Edna Mie Kanazawa

Desenvolvimento de um Sistema Integrado de Compilação, Montagem e Simulação para Programação de um Processador Discreto Microprogramável

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Wilian Soares Lacerda

Lavras
Minas Gerais - Brasil
2001

Edna Mie Kanazawa

Desenvolvimento de um Sistema Integrado de Compilação, Montagem e Simulação para Programação de um Processador Discreto Microprogramável

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *27 de junho de 2001*

Prof. Antônio Maria P. Rezende

Prof. João Carlos Giacomin

Prof. Wilian Soares Lacerda
(Orientador)

Lavras
Minas Gerais - Brasil

Especialmente à minha mãe, que mesmo distante sabe medir palavras de incentivo. Ao Fábio, companheiro de todos os momentos. Às irmãs e amigas Júlia e Alice. E a todos amigos...

Agradecimentos

Ao orientador, Wilian Soares Lacerda, que tornou possível o desenvolvimento da Unidade Processadora Discreta Microprogramável e do Sistema Integrado de Compilação, Montagem e Simulação.

Aos professores do Departamento de Ciência da Computação, que ajudaram de diversas formas no desenvolvimento e na implementação dos projetos.

Resumo

Processadores dedicados são freqüentemente encontrados em nosso cotidiano. A geração de aplicações para estes processadores envolvem diversos processos, que se iniciam com um programa em uma linguagem de alto nível até a inserção do programa no processador.

O objetivo deste projeto é elaborar um sistema integrado através do qual é possível verificar os processos envolvidos na geração de aplicações para Unidade Processadora Discreta Microprogramável, desenvolvida no Programa Institucional de Bolsas de Iniciação Científica - PIBIC/CNPq (período de setembro/1999 a julho/2000).

O sistema integrado constitui-se de três módulos: Módulo de Compilação, Módulo de Montagem e o Módulo de Simulação.

O processo de compilação permite que as aplicações sejam implementadas em linguagem de alto nível. O compilador realiza a tradução de um programa em alto nível para uma linguaguem em baixo nível.

O processo de montagem converte as instruções em linguagem assembly para linguagem de máquina (códigos binários) a ser executado pelo processador.

O processo de simulação permite ao desenvolvedor executar virtualmente as instruções da aplicação antes de implementá-las no processador.

O sistema integrado juntamente com a Unidade Processadora Discreta Microprogramável descreve todos os processos utilizados para geração de uma aplicação para um processador dedicado.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Grande parte dos equipamentos eletrônicos, que vão desde brinquedos a computadores de bordo utilizam um processador dedicado. Devido a isso todo aluno de Ciência da Computação, além da programação, deve ter conhecimentos básicos de como é o processamento dentro do computador.

A geração de aplicações para esses processadores envolvem diversos processos, que se iniciam com um programa desenvolvido em uma linguagem de alto nível até o processamento deste código pelo processador.

O objetivo deste projeto é elaborar um sistema integrado através do qual é possível verificar os processos empregados para a geração de aplicações para um processador dedicado.

O processador dedicado utilizado denomina-se Unidade Processadora Discreta Microprogramável (UPDM)[KL00], desenvolvido no Programa Institucional de Bolsas de Iniciação Científica – PIBIC/CNPq período de setembro/1999 a julho/2000.

O sistema integrado constitui-se de 3 módulos: compilador, montador e simulador, desenvolvidos para programação da UPDM.

O trabalho dividiu-se em 5 capítulos descritos nos itens a seguir:

- Capítulo 2 – É feito uma breve descrição da Unidade Processadora Discreta – UPDM. São descritas também as instruções disponíveis para programação da UPDM;

- Capítulo 3 – São descritos os principais conceitos apresentados na litetura sobre todos os processos envolvidos para desenvolvido do projeto;

- Capítulo 4 – Baseado na pesquisa realizada, neste capítulo é apresentado uma visão geral da proposta e formalização de um contexto para desenvolvimento do projeto;
- Capítulo 5 – Neste capítulo é descrita a metodologia e os recursos utilizados para concretização do projeto;
- Capítulo 6 – O sistema foi dividido em fases para facilitar seu desenvolvimento. Neste capítulo são apresentados os principais resultados e discussões obtidos na execução do projeto.

Nos Apêndices são apresentados códigos fontes dos softwares desenvolvidos e exemplos de aplicações utilizadas para teste do sistema.

Capítulo 2

Unidade Processadora Discreta Microprogramável - UPDM

A UPDM é um protótipo de um processador, que segue a arquitetura monoprocessada de Von Neumman [Hay98] [Mal86] [Tan84] [Tau84] [Tok85] [Zuf78].

No desenvolvimento do protótipo objetivou-se obter uma arquitetura simples e de fácil entendimento, possibilitando o desenvolvimento de tecnologias mais avançadas.

Na Figura 2.1 apresenta-se o protótipo da UPDM.

