'b011010,5'b01011,5'b00000,16'd1;
memoria[32'd102] =6'b000000,5'b01100,5'b01010,5'b01011,11'd0;
memoria[32'd103] =6'b011001,5'b01001,5'b01100,16'd0;
memoria[32'd104] =6'b011000,5'b00000,5'b11010,5'b01001,11'd4;
memoria[32'd105] =6'b010001,26'd58;
//.L5
memoria[32'd106] =6'b010010,5'b11100,21'd0;
//.main
memoria[32'd107] =6'b010111,5'b00001,5'b11010,16'd1;
memoria[32'd108] =6'b011010,5'b00010,5'b00000,16'd0;
memoria[32'd109] =6'b011001,5'b00001,5'b00010,16'd0;
memoria[32'd110] =6'b011000,5'b00000,5'b11010,5'b00001,11'd1;
//.L6
memoria[32'd111] =6'b010111,5'b00011,5'b11010,16'd1;
memoria[32'd112] =6'b011010,5'b00100,5'b00000,16'd3;
memoria[32'd113] =6'b100010,5'b00011,5'b00100,16'd127;
memoria[32'd114] =6'b010111,5'b00110,5'b11010,16'd1;
memoria[32'd115] =6'b000000,5'b00110,5'b11011,5'b00110,11'd0;
memoria[32'd116] =6'b010111,5'b00110,5'b00101,16'd0;
memoria[32'd117] =6'b011101,5'b11111,21'd0;
memoria[32'd118] =6'b011001,5'b00101,5'b11111,16'd0;
memoria[32'd119] =6'b011000,5'b00000,5'b00110,5'b00101,11'd0;
memoria[32'd120] =6'b010111,5'b00111,5'b11010,16'd1;
memoria[32'd121] =6'b010111,5'b01000,5'b11010,16'd1;
memoria[32'd122] =6'b011010,5'b01001,5'b00000,16'd1;
memoria[32'd123] =6'b000000,5'b01010,5'b01000,5'b01001,11'd0;
memoria[32'd124] =6'b011001,5'b00111,5'b01010,16'd0;
memoria[32'd125] =6'b011000,5'b00000,5'b11010,5'b00111,11'd1;
memoria[32'd126] =6'b010001,26'd111;
//.L7
memoria[32'd127] =6'b010111,5'b01011,5'b11011,16'd0;
memoria[32'd128] =6'b011001,5'b01101,5'b01011,16'd0;
```

```
141 memoria[32'd129] =6'b011010,5'b01100,5'b00000,16'd0;
142 memoria[32'd130] =6'b011001,5'b01110,5'b01100,16'd0;
143 memoria[32'd131] =6'b011010,5'b00001,5'b00000,16'd3;
144 memoria[32'd132] =6'b011001,5'b01111,5'b00001,16'd0;
145 memoria[32'd133] =6'b000001,5'b11010,5'b11010,16'd1;
146 memoria[32'd134] =6'b010011,5'b11100,21'd52;
147 memoria[32'd135] =6'b000011,5'b11010,5'b11010,16'd1;
148 memoria[32'd136] =6'b010111,5'b00010,5'b11010,16'd1;
149 memoria[32'd137] =6'b011010,5'b00011,5'b00000,16'd0;
150 memoria[32'd138] =6'b011001,5'b00010,5'b00011,16'd0;
151 memoria[32'd139] =6'b011000,5'b00000,5'b11010,5'b00010,11'd1;
152 //.L8
153 memoria[32'd140] =6'b010111,5'b00100,5'b11010,16'd1;
154 memoria[32'd141] =6'b011010,5'b00101,5'b00000,16'd3;
155 memoria[32'd142] =6'b100010,5'b00100,5'b00101,16'd156;
156 memoria[32'd143] =6'b010111,5'b00111,5'b11010,16'd1;
157 memoria[32'd144] =6'b000000,5'b00111,5'b11011,5'b00111,11'd0;
158 memoria[32'd145] =6'b010111,5'b00111,5'b00110,16'd0;
159 memoria[32'd146] =6'b011001,5'b01101,5'b00110,16'd0;
160 memoria[32'd147] =6'b011110,5'b00000,5'b01101,16'd0;
161 memoria[32'd148] =6'b100000,5'b00000,5'b00000,5'b00000,11'd0;
162 memoria[32'd149] =6'b010111,5'b01000,5'b11010,16'd1;
163 memoria[32'd150] =6'b010111,5'b01001,5'b11010,16'd1;
164 memoria[32'd151] =6'b011010,5'b01010,5'b00000,16'd1;
165 memoria[32'd152] =6'b000000,5'b01011,5'b01001,5'b01010,11'd0;
166 memoria[32'd153] =6'b011001,5'b01000,5'b01011,16'd0;
167 memoria[32'd154] =6'b011000,5'b00000,5'b11010,5'b01000,11'd1;
168 memoria[32'd155] =6'b010001,26'd140;
169 //.L9
170 memoria[32'd156] =6'b010001,26'd157;
171 //.end
172 memoria[32'd157] =6'b011111,5'b00000,5'b00000,5'b00000,11'd0;
```

Figura 10 – Vetor como variável global

```
int vet[ 10 ];          (alloc, vet, 10, global )
```

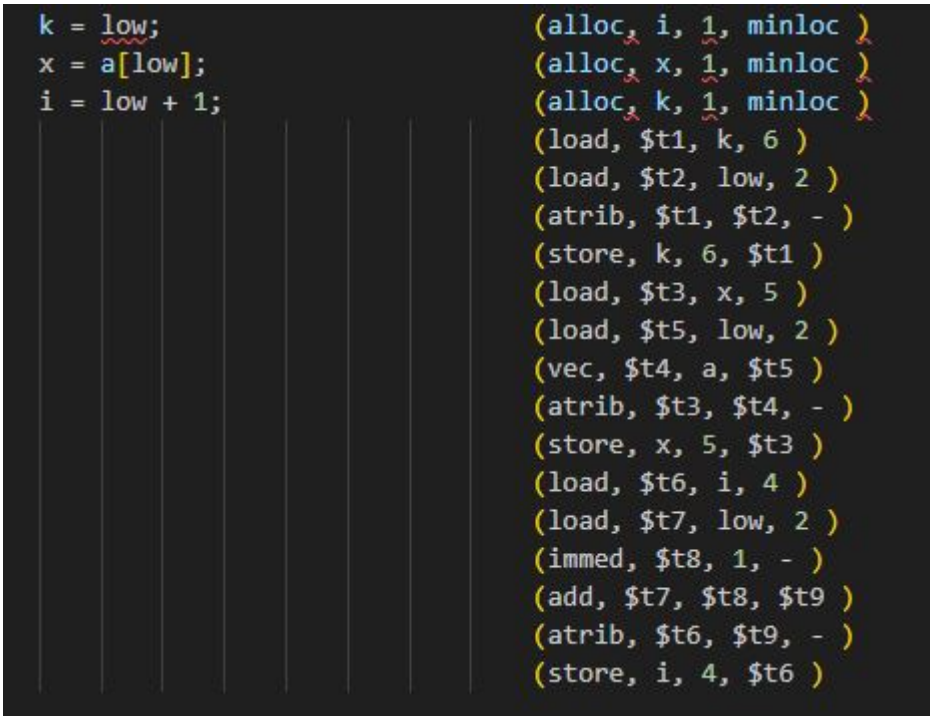
Fonte: O autor

Figura 11 – função minloc pt.1