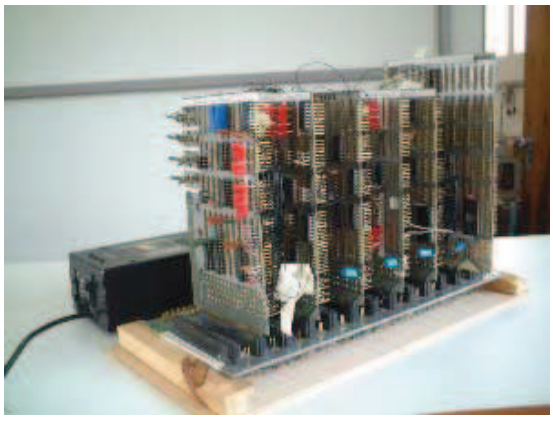


Figura 2.1: Unidade Processadora Discreta Microprogramável - UPDM

A UPDM é basicamente um processador de 8 bits possuindo as características descritas abaixo:

- barramento de dados de 8 bits;
- barramento de endereços de 8 bits;
- 8 páginas de memórias de 256 bytes;
- 8 entradas digitais e 1 entrada analógica (ADC 0804);
- 8 saídas digitais e 1 saída analógica (DAC 0801);
- clock automático - execução completa do programa;
- clock manual - permite a execução de instrução por instrução;
- velocidade de execução de 1kHz;
- a UPDM possui um conjunto 33 instruções assembly disponíveis na biblioteca do sistema e pode ter seu conteúdo ampliado pelo usuário;
- cada instrução pode possuir até 13 microinstruções.

Na Tabela 2.1 tem-se descritas as 33 instruções assembly disponíveis para programação da UPDM.

Tabela 2.1: Conjunto de Instruções Assembly da UPDM

n	Instrução	Código	Descrição
1	ADI <dado>	05	ACC← ACC + <dado>
2	ADD <endereço>	20	ACC← ACC + [endereço]
3	SUBI <dado>	06	ACC← ACC - <dado>
4	SUB <endereço>	21	ACC← ACC - [endereço]
5	CMP <dado>	19	compara ACC com <dado> e seta Flag de igual se ACC = dado
6	INC	07	ACC← ACC + 1
7	DCC	08	ACC← ACC - 1
8	LDI <dado>	04	ACC ←<dado>
9	LDA <endereço>	09	ACC ←[endereço]
10	STA <endereço>	10	[endereço]← ACC
11	AND <dado>	11	ACC← ACC AND <dado>
12	OR <dado>	12	ACC ← ACC OR <dado>
13	XOR<dado>	13	ACC ← ACC XOR <dado>
14	CLR	14	ACC ←0
15	NOT	15	ACC ← /ACC
16	JMP <endereço>	16	CP ← endereço
17	JZ <endereço>	82	se /ZERO = 0 CP ← endereço
18	JC <endereço>	91	se /VAI-UM = 0 CP ← endereço
19	JS <endereço>	A7	se SINAL = 1 CP ← endereço
20	JEQ <endereço>	E5	se IGUAL =1 CP ← endereço
21	JNZ <endereço>	B3	se /ZERO =1 CP ← endereço
22	JNC <endereço>	C1	se /VAI-UM = 1 CP ← endereço
23	JNS <endereço>	D6	se SINAL = 0 CP ← endereço
24	JNEQ <endereço>	F4	se IGUAL = 0 CP ← endereço
25	CALL endereço	17	[AP] ←CP; CP ← endereço; AP ←AP - 1
26	RET	18	CP ←[AP]; AP ← AP + 1
27	INA	22	ACC ← entrada analógica
28	IND	23	ACC ← entrada digital
29	OUTA	24	saída analógica ← ACC
30	OUTD	25	saída digital← ACC
31	HALT	26	Parada
32	PUSH	27	[AP] ←A CC
33	POP	28	ACC ← [AP]

Capítulo 3

Revisão de Literatura

"A maioria dos programadores que utilizam uma máquina de nível n está apenas interessada no nível do topo, que lembra o mínimo possível a linguagem de máquina situada embaixo. Entretanto, os interessados em compreender como um computador realmente funciona devem estudar todos os níveis. As pessoas ao projetarem novos computadores, ou novos níveis (máquinas virtuais) devem também estar familiarizadas com outros níveis além do nível do topo."[Tan84].

Todo programador deve ter em mente que sua aplicação percorrerá diversos níveis antes de ser executado. O conhecimento de como esses processos ocorrem evitam ao programador diversos problemas antes da execução da aplicação pela máquina.

O sistema integrado, que envolve compilação, montagem e simulação torna-se complexo devido às diversas áreas envolvidas, tais como arquitetura de computadores [JF90] [Tan84] [Mal86] [Tok85] [Tau84] [Zuf78], linguagens de computadores, teoria das linguagens, algoritmos, e engenharia de software.

A parte mais complexa do sistema integrado é o processo de compilação, isto é devido ao envolvimento não só das diversas áreas que um cientista da computação abrange, mas também o estudo da teoria da linguagem [Aho95].

Na literatura [Aho95], são abordados os diversos tópicos para construção de um compilador, além do estudo de casos de alguns compiladores, que utilizam técnicas apresentadas na literatura.

3.1 Compilador

O compilador é um programa que lê um programa escrito numa linguagem (a linguagem fonte) e o traduz num programa equivalente numa outra linguagem (a linguagem alvo) [Aho95] [SM89].

A Figura 3.1 expressa graficamente o processo do compilador.

Figura 3.1: Processo do Compilador

Existem variedades tanto de linguagens fontes como de linguagens alvos. Podemos citar como linguagens fontes: C [Sch90], C++ [DD98] [Str00], Fortran, Pascal, etc. As linguagens alvos podem ser uma outra linguagem de programação ou linguagem de máquina.

Durante o processo de compilação é importante se ter o relato da presença de erros no programa fonte.

Na Figura 3.2 é apresentada a estrutura geral de um compilador [Aho95].

Figura 3.2: Estrutura geral de um compilador

3.1.1 Análise Léxica

A análise léxica tem a função de reconhecer cada token (seqüência de caracteres) do programa fonte. Por exemplo, na análise léxica os caracteres no enunciado de atribuição

$$A = B + C * 60$$

poderiam ser agrupados nos tokens descritos na Tabela 3.1.

Tabela 3.1: Reconhecimento dos tokens na análise léxica

A, B, C	identificadores
=	símbolo de atribuição
+	sinal de adição
*	sinal de multiplicação
60	número

3.1.2 Gerador de Análise Léxica

O Flex [Uni95] é uma ferramenta para geração de scanners: programas que reconhecem padrões léxicos em um texto.

O Flex é geralmente usado da forma delineada na Figura 3.3. Primeiramente tem-se um arquivo com as especificações léxicas agrupados em um arquivo flex.l. Em seguida o arquivo é processado pelo aplicativo Flex, afim de produzir um programa em C, lex.yy.c. O programa lex.yy.c consiste em uma representação tabular de um diagrama de transições construído a partir das expressões regulares de flex.l, juntamente com uma rotina padrão que usa a tabela afim de reconhecer os lexemas. As ações associadas às expressões regulares em lex.l são trechos de códigos escritos em C que são carregados diretamente em lex.yy.c. O arquivo lex.yy.c é processado por um compilador C e é gerado um programa objeto a.out, que transforma um fluxo de caracteres de entrada numa seqüência de tokens.

Um arquivo Flex constitui-se de 3 partes separadas por %%, como demonstrado na Figura 3.4.

A seção de declarações incluem:

- as definições de macros como:

  digito [01]+ /* substituir {digito} por [01]+ ao processar as regras */

  frac [0-9]+ /* substituir {frac} por [0-9]+ ao processar as regras */

Figura 3.3: Gerando um analisador léxico com Flex

Figura 3.4: Formato de um arquivo Flex

nl \n /* substituir {nl} por \n ao processar as regras */

- a inclusão das linhas de comandos em C devem ser delimitadas por <%{> e <%}>, como:

  %{

  #include <y.tab.h>

  extern int yylval;

  %}

- a redefinição dos tamanhos das tabelas utilizadas pelo gerador, tais como número de nós da árvore sintática, número de transições e número de estados.

A seção de regras define a funcionalidade do analisador léxico. Cada regra compreende uma sequência válida de caracteres (utilizando literais e expressões regulares) e as ações semânticas associadas a ela.

As regras de definições são descritas na forma:

P1 {ação1}

P2 {ação2}

P3 {ação3}
P4 {ação4}
.....
Pn {açãon}

Onde cada Pi é uma expressão regular e cada açãoi é um fragmento de programa descrevendo que ação o analisador léxico deverá tomar quando o padrão Pi reconhecer um lexema. As ações são escritas em C.

A seção de procedimentos auxiliares permite complementar as ações.

Quando uma seqüência de caracteres de entrada casa com mais de uma regra, Flex adota uma das seguintes estratégias para resolver ambiguidades:

- escolher a regra que consegue casar a maior seqüência de caracteres possível;
- quando existe mais de uma regra que case com a maior seqüência de caracteres, será escolhida aquela que aparece em primeiro na seção de regras.

3.1.3 Geração de aplicação

Nesta seção ilustra-se a utilização de uma aplicação simples do Flex.

Na Figura 3.5 descreve-se o algoritmo de uma aplicação.

```
1   DIGIT [0-9]
2   %%
3   [1-9][DIGIT]*       printf("Dec");
4   0[0-7]*             printf("Oct");
5   0x[0-9A-Fa-f]+      printf("Hex");
6   <<EOF>>             return 0;
7   %%
8   int main(int argc, char *argv[]) {
9   FILE *f_in;
10
11  if (argc == 2) {
12          if (f_in = fopen(argv[1],"r"))
13                  yyin = f_in;
14          else
15                  perror(argv[0]);
16  }
17  else
18          yyin = stdin;
19
20  yylex();
21  return(0);
22  }
```

Figura 3.5: Algoritmo de uma aplicação simples utilizando Flex

Considere o exemplo com a especificação Flex apresentada na Figura 3.5 escrita em um arquivo denominado exemplo.l, onde .l é uma extensão padrão para esse tipo de arquivo. Para gerar o analisador léxico, Flex é invocado recebendo esse arquivo como entrada.

> flex exemplo.l

A execução desse comando gera um arquivo-fonte C denominado lex.yy.c, que implementa os procedimentos do analisador léxico. Para gerar o código executável, este programa deve ser compilado e ligado com a biblioteca libfl, que contém os procedimentos padrões internos do flex

> gcc lex.yy.c -lfl -o exemplo.exe

O arquivo executável exemplo.exe conterá o analisador léxico para inteiros sem sinal. Se aplicativo for invocado sem argumentos, exemplo irá aguardar a entrada do teclado para proceder à análise das strings; o término da execução será determinado pela entrada do caracter control-D. Se for invocado com um argumento na linha de comando, o aplicativo irá interpretar esse argumento como o nome de um arquivo que conterá o texto que deve ser analisado, processando-o do início ao fim.