```
int minloc ( int a[], int low, int high )          (fun, minloc, 0, - )
                                                   (arg, low, 2, minloc )
                                                   (arg, high, 3, minloc )
```

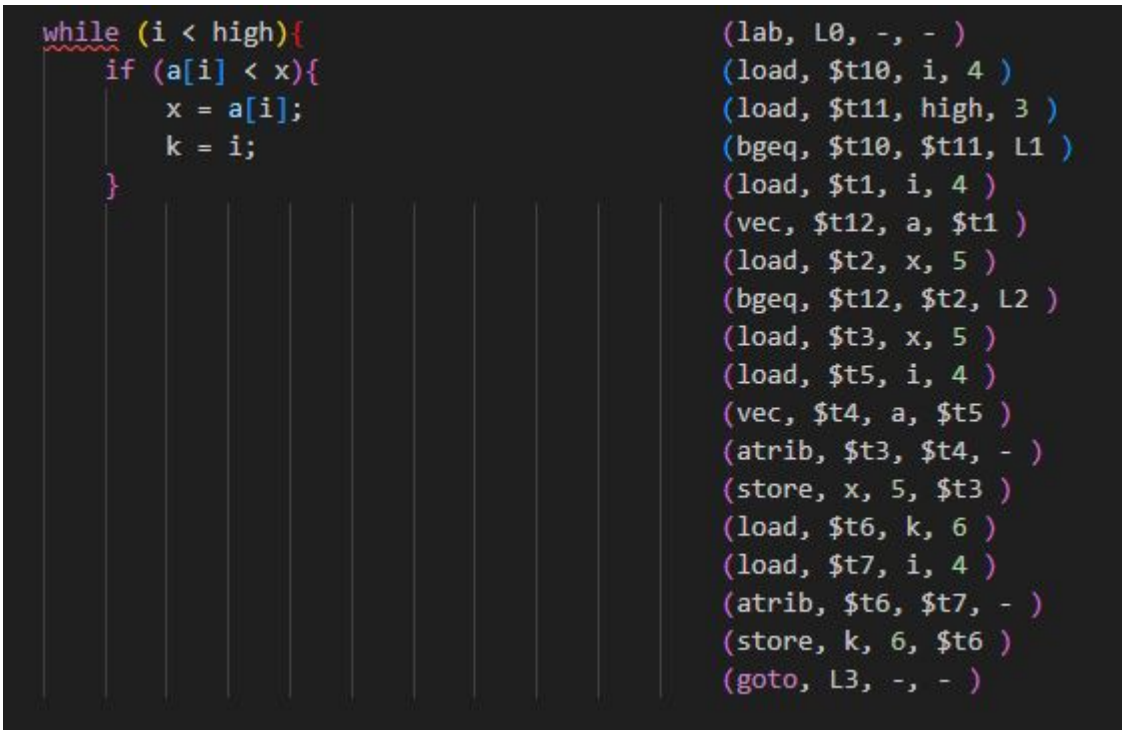
Fonte: O autor

Figura 12 – função minloc pt.2



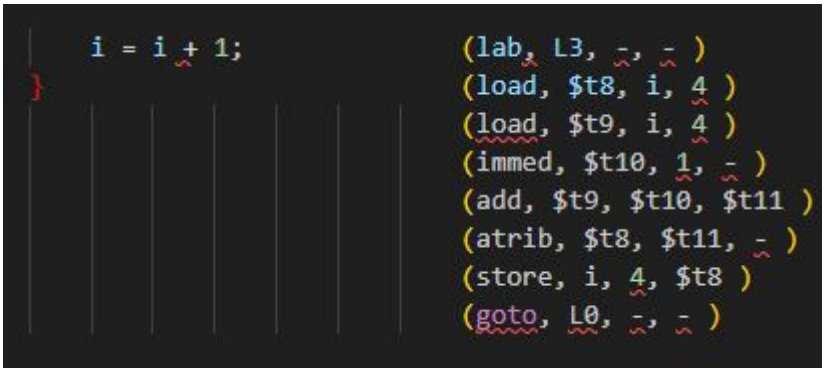
Fonte: O autor

Figura 13 – função minloc pt.3



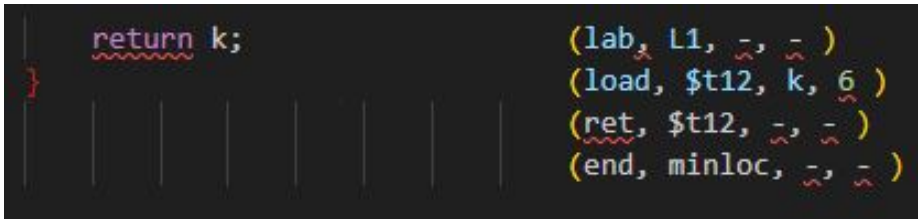
Fonte: O autor

Figura 14 – função minloc pt.4



Fonte: O autor

Figura 15 – função minloc pt.5



Fonte: O autor

Figura 16 – Função sort pt.1

Fonte: O autor

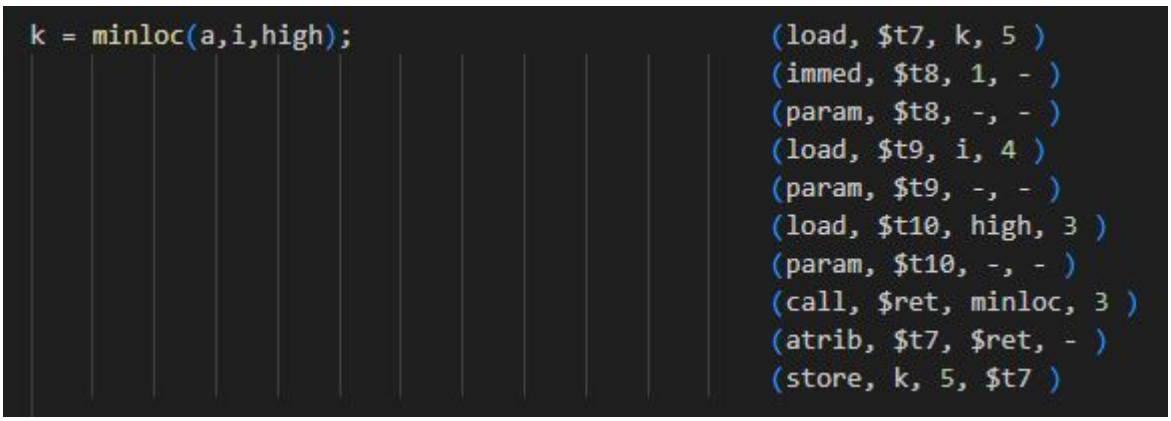
Figura 17 – Função sort pt.2

Fonte: O autor

Figura 18 – Função sort pt.3

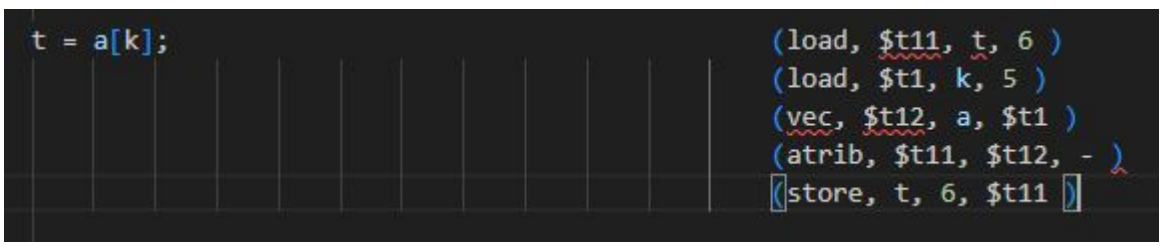
Fonte: O autor

Figura 19 – Função sort pt.4



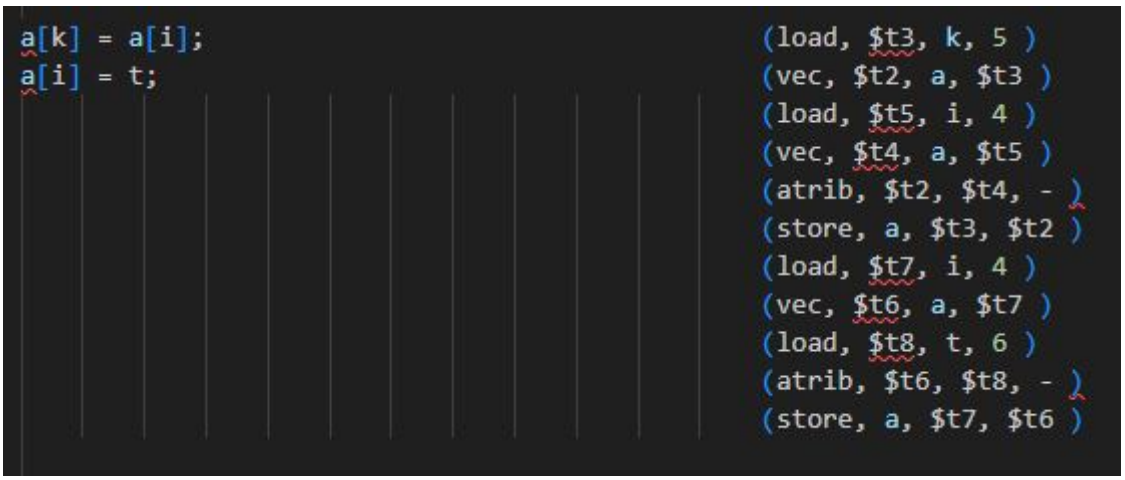
Fonte: O autor

Figura 20 – Função sort pt.5



Fonte: O autor

Figura 21 – Função sort pt.6



Fonte: O autor

Figura 22 – Função sort pt.7

Fonte: O autor

5.1.5 Correspondência entre o código intermediário e o assembly

Figura 23 – Função sort pt.7

Fonte: O autor

5.2 Gcd

5.2.1 Código Fonte