Para a descrição do padrão, o Flex define uma linguagem para descrição de expressões regulares. Esta linguagem mantém a notação para expressões regulares, ou seja, a ocorrência de um caracter "a" indica a ocorrência deste se R é uma expressão regular, R- indica a ocorrência dessa expressão zero ou mais vezes; e se S também é uma expressão regular, então RS é a concatenação dessas expressões e R|S indica a ocorrência da expressão R ou da expressão S. Além dessas construções, a linguagem oferece ainda as extensões descritas na Tabela 3.2.

Tabela 3.2: Extensões para descrição de expressões regulares

.	qualquer caráter exceto \n
$[xyz]$	uma classe de caracteres, x ou y ou z
$[a-f]$	classe de caracteres, qualquer caráter entre 'a' e 'f'
$[\hat{}xyz]$	classe de caracteres negada, qualquer caracter exceto 'x' ou 'y' ou 'z'
R+	uma ou mais ocorrências da expressão regular R
R?	0 ou uma ocorrência da expressão regular R
R{4}	exatamente quatro ocorrências da expressão regular R
R{2,}	pelo menos duas ocorrências da expressão regular R
R{2,4}	entre duas e quatro ocorrências da expressão regular R
^R	a expressão regular R ocorrendo apenas no início de uma linha
R$	a expressão regular R ocorrendo apenas no final de uma linha
«EOF»	fim de arquivo

Um caracter especial pode ser escrito utilizando-se a construção \X; por exemplo, \. permite especificar a ocorrência de um ponto na expressão regular. Outra forma de indicar que uma string deve ser interpretada, é colocando-a entre aspas na regra.

3.1.4 Análise Sintática

A análise sintática envolve o agrupamento dos tokens formando frases gramaticais, dando origem a uma árvore sintática (Figura 3.6), que tem por base a gramática da linguagem fonte [SM89] e como resultado a estruturação.

Figura 3.6: Árvore gramatical para A = B+C*60

3.1.5 Análise Semântica

A terceira grande tarefa do compilador refere-se à tradução do programa fonte para o programa objeto. Em geral, a geração de código vem acompanhada, em muitas implementações, das atividades de análise semântica, que são responsáveis pela captação do significado do texto fonte. Esta operação é essencial à realização da tradução do texto fonte por parte das rotinas de geração de código.

Denomina-se semântica de uma sentença o significado por ela assumido dentro do contexto em que se encontra. Semântica de uma linguagem é a interpretação que se pode atribuir ao conjunto de todas as suas sentenças. Ao contrário da sintaxe, que é facilmente formalizável, a semântica exige para isso notações mais complexas. Assim, na maioria das linguagens de programação, a semântica tem sido especificada informalmente, geralmente através de linguagem natural.

As atividades de tradução, exercidas pelos compiladores, baseiam-se em uma perfeita compreensão da semântica da linguagem a ser compilada, uma vez que é disto que depende a criação das rotinas de geração de código, responsáveis pela obtenção do código objeto a partir do programa fonte.

A principal função da análise semântica é criar, a partir do texto fonte, uma interpretação deste texto fonte, expressa em alguma notação adequada - geralmente uma linguagem intermediária do compilador. Isto é feito com base nas informações das tabelas e nas saídas dos outros analisadores. Tipicamente, as ações semânticas englobam funções como:

- Criação e manutenção da tabela de símbolos: em geral, esta tarefa é executada pelo analisador léxico, mas, por razões de reaproveitamento dos programas de análise léxica - e mesmo sintática - as ações semânticas podem incorporar essa tarefa. A cada ocorrência de um identificador no texto fonte, a tabela de símbolos é consultada ou alterada. Sempre que um objeto da linguagem é declarado ou encontrado em um contexto, ele deve ser inserido na tabela de símbolos, se já não estiver presente. A criação e manutenção da tabela de símbolos é vital para o funcionamento do compilador, pois é nela que são guardados os nomes dos objetos definidos pelo programador e que são referenciados ao longo do programa. São, em geral, organizadas de modo que reflitam a estrutura do programa fonte, guardando, por exemplo, informações sobre o escopo em que os identificadores são definidos.

- Associar aos símbolos os stributos correspondentes: é necessário acrescentar, para cada identificador da tabela de símbolos, um conjunto de informações que seja suficiente para caracterizá-lo como sendo correspondente a um determinado objeto, indicando todas as características que tal objeto exige e que sejam de interesse para o processo de geração de código.

- Verificar a compatibilidade de tipos: a checagem de tipos é um recurso auxiliar para as atividades de geração de código, uma vez que o uso de dados cujos tipos sejam diferentes - embora coerentes - impõem ao gerador de código a tarefa de efetuar as conversões necessárias. Isto deve ser feito para permitir que as operações especificadas pelos comandos da linguagem sejam realizadas adequadamente.

3.1.6 Geração de Código Intermediário

A linguagem utilizada para a geração de um código em formato intermediário entre a linguagem de alto nível e a linguagem assembly deve representar, de forma independente do processador para o qual o programa será gerado, todas as expressões do programa original. Duas formas usuais para esse tipo de representação são a notação posfixa e o código de três endereços.

No código de três endereços cada endereço de memória atua como um registrador. Para o exemplo:

$$A = B + C * 60$$

pode ser expresso no código de três endereços como descrito na Figura3.7.

```
temp1 = 60
temp2 = C*temp1
temp3 = B+temp2
```

Figura 3.7: Código de três endereços gerado para A = B + C * 60

Na otimização de códigos, o código gerado pela fase de geração de código intermediário é melhorado, de tal forma que venha resultar num código de máquina mais rápido em tempo de execução [Aho95].

Otimizando o código da Figura 3.7 tem-se como resultado o código apresentado na Figura 3.8.

```
temp1 = C*60
A = B + temp1
```

Figura 3.8: Código de três endereços gerado para A = B + C * 60 melhorado

3.1.7 Geração de Código Objeto

A geração de código é a fase final do compilador, que consiste no código de montagem.

As instruções intermediárias são cada uma traduzida numa seqüência de instruções de máquina que realizam a mesma tarefa.

A tradução do código da Figura 3.8 poderia ser traduzido para o código mostrado na Figura 3.9.

```
MOV C, R2
MULT 60, R2
MOV B, R1
ADD R2, R1
MOV R1,A
```

Figura 3.9: Código objeto gerado da equação A = B + C * 60

Após a obtenção da representação simbólica para a linguagem de máquina, no caso temos como exemplo o código da Figura 3.9, gerado pela fase final do compilador, o tradutor passa a ser o montador.

3.2 Montador

A utilização da linguagem de montagem se deve a sua facilidade de programação, comparando-se com a utilização de códigos binários. Esta também é considerada a versão mnemônica do código de máquina.

O processo de montagem, representado graficamente pela Figura 3.10, pode ocorrer de diversas formas em diferentes máquinas. Devido a algumas semelhanças é possível descrevê-los em termos gerais.

Figura 3.10: Processo de montagem

Um dos processos de montagem é conhecido como "Montadores de dois passos". Este processo será brevemente descrito.

A principal função do passo um é construir uma tabela denominada tabela de símbolos, contento os valores de todos os símbolos [Tan84].

Juntamente com o início do passo um inicializa-se também um contador, que é incrementado a cada instrução processada. Assim é possível identificar juntamente

com o símbolo o seu endereço.

A função do passo dois é gerar o programa objeto e possivelmente imprimir a lista de montagem [Tan84].

Além da tradução dos códigos das instruções o montador também interpreta algumas diretivas de montagem (Tabela 3.3).

Tabela 3.3: Exemplos de diretivas de montagem

Diretiva	Desrição
EQU	Equivalente
ORG X	Origem em X
DS n	Reserva n bytes na memória
DB n	Inicializa posição de memória atual com o valor n
END	Fim do programa a ser montado

Os processos de compilação e montagem são programas tradutores que trabalham com níveis de linguagens diferentes. A linguagem associada ao compilador aproxima-se da linguagem do usuário e a linguagem associada ao montador aproxima-se a linguagem da máquina.

3.3 Simulador

O processo de simulação permite ao usuário verificar o comportamento do processador ao executar uma aplicação. Se o comportamento da aplicação não é o esperado, o usuário poderá corrigí-lo, sem a necessidade de apagá-lo da memória do processador.

Toda a estrutura interna do processador é exposta ao usuário de maneira que ele possa acompanhar a execução da aplicação.

Baseado na estrutura da UPDM, o usuário pode ter a visão dos seguintes itens:

- Contador de Programa (CP)
- Apontador de Pilha (AP)
- Acumulador (ACC)
- Entrada Digital (ED)
- Entrada Analógica (EA)
- Saída Digital (SD)

- Saída Analógica (SA)
- Flags de sinal, zero, igual e vai-um

Além do acompanhamento pelo o conteúdo dos registradores, é indicado durante o processo de execução a linha que está sendo processada.

Capítulo 4

Proposição

O sistema integrado é constituído pelos processos de compilação, montagem e simulação, e também pela UPDM descrevem perfeitamente o caminho que uma simples instrução, de soma por exemplo, percorrerá para ser executada. A importância desta visão, permite o desenvolvimento de novas técnicas em cada um dos processos. O programador que possui esta visão comete menos erros, que um programador que possui somente a visão do topo. Um programador com todos os processos em mente, observa pontos tanto da linguagem utilizada como da máquina.

O processo de compilação permite que uma linguagem de programação se aproxime mais da linguagem do usuário. E o processo montador aproxima-se do código da máquina.

Este projeto propõe atingir os seguintes objetivos:

- Estudo das técnicas de montagem de instruções de baixo nível para processador digital, e implementação de um montador para a UPDM.
- Desenvolvimento de um programa simulador da UPDM, para testes dos programas gerados pelo montador.
- Desenvolvimento de um compilador de linguagem de alto nível para a UPDM.

A implementação de cada objetivo gera o contexto do projeto descrito na Figura 4.1. Os módulos compilador, montador, simulador e UPDM são independentes. Isto permite ao programador desenvolver aplicações em alto nível ou em baixo nível.