```
1  int gcd (int u, int v){
2      if ( v == 0 ){
3          return u;
4      }
5      else {
6          return gcd(v, u-u / v*v);
7      }
8  }
9
10 void main (void){
11     int x;
12     int y;
13
14     x = input ();
15     y = input ();
16     output (gcd (x ,y));
17 }
```

5.2.2 Código Intermediário

```
1  (fun, gcd, 0, - )
2  (arg, u, 1, gcd )
3  (arg, v, 2, gcd )
4  (load, $t1, v, 2 )
5  (immed, $t2, 0, - )
6  (bne, $t1, $t2, L0 )
7  (load, $t3, u, 1 )
8  (ret, $t3, -, - )
9  (goto, L1, -, - )
10 (lab, L0, -, - )
```

```
(load, $t4, v, 2 )
(param, $t4, -, - )
(load, $t5, u, 1 )
(load, $t6, u, 1 )
(load, $t7, v, 2 )
(div, $t6, $t7, $t8 )
(load, $t9, v, 2 )
(mult, $t8, $t9, $t10 )
(sub, $t5, $t10, $t11 )
(param, $t11, -, - )
(call, $ret, gcd, 2 )
(ret, $ret, -, - )
(lab, L1, -, - )
(end, gcd, -, - )
(fun, main, 0, - )
(alloc, y, 1, main )
(alloc, x, 1, main )
(load, $t12, x, 2 )
(call, $ret, input, 0 )
(atrib, $t12, $ret, - )
(store, x, 2, $t12 )
(load, $t1, y, 1 )
(call, $ret, input, 0 )
(atrib, $t1, $ret, - )
(store, y, 1, $t1 )
(load, $t2, x, 2 )
(param, $t2, -, - )
(load, $t3, y, 1 )
(param, $t3, -, - )
(call, $ret, gcd, 2 )
(param, $ret, -, - )
(call, $ret, output, 1 )
(end, main, -, - )
(end, -, -, - )
```

5.2.3 Código Assembly

```
0:      j 26
.gcd
1:      sw $p1, $sp, 1
2:      sw $p2, $sp, 2
3:      lw $t1, $sp, 2
4:      movi $t2, $zero, 0
5:      bne $t1, $t2, 10
6:      lw $t3, $sp, 1
7:      add $ret, $zero, $t3
8:      jra
9:      j 25
.L0
10:     lw $t4, $sp, 2
11:     mov $p1, $t4, 0
12:     lw $t5, $sp, 1
13:     lw $t6, $sp, 1
14:     lw $t7, $sp, 2
15:     div $t8, $t6, $t7
16:     lw $t9, $sp, 2
17:     mult $t10, $t8, $t9
18:     sub $t11, $t5, $t10
```

```
19:     mov $p2, $t11, 0
20:     addi $sp, $sp, 2
21:     jal 1
22:     subi $sp, $sp, 2
23:     add $ret, $zero, $ret
24:     jra
.L1
25:     jra
.main
26:     lw $t12, $sp, 2
27:     in $ret
28:     mov $t12, $ret, 0
29:     sw $t12, $sp, 2
30:     lw $t1, $sp, 1
31:     in $ret
32:     mov $t1, $ret, 0
33:     sw $t1, $sp, 1
34:     lw $t2, $sp, 2
35:     mov $p1, $t2, 0
36:     lw $t3, $sp, 1
37:     mov $p2, $t3, 0
38:     addi $sp, $sp, 2
39:     jal 1
40:     subi $sp, $sp, 2
41:     mov $p1, $ret, 0
42:     out $p1
43:     pause $zero, $zero, $zero
44:     j 45
.end
45:     end
```

5.2.4 Código executável

```
memoria[32'd0] =6'b010001,26'd26;
//.gcd
memoria[32'd1] =6'b011000,5'b00000,5'b11010,5'b01101,11'd1;
memoria[32'd2] =6'b011000,5'b00000,5'b11010,5'b01110,11'd2;
memoria[32'd3] =6'b010111,5'b00001,5'b11010,16'd2;
memoria[32'd4] =6'b011010,5'b00010,5'b00000,16'd0;
memoria[32'd5] =6'b010101,5'b00001,5'b00010,16'd10;
memoria[32'd6] =6'b010111,5'b00011,5'b11010,16'd1;
memoria[32'd7] =6'b000000,5'b11111,5'b00000,5'b00011,11'd0;
memoria[32'd8] =6'b010010,5'b11100,21'd0;
memoria[32'd9] =6'b010001,26'd25;
//.L0
memoria[32'd10] =6'b010111,5'b00100,5'b11010,16'd2;
memoria[32'd11] =6'b011001,5'b01101,5'b00100,16'd0;
memoria[32'd12] =6'b010111,5'b00101,5'b11010,16'd1;
memoria[32'd13] =6'b010111,5'b00110,5'b11010,16'd1;
memoria[32'd14] =6'b010111,5'b00111,5'b11010,16'd2;
memoria[32'd15] =6'b000110,5'b01000,5'b00110,5'b00111,11'd0;
memoria[32'd16] =6'b010111,5'b01001,5'b11010,16'd2;
memoria[32'd17] =6'b000100,5'b01010,5'b01000,5'b01001,11'd0;
memoria[32'd18] =6'b000010,5'b01011,5'b00101,5'b01010,11'd0;
memoria[32'd19] =6'b011001,5'b01110,5'b01011,16'd0;
memoria[32'd20] =6'b000001,5'b11010,5'b11010,16'd2;
memoria[32'd21] =6'b010011,5'b11100,21'd1;
memoria[32'd22] =6'b000011,5'b11010,5'b11010,16'd2;
```

```
26 memoria[32'd23] =6'b000000,5'b11111,5'b00000,5'b11111,11'd0;
27 memoria[32'd24] =6'b010010,5'b11100,21'd0;
28 //.L1
29 memoria[32'd25] =6'b010010,5'b11100,21'd0;
30 //.main
31 memoria[32'd26] =6'b010111,5'b01100,5'b11010,16'd2;
32 memoria[32'd27] =6'b011101,5'b11111,21'd0;
33 memoria[32'd28] =6'b011001,5'b01100,5'b11111,16'd0;
34 memoria[32'd29] =6'b011000,5'b00000,5'b11010,5'b01100,11'd2;
35 memoria[32'd30] =6'b010111,5'b00001,5'b11010,16'd1;
36 memoria[32'd31] =6'b011101,5'b11111,21'd0;
37 memoria[32'd32] =6'b011001,5'b00001,5'b11111,16'd0;
38 memoria[32'd33] =6'b011000,5'b00000,5'b11010,5'b00001,11'd1;
39 memoria[32'd34] =6'b010111,5'b00010,5'b11010,16'd2;
40 memoria[32'd35] =6'b011001,5'b01101,5'b00010,16'd0;
41 memoria[32'd36] =6'b010111,5'b00011,5'b11010,16'd1;
42 memoria[32'd37] =6'b011001,5'b01110,5'b00011,16'd0;
43 memoria[32'd38] =6'b000001,5'b11010,5'b11010,16'd2;
44 memoria[32'd39] =6'b010011,5'b11100,21'd1;
45 memoria[32'd40] =6'b000011,5'b11010,5'b11010,16'd2;
46 memoria[32'd41] =6'b011001,5'b01101,5'b11111,16'd0;
47 memoria[32'd42] =6'b011110,5'b00000,5'b01101,16'd0;
48 memoria[32'd43] =6'b100000,5'b00000,5'b00000,5'b00000,11'd0;
49 memoria[32'd44] =6'b010001,26'd45;
50 //.end
51 memoria[32'd45] =6'b011111,5'b00000,5'b00000,5'b00000,11'd0;
```

5.2.5 Correspondência entre o código fonte e o intermediário

Figura 24 – Cabeçalho da função gcd

```
int gcd (int u, int v){
```

```
(fun, gcd, 0, - )
(arg, u, 1, gcd )
(arg, v, 2, gcd )
```

Fonte: O autor

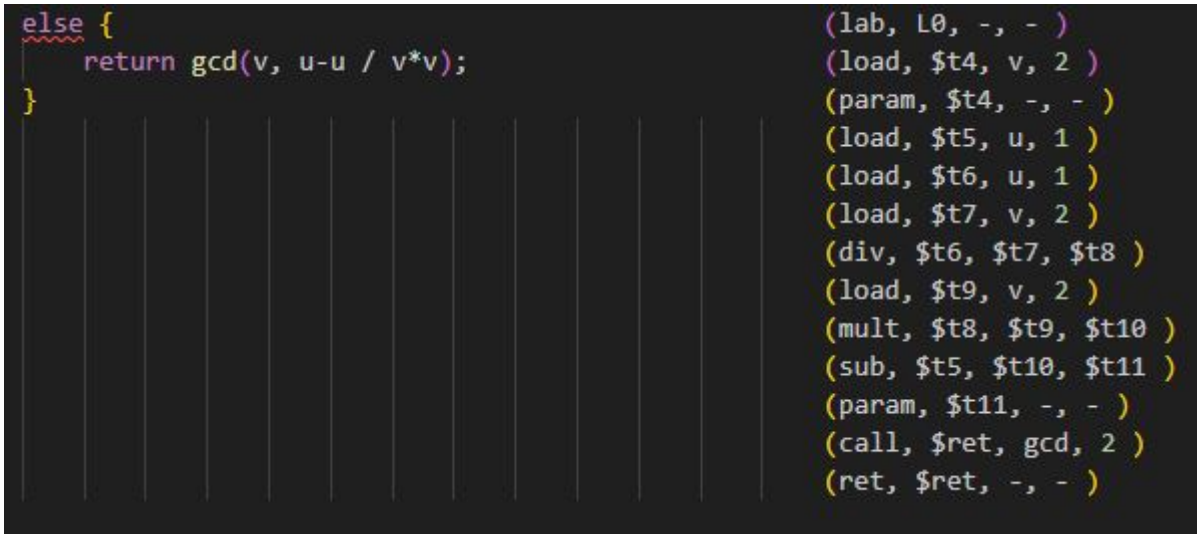
Figura 25 – Estrutura de decisão

```
if ( v == 0 ){
    return u;
}
```

```
(load, $t1, v, 2 )
(immed, $t2, 0, - )
(bne, $t1, $t2, L0 )
(load, $t3, u, 1 )
(ret, $t3, -, - )
(goto, L1, -, - )
```

Fonte: O autor

Figura 26 – Condição falsa na estrutura anterior

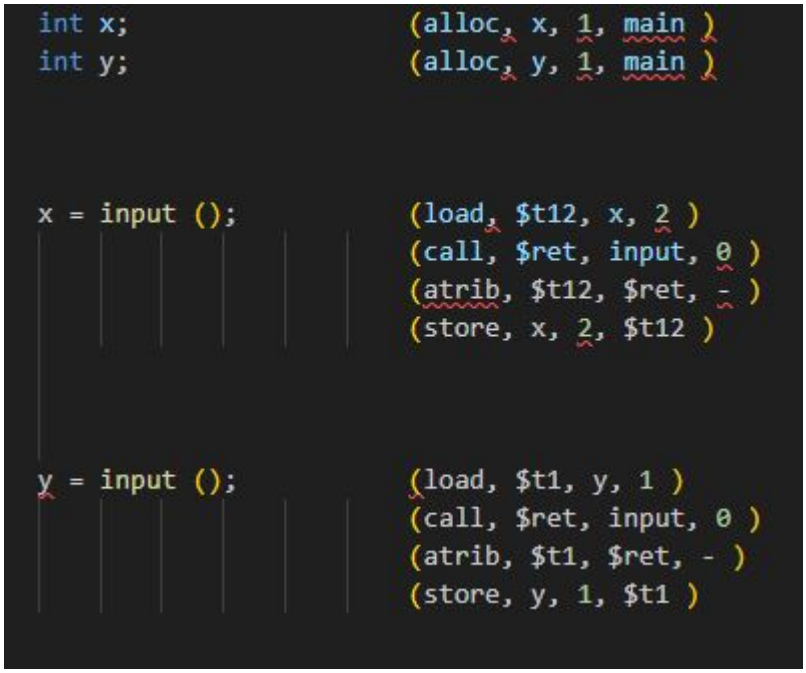


Fonte: O autor

Figura 27 – Cabeçalho da função main

Fonte: O autor

Figura 28 – Declarações de variáveis e entrada de dados



Fonte: O autor

Figura 29 – Saida de dados como retorno de função

```
output (gcd (x ,y));          (load, $t2, x, 2 )
                              (param, $t2, -, - )
                              (load, $t3, y, 1 )
                              (param, $t3, -, - )
                              (call, $ret, gcd, 2 )
                              (param, $ret, -, - )
                              (call, $ret, output, 1 )
```

Fonte: O autor

5.2.6 Correspondência entre o código intermediário e o assembly

Figura 30 – Cabeçalho da função gcd

```
(fun, gcd, 0, - )          .gcd
(arg, u, 1, gcd )           sw $p1, $sp, 1
(arg, v, 2, gcd )           sw $p2, $sp, 2
```

Fonte: O autor

Figura 31 – Estrutura de decisão pt.1

```
(load, $t1, v, 2 )          3:     lw $t1, $sp, 2
(immed, $t2, 0, - )         4:     movi $t2, $zero, 0
(bne, $t1, $t2, L0 )        5:     bne $t1, $t2, 10
```

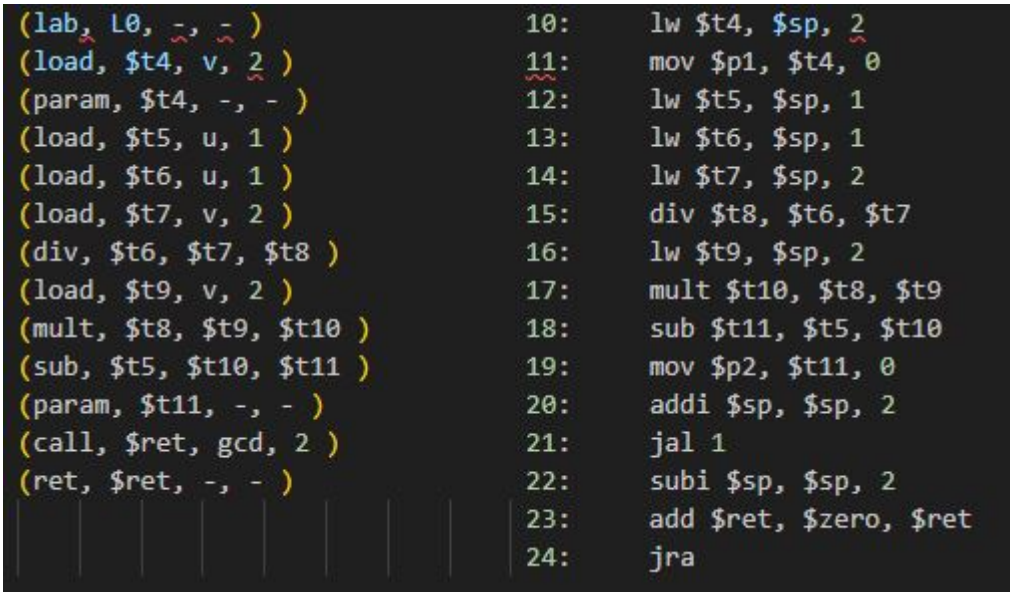
Fonte: O autor

Figura 32 – Estrutura de decisão pt.2