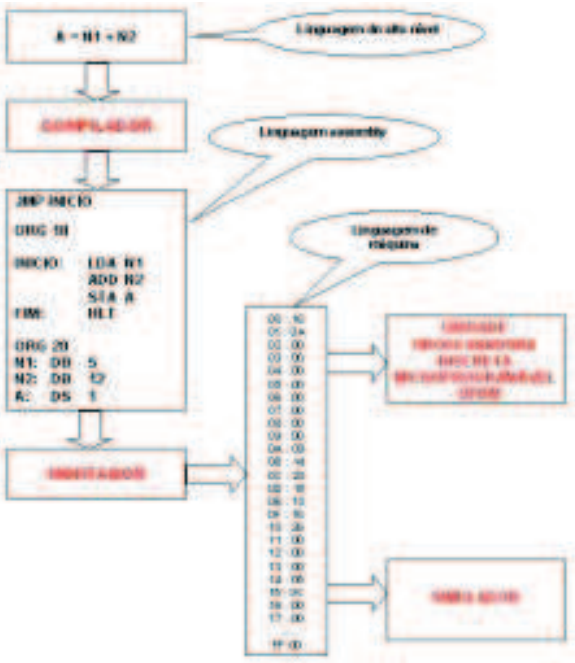


Figura 4.1: Contexto do projeto

O compilador e o montador são módulos de tradução que trabalham com níveis de linguagens diferentes. Quando o nível de tradução ocorre entre uma linguaguem de alto nível e uma linguaguem assembly, o tradutor denomina-se compilador. Quando trata-se da tradução entre linguaguem assembly e linguaguem de máquina o tradutor é o montador.

A linguaguem de máquina gerada pelo montador, é inserida diretamente na memória do processador, ou pode-se verificar seu comportamento utilizando-se o simulador, projetado para desenvolver as mesmas funções da UPDM.

Capítulo 5

Material e Método

Os objetivos do projeto foram alcançados utilizando os métodos descritos abaixo.

- O estudo das técnicas de algoritmos de montagem de instruções de baixo nível para processadores digitais.
- Confecção de um software montador para as instruções da UPDM.
- Testes do programa montador, utilizando-se de algoritmos implementados para UPDM.
- Confecção de um sofware simulador, para a verificação do comportamento da UPDM.
- Verificação da funcionalidade do programa simulador utilizando-se aplicações geradas para a UPDM.
- Estudo de técnicas para construção de um compilador.
- Confecção de um compilador, baseado na estrutura da linguagem C.
- Gerar um ambiente gráfico para acesso aos recursos do sistema integrado.

Para o desenvolvimento dos programas compilador, montador e simulador utilizou-se dos recursos da programação orientada a objetos e da linguagem C++ [DD98] [Sch90] [Str00] .

O compilador utilizado denomina-se mingw32 desenvolvido para o ambiente Windows [Kha00].

Para desenvolvimento da parte gráfica do sistema integrado, utilizou-se os recursos do compilador C++Builder 3.0 [Hay98].

Todas as aplicações geradas foram simuladas e executadas pela UDPM.

Capítulo 6

Resultados e Discussões

O projeto foi dividido em 3 fases:

1. Desenvolvimento do Programa Montador;
2. Desenvolvimento do Programa Simulador;
3. Desenvolvimento do Programa de Compilação.

Cada fase é agrupada à interface gráfica do sistema integrado para programação da UPDM.

A interface gráfica gerada facilita o acesso aos recursos do sistema integrado.

Ao ativar o recurso gráfico é apresentada a tela principal (Figura 6.1) a partir da qual as opções de edição, compilação, montagem, simulação e ajuda podem ser processadas.

O código fonte da interface gráfica gerada para o sistema integrado encontra-se no Apêndice A.

6.1 Programa Montador

A opção "Editor - Montador", da tela principal disponibiliza para o usuário um editor de texto, onde este poderá editar um programa fonte, utilizando a linguagem assembly da UPDM.

O arquivo gerado terá a extensão .asm.

Para ativar o processo de montagem é necessário entrar com o nome arquivo fonte, que possui a extensão .asm. Ao ativar a opção "Montador" é exibida uma caixa de diálogo (Figura 6.2), onde deve-se entrar com o nome arquivo fonte.

Figura 6.1: Tela principal do Sistema Integrado

O processo de montagem segue o fluxograma apresentado nas Figuras 6.3, 6.4 e 6.5. Os fluxogramas descrevem suscintamente o processo montador.

Durante o processo de montagem são gerados 5 arquivos, descritos abaixo:

- <arquivo>.hex - constitui-se do código de máquina;
- <arquivo>.lst - constitui-se do código fonte juntamente com o código de máquina;
- <arquivo>.map - constitui-se da tabela de símbolos;
- <arquivo>.log - constitui-se dos erros e sucessos gerados pela montagem do código fonte;
- <arquivo>.sml - constitui-se de um arquivo auxiliar para o processo de simulação.

O código do processo de montagem encontra-se no Apêndice B.

Figura 6.2: Caixa de diálogo para entrada do arquivo montador

No Apêndice C encontram-se ilustrados alguns exemplos de aplicações, desenvolvidos em linguagem assembly para a UPDM. Juntamente com aplicação é ilustrado o código de máquina (arquivo.hex) gerado pelo Programa Montador.

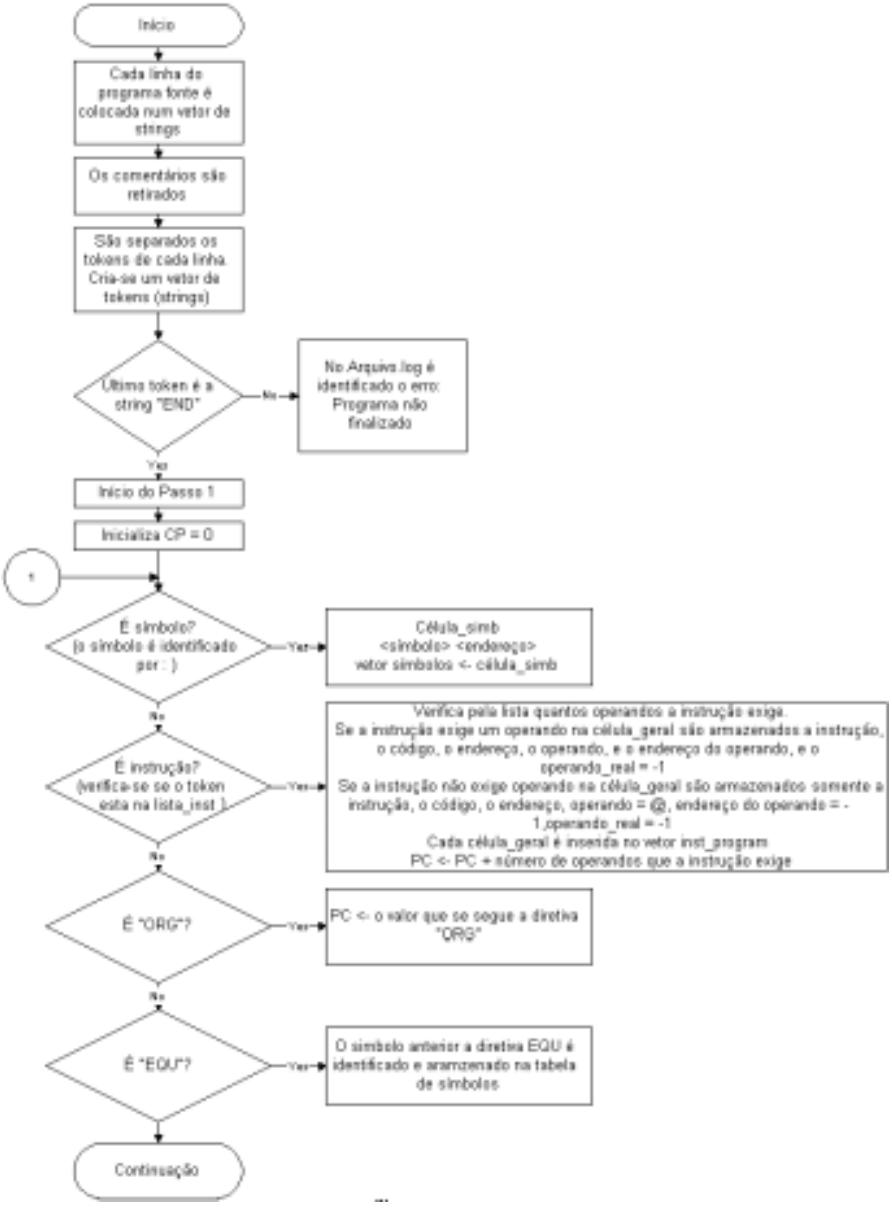


Figura 6.3: Processo de montagem - Fase 1

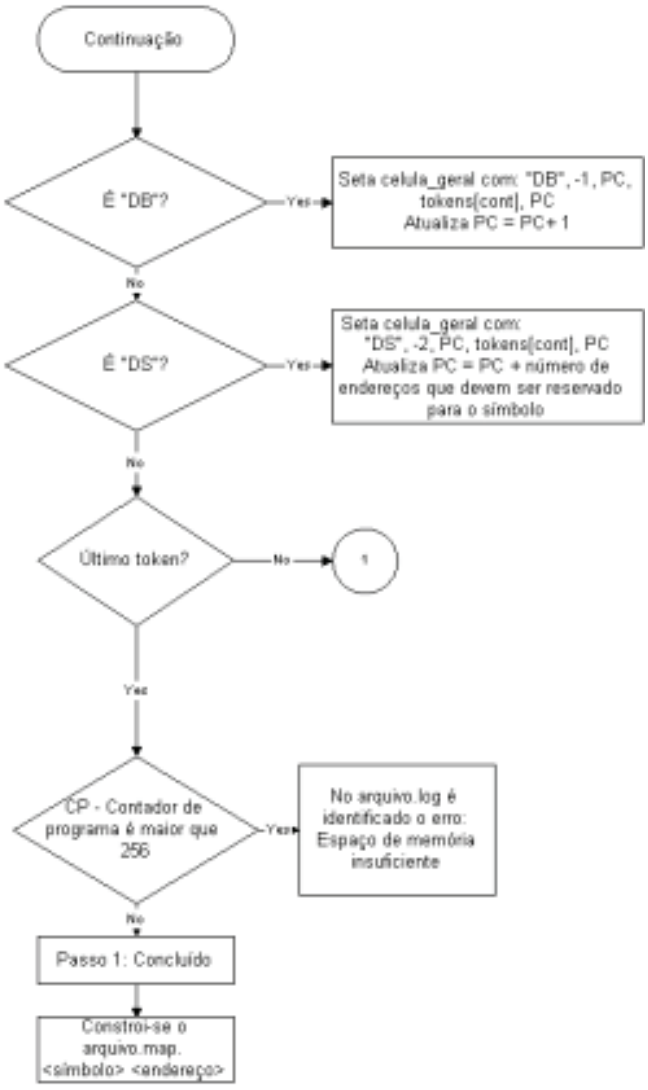


Figura 6.4: Processo de montagem - Fase 2

Figura 6.5: Processo de montagem - Fase 3

6.2 Programa Simulador

Para execução do processo de simulação, inicialmente indica-se o nome do arquivo com extensão .hex gerado pelo proceso de montagem. Logo que o usuário selecionar a opção "Simulador" será apresentada uma caixa de diálogo (Figura6.6) onde, deve-se entrar com o nome do arquivo.