```
(load, $t3, u, 1 )          6:     lw $t3, $sp, 1
(ret, $t3, -, - )           7:     add $ret, $zero, $t3
(goto, L1, -, - )           8:     jra
```

Fonte: O autor

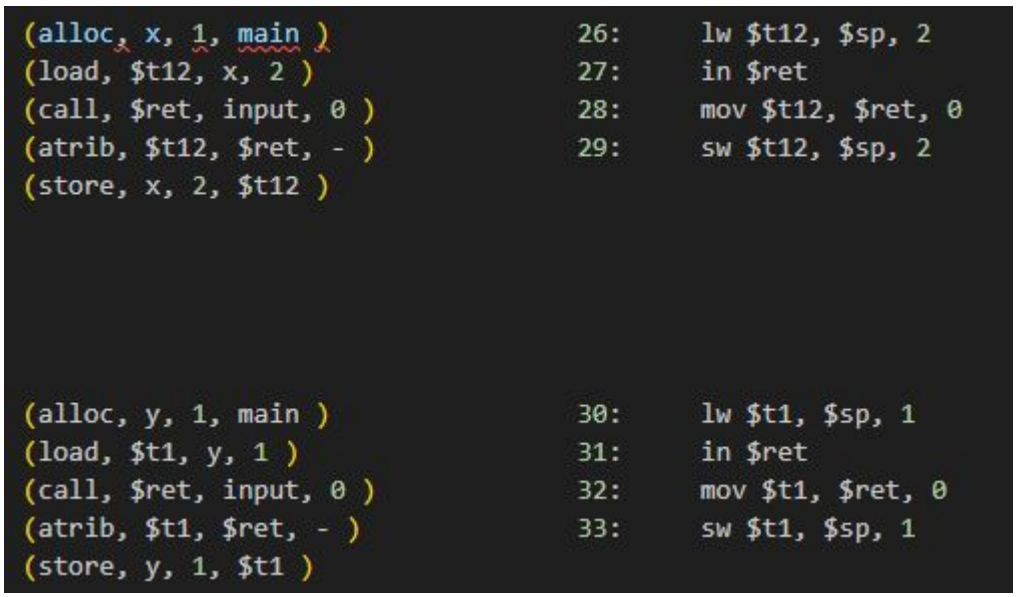
Figura 33 – Condição falsa na estrutura anterior



```
(lab, L0, -, - )                10:     lw $t4, $sp, 2
(load, $t4, v, 2 )              11:     mov $p1, $t4, 0
(param, $t4, -, - )             12:     lw $t5, $sp, 1
(load, $t5, u, 1 )              13:     lw $t6, $sp, 1
(load, $t6, u, 1 )              14:     lw $t7, $sp, 2
(load, $t7, v, 2 )              15:     div $t8, $t6, $t7
(div, $t6, $t7, $t8 )           16:     lw $t9, $sp, 2
(load, $t9, v, 2 )              17:     mult $t10, $t8, $t9
(mult, $t8, $t9, $t10 )         18:     sub $t11, $t5, $t10
(sub, $t5, $t10, $t11 )         19:     mov $p2, $t11, 0
(param, $t11, -, - )            20:     addi $sp, $sp, 2
(call, $ret, gcd, 2 )           21:     jal 1
(ret, $ret, -, - )              22:     subi $sp, $sp, 2
                                23:     add $ret, $zero, $ret
                                24:     jra
```

Fonte: O autor

Figura 34 – Declarações de variáveis e entrada de dados

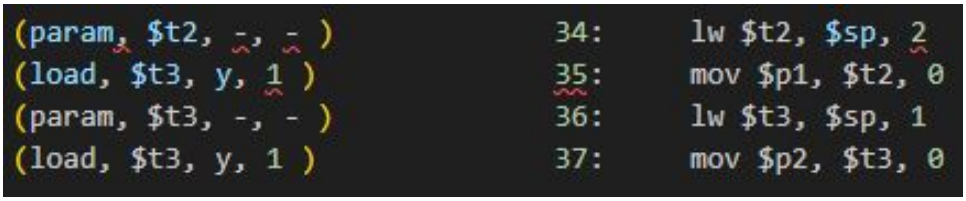


```
(alloc, x, 1, main )            26:     lw $t12, $sp, 2
(load, $t12, x, 2 )             27:     in $ret
(call, $ret, input, 0 )         28:     mov $t12, $ret, 0
(atrib, $t12, $ret, - )         29:     sw $t12, $sp, 2
(store, x, 2, $t12 )

(alloc, y, 1, main )            30:     lw $t1, $sp, 1
(load, $t1, y, 1 )              31:     in $ret
(call, $ret, input, 0 )         32:     mov $t1, $ret, 0
(atrib, $t1, $ret, - )          33:     sw $t1, $sp, 1
(store, y, 1, $t1 )
```

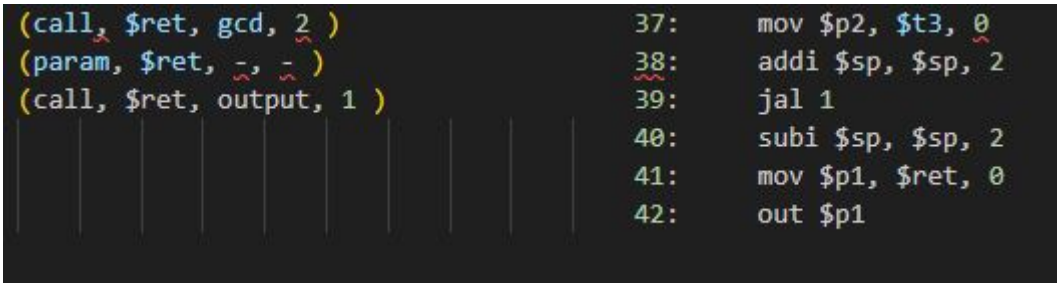
Fonte: O autor

Figura 35 – Alocação dos parâmetros da função



Fonte: O autor

Figura 36 – Saida de dados como retorno de função



Fonte: O autor

5.3 Fibonacci

5.3.1 Código Fonte

```
1  int fibonacci (int n){
2
3      int c;
4      int next;
5      int first;
6      int second;
7      c = 0;
8      first = 0;
9      second = 1;
10     while (c <= n ){
11         if(c <= 1) {
12             next = c;
13         } else {
14             next = first + second;
15             first = second;
16             second = next;
17         }
18         c = c + 1;
19     }
20     return next;
21 }
22
23 void main (void){
24     int n;
25     n = input() ;
26     output(fibonacci(n));
27 }
```

5.3.2 Código Intermediário

```
1  (fun, fibonacci, 0, - )
2  (arg, n, 1, fibonacci )
3  (alloc, c, 1, fibonacci )
4  (alloc, next, 1, fibonacci )
5  (alloc, first, 1, fibonacci )
6  (alloc, second, 1, fibonacci )
7  (load, $t1, c, 2 )
8  (immed, $t2, 0, - )
9  (atrib, $t1, $t2, - )
10 (store, c, 2, $t1 )
11 (load, $t3, first, 4 )
12 (immed, $t4, 0, - )
13 (atrib, $t3, $t4, - )
14 (store, first, 4, $t3 )
15 (load, $t5, second, 5 )
16 (immed, $t6, 1, - )
17 (atrib, $t5, $t6, - )
18 (store, second, 5, $t5 )
19 (lab, L0, -, - )
20 (load, $t7, c, 2 )
21 (load, $t8, n, 1 )
22 (bgt, $t7, $t8, L1 )
23 (load, $t9, c, 2 )
24 (immed, $t10, 1, - )
25 (bgt, $t9, $t10, L2 )
26 (load, $t11, next, 3 )
27 (load, $t12, c, 2 )
28 (atrib, $t11, $t12, - )
29 (store, next, 3, $t11 )
30 (goto, L3, -, - )
31 (lab, L2, -, - )
32 (load, $t1, next, 3 )
33 (load, $t2, first, 4 )
34 (load, $t3, second, 5 )
35 (add, $t2, $t3, $t4 )
36 (atrib, $t1, $t4, - )
37 (store, next, 3, $t1 )
38 (load, $t5, first, 4 )
39 (load, $t6, second, 5 )
40 (atrib, $t5, $t6, - )
41 (store, first, 4, $t5 )
42 (load, $t7, second, 5 )
43 (load, $t8, next, 3 )
44 (atrib, $t7, $t8, - )
45 (store, second, 5, $t7 )
46 (lab, L3, -, - )
47 (load, $t9, c, 2 )
48 (load, $t10, c, 2 )
49 (immed, $t11, 1, - )
50 (add, $t10, $t11, $t12 )
51 (atrib, $t9, $t12, - )
52 (store, c, 2, $t9 )
53 (goto, L0, -, - )
54 (lab, L1, -, - )
55 (load, $t1, next, 3 )
56 (ret, $t1, -, - )
57 (end, fibonacci, -, - )
58 (fun, main, 0, - )
```

```
(alloc, n, 1, main )
(load, $t2, n, 1 )
(call, $ret, input, 0 )
(atrib, $t2, $ret, - )
(store, n, 1, $t2 )
(load, $t3, n, 1 )
(param, $t3, -, - )
(call, $ret, fibonacci, 1 )
(param, $ret, -, - )
(call, $ret, output, 1 )
(end, main, -, - )
(end, -, -, - )
```

5.3.3 Código Assembly

```
0:      j 50
.fibonacci
1:      sw $sp, $p1, 1
2:      lw $sp, $t1, 2
3:      addi $zero, $t2, 0
4:      add $zero, $t2, $t1
5:      sw $sp, $t1, 2
6:      lw $sp, $t3, 4
7:      addi $zero, $t4, 0
8:      add $zero, $t4, $t3
9:      sw $sp, $t3, 4
10:     lw $sp, $t5, 5
11:     addi $zero, $t6, 1
12:     add $zero, $t6, $t5
13:     sw $sp, $t5, 5
.L0
14:     lw $sp, $t7, 2
15:     lw $sp, $t8, 1
16:     bgt $t7, $t8, 46
17:     lw $sp, $t9, 2
18:     addi $zero, $t10, 1
19:     bgt $t9, $t10, 25
20:     lw $sp, $t11, 3
21:     lw $sp, $t12, 2
22:     add $zero, $t12, $t11
23:     sw $sp, $t11, 3
24:     j 39
.L2
25:     lw $sp, $t1, 3
26:     lw $sp, $t2, 4
27:     lw $sp, $t3, 5
28:     add $t2, $t3, $t4
29:     add $zero, $t4, $t1
30:     sw $sp, $t1, 3
31:     lw $sp, $t5, 4
32:     lw $sp, $t6, 5
33:     add $zero, $t6, $t5
34:     sw $sp, $t5, 4
35:     lw $sp, $t7, 5
36:     lw $sp, $t8, 3
37:     add $zero, $t8, $t7
38:     sw $sp, $t7, 5
.L3
```

```
39:     lw $sp, $t9, 2
40:     lw $sp, $t10, 2
41:     addi $zero, $t11, 1
42:     add $t10, $t11, $t12
43:     add $zero, $t12, $t9
44:     sw $sp, $t9, 2
45:     j 14
.L1
46:     lw $sp, $t1, 3
47:     add $zero, $t1, $ret
48:     jra
49:     jra
.main
50:     lw $sp, $t2, 1
51:     in $ret
52:     add $zero, $ret, $t2
53:     sw $sp, $t2, 1
54:     lw $sp, $t3, 1
55:     add $zero, $t3, $p1
56:     addi $sp, $sp, 1
57:     jal 1
58:     subi $sp, $sp, 1
59:     add $zero, $ret, $p1
60:     out $p1
61:     j 62
.end
62:     end
```

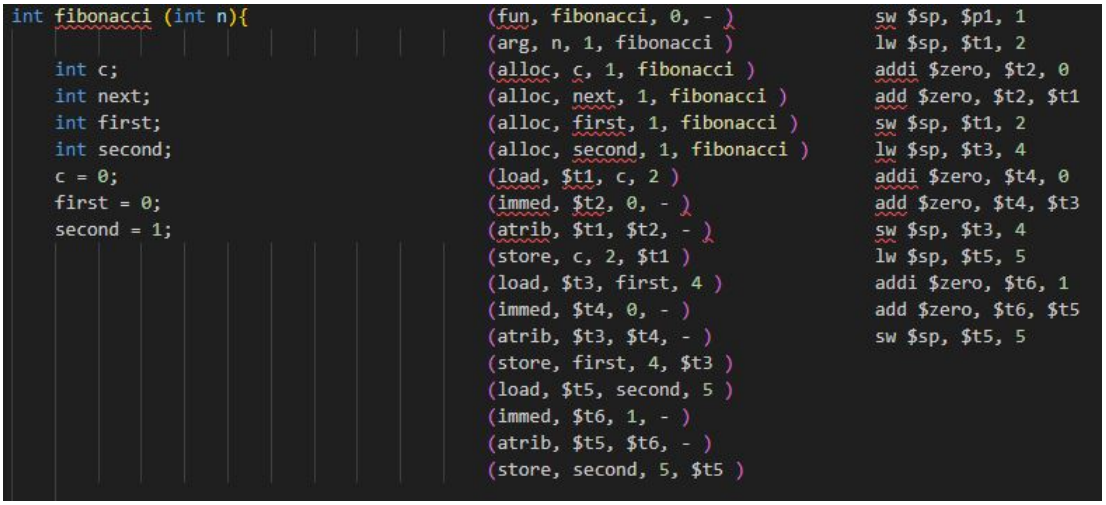
5.3.4 Código executável

```
instrucao[32'd0] =6'b010001_00000000000000000000110010
//.fibonacci
instrucao[32'd1] =6'b011000_5'b11010_5'b01101_00000000000000000001
instrucao[32'd2] =6'b010111_5'b11010_5'b00001_00000000000000000010
instrucao[32'd3] =6'b000001_5'b00000_5'b00010_0000000000000000
instrucao[32'd4] =6'b000000_5'b00000_5'b00010_5'b00001_00000
instrucao[32'd5] =6'b011000_5'b11010_5'b00001_00000000000000000010
instrucao[32'd6] =6'b010111_5'b11010_5'b00011_00000000000000000100
instrucao[32'd7] =6'b000001_5'b00000_5'b00100_0000000000000000
instrucao[32'd8] =6'b000000_5'b00000_5'b00100_5'b00011_00000
instrucao[32'd9] =6'b011000_5'b11010_5'b00011_00000000000000000100
instrucao[32'd10] =6'b010111_5'b11010_5'b00101_00000000000000000101
instrucao[32'd11] =6'b000001_5'b00000_5'b00110_0000000000000001
instrucao[32'd12] =6'b000000_5'b00000_5'b00110_5'b00101_00000
instrucao[32'd13] =6'b011000_5'b11010_5'b00101_00000000000000000101
//.L0
instrucao[32'd14] =6'b010111_5'b11010_5'b00111_00000000000000000010
instrucao[32'd15] =6'b010111_5'b11010_5'b01000_00000000000000000001
instrucao[32'd16] =6'b_5'b00111_5'b01000_0000000000101110
instrucao[32'd17] =6'b010111_5'b11010_5'b01001_00000000000000000010
instrucao[32'd18] =6'b000001_5'b00000_5'b01010_0000000000000001
instrucao[32'd19] =6'b_5'b01001_5'b01010_0000000000011001
instrucao[32'd20] =6'b010111_5'b11010_5'b01011_00000000000000000011
instrucao[32'd21] =6'b010111_5'b11010_5'b01100_00000000000000000010
instrucao[32'd22] =6'b000000_5'b00000_5'b01100_5'b01011_00000
instrucao[32'd23] =6'b011000_5'b11010_5'b01011_00000000000000000011
instrucao[32'd24] =6'b010001_00000000000000000000100111
//.L2
```

```
29 instrucao[32'd25] =6'b010111_5'b11010_5'b00001_00000000000000011
30 instrucao[32'd26] =6'b010111_5'b11010_5'b00010_00000000000000100
31 instrucao[32'd27] =6'b010111_5'b11010_5'b00011_00000000000000101
32 instrucao[32'd28] =6'b000000_5'b00010_5'b00011_5'b00100_00000
33 instrucao[32'd29] =6'b000000_5'b00000_5'b00100_5'b00001_00000
34 instrucao[32'd30] =6'b011000_5'b11010_5'b00001_00000000000000011
35 instrucao[32'd31] =6'b010111_5'b11010_5'b00101_00000000000000100
36 instrucao[32'd32] =6'b010111_5'b11010_5'b00110_00000000000000101
37 instrucao[32'd33] =6'b000000_5'b00000_5'b00110_5'b00101_00000
38 instrucao[32'd34] =6'b011000_5'b11010_5'b00101_00000000000000100
39 instrucao[32'd35] =6'b010111_5'b11010_5'b00111_00000000000000101
40 instrucao[32'd36] =6'b010111_5'b11010_5'b01000_00000000000000011
41 instrucao[32'd37] =6'b000000_5'b00000_5'b01000_5'b00111_00000
42 instrucao[32'd38] =6'b011000_5'b11010_5'b00111_00000000000000101
43 //.L3
44 instrucao[32'd39] =6'b010111_5'b11010_5'b01001_00000000000000010
45 instrucao[32'd40] =6'b010111_5'b11010_5'b01010_00000000000000010
46 instrucao[32'd41] =6'b000001_5'b00000_5'b01011_0000000000000001
47 instrucao[32'd42] =6'b000000_5'b01010_5'b01011_5'b01100_00000
48 instrucao[32'd43] =6'b000000_5'b00000_5'b01100_5'b01001_00000
49 instrucao[32'd44] =6'b011000_5'b11010_5'b01001_00000000000000010
50 instrucao[32'd45] =6'b010001_00000000000000000000000001110
51 //.L1
52 instrucao[32'd46] =6'b010111_5'b11010_5'b00001_00000000000000011
53 instrucao[32'd47] =6'b000000_5'b00000_5'b00001_5'b11111_00000
54 instrucao[32'd48] =6'b010010_5'b11100_00000000000000000000
55 instrucao[32'd49] =6'b010010_5'b11100_00000000000000000000
56 //.main
57 instrucao[32'd50] =6'b010111_5'b11010_5'b00010_00000000000000001
58 instrucao[32'd51] =6'b011101_5'b00000_5'b11111_00000000000000000
59 instrucao[32'd52] =6'b000000_5'b00000_5'b11111_5'b00010_00000
60 instrucao[32'd53] =6'b011000_5'b11010_5'b00010_00000000000000001
61 instrucao[32'd54] =6'b010111_5'b11010_5'b00011_00000000000000001
62 instrucao[32'd55] =6'b000000_5'b00000_5'b00011_5'b01101_00000
63 instrucao[32'd56] =6'b000001_5'b11010_5'b11010_0000000000000001
64 instrucao[32'd57] =6'b010011_00000000000000000000000000001
65 instrucao[32'd58] =6'b000011_5'b11010_5'b11010_0000000000000001
66 instrucao[32'd59] =6'b000000_5'b00000_5'b11111_5'b01101_00000
67 instrucao[32'd60] =6'b011110_5'b01101_00000000000000000000000
68 instrucao[32'd61] =6'b010001_00000000000000000000000111110
69 //.end
70 instrucao[32'd62] =6'b011111_5'b00000_5'b00000_5'b00000_00000
```

5.3.5 Correspondência entre o códigos

Figura 37 – Fibonacci pt.1



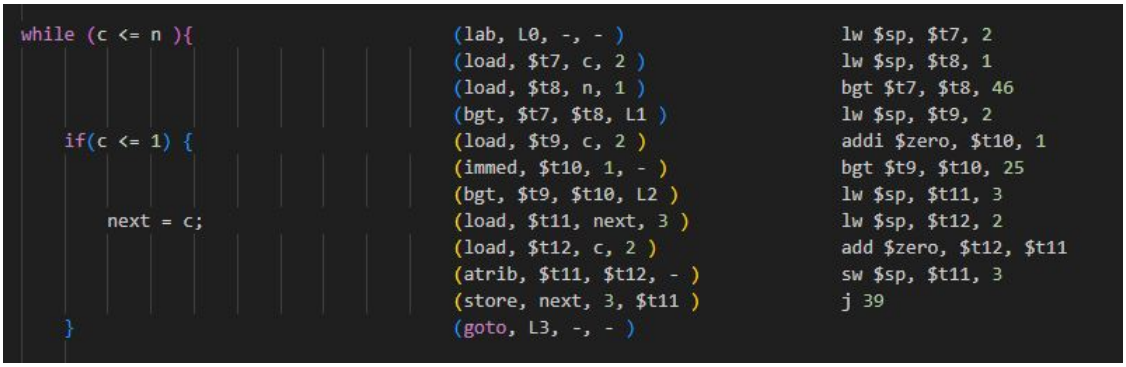
```
int fibonacci (int n){
    int c;
    int next;
    int first;
    int second;
    c = 0;
    first = 0;
    second = 1;
```

```
(fun, fibonacci, 0, - )
(arg, n, 1, fibonacci )
(alloc, c, 1, fibonacci )
(alloc, next, 1, fibonacci )
(alloc, first, 1, fibonacci )
(alloc, second, 1, fibonacci )
(load, $t1, c, 2 )
(immed, $t2, 0, - )
(atrib, $t1, $t2, - )
(store, c, 2, $t1 )
(load, $t3, first, 4 )
(immed, $t4, 0, - )
(atrib, $t3, $t4, - )
(store, first, 4, $t3 )
(load, $t5, second, 5 )
(immed, $t6, 1, - )
(atrib, $t5, $t6, - )
(store, second, 5, $t5 )
```

```
sw $sp, $p1, 1
lw $sp, $t1, 2
addi $zero, $t2, 0
add $zero, $t2, $t1
sw $sp, $t1, 2
lw $sp, $t3, 4
addi $zero, $t4, 0
add $zero, $t4, $t3
sw $sp, $t3, 4
lw $sp, $t5, 5
addi $zero, $t6, 1
add $zero, $t6, $t5
sw $sp, $t5, 5
```

Fonte: O autor

Figura 38 – Fibonacci pt.2



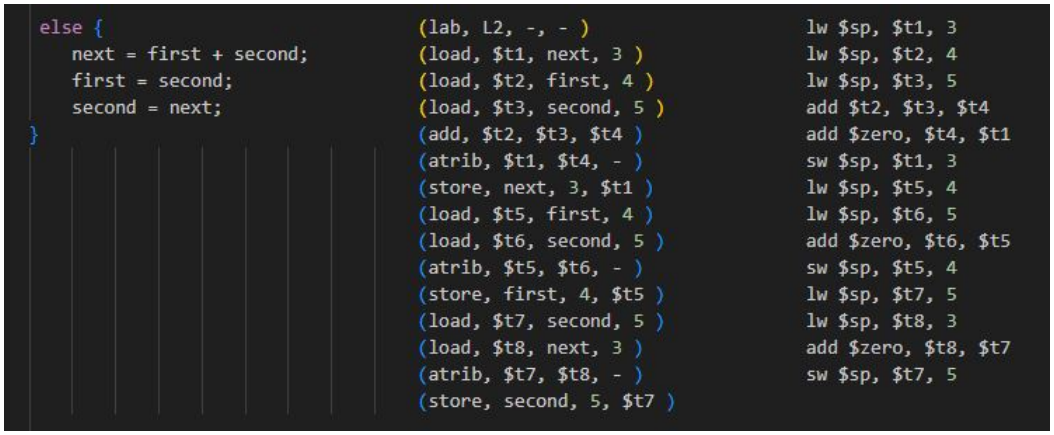
```
while (c <= n ){
    if(c <= 1) {
        next = c;
    }
```

```
(lab, L0, -, - )
(load, $t7, c, 2 )
(load, $t8, n, 1 )
(bgt, $t7, $t8, L1 )
(load, $t9, c, 2 )
(immed, $t10, 1, - )
(bgt, $t9, $t10, L2 )
(load, $t11, next, 3 )
(load, $t12, c, 2 )
(atrib, $t11, $t12, - )
(store, next, 3, $t11 )
(goto, L3, -, - )
```

```
lw $sp, $t7, 2
lw $sp, $t8, 1
bgt $t7, $t8, 46
lw $sp, $t9, 2
addi $zero, $t10, 1
bgt $t9, $t10, 25
lw $sp, $t11, 3
lw $sp, $t12, 2
add $zero, $t12, $t11
sw $sp, $t11, 3
j 39
```

Fonte: O autor

Figura 39 – Fibonacci pt.3



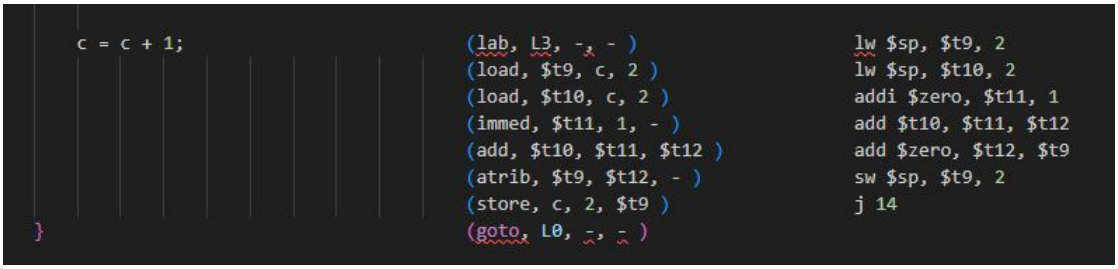
```
else {
    next = first + second;
    first = second;
    second = next;
}
```

```
(lab, L2, -, - )
(load, $t1, next, 3 )
(load, $t2, first, 4 )
(load, $t3, second, 5 )
(add, $t2, $t3, $t4 )
(atrib, $t1, $t4, - )
(store, next, 3, $t1 )
(load, $t5, first, 4 )
(load, $t6, second, 5 )
(atrib, $t5, $t6, - )
(store, first, 4, $t5 )
(load, $t7, second, 5 )
(load, $t8, next, 3 )
(atrib, $t7, $t8, - )
(store, second, 5, $t7 )
```

```
lw $sp, $t1, 3
lw $sp, $t2, 4
lw $sp, $t3, 5
add $t2, $t3, $t4
add $zero, $t4, $t1
sw $sp, $t1, 3
lw $sp, $t5, 4
lw $sp, $t6, 5
add $zero, $t6, $t5
sw $sp, $t5, 4
lw $sp, $t7, 5
lw $sp, $t8, 3
add $zero, $t8, $t7
sw $sp, $t7, 5
```

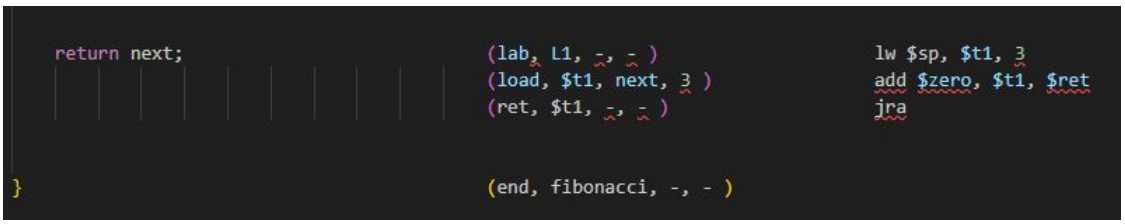
Fonte: O autor

Figura 40 – Fibonacci pt.4



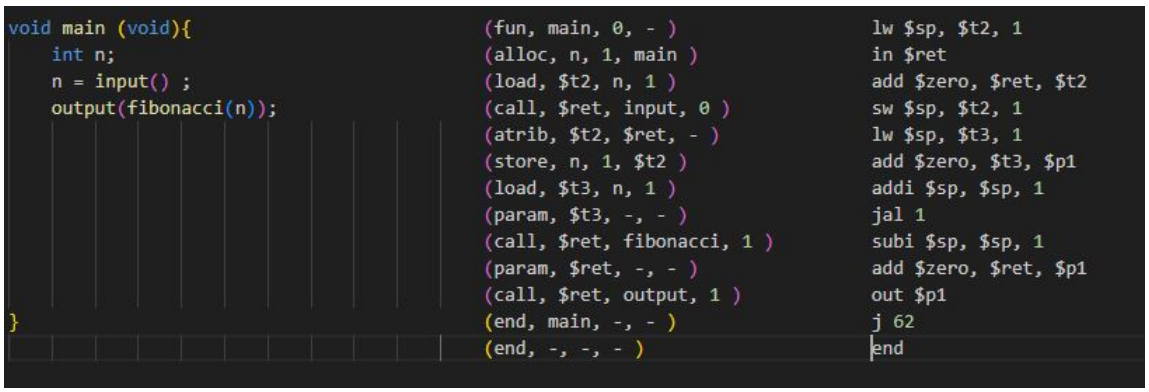
Fonte: O autor

Figura 41 – Fibonacci pt.5



Fonte: O autor

Figura 42 – Fibonacci pt.6



Fonte: O autor

6 Conclusão

Neste trabalho foi possível aplicar os conhecimentos teóricos de compiladores, permindo-se de fato a real compreensão dos assuntos estudados na unidade curricular de compiladores. Sendo necessário varias horas para o entendimento de como cada modulo do compilador funcionava e se comunicavam de forma prática.

Durante a implementação do compilador o principal desafio foi entender como as chamadas de função, os vetores e as pilhas de recursão funcionam no processador implementado anteriormente, para que se então fosse implementado no compilador.

Por fim, foram necessárias modificações na estrutura do processador, para comportar processos necessários para a execução dos algoritmos, que não haviam sido observados no laboratório anterior.

Referências

1 JOHANN, M. *Análise Léxica*. Disponível em: <https://www.inf.ufrgs.br/~johann/comp/aula02.lexica.pdf>. Acesso em: 7 jun. 2023. Citado na página 35.

2 LOUDEN, K. C. *Compiladores-Princípios e Práticas*. [S.l.]: Cengage Learning Editores, 2004. Citado 3 vezes nas páginas 37, 38 e 41.

3 AHO, A. V.; SETHI, R.; ULLMAN, J. D. Compiladores: Princıpios, técnicas e ferramentas. *LTC, Rio de Janeiro, Brasil*, 1995. Citado 2 vezes nas páginas 38 e 40.

4 COMPILADORES para humanos. Disponível em: <https://johnidm.gitbooks.io/compiladores-para-humanos/content/>. Acesso em: 7 jun. 2023. Citado na página 38.

5 TANENBAUM, A. S.; ZUCCHI, W. L. *Organização estruturada de computadores*. [S.l.]: Pearson Prentice Hall, 2009. Citado na página 50.