Figura 6.6: Caixa de diálogo para entrada do arquivo.hex

Confirmada a existência do arquivo é apresentada a tela principal do processo de simulação (Figura 6.7).

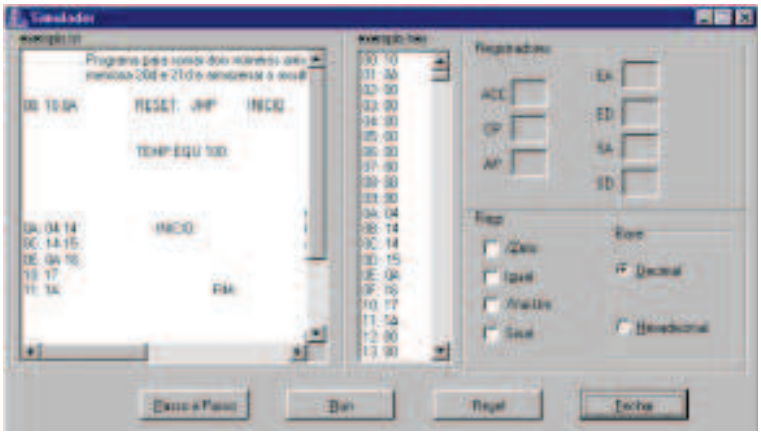


Figura 6.7: Tela principal do processo simulador

O principal passo executado pelo processo de simulação é apresentado em forma de fluxograma na Figura 6.8.

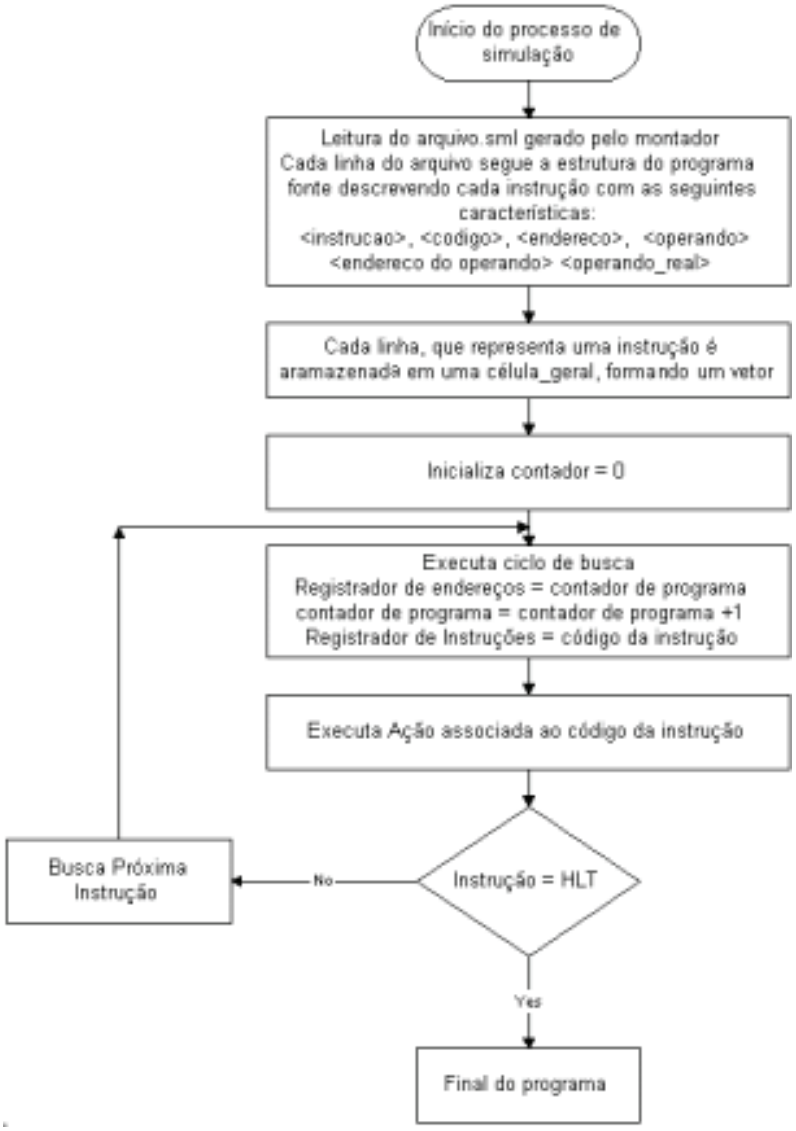


Figura 6.8: Principal rotina do processo de simulação

O processo de simulação permite executar o código fonte de duas maneiras:

- execução automática - executando-se a opção "Run";
- execução passo a passo - executando-se a opção "Passo a Passo".

O processo de simulação permite editar um valor para as entradas digitais e analógicas. Isto é executado quando as instruções IND ou INA são encontradas no código fonte. Para entrada do valor desejado é apresentada ao usuário uma caixa de diálogo (Figura 6.9). O limite apresentado para os valores da entradas analógica (0 à 5 Volts) é devido ao conversor analógico-digital utilizado pela UPDM. E devido a utilização de um conversor digital-analógico de 8 bits, a entrada digital é limitada em 0 a 255.

Figura 6.9: Caixas de diálogo para entrada de dados diagital e analógico

A cada passo de execução é indicada no arquivo.hex a posição da memória que está sendo lida e executada.

O código do programa simulador encontra-se no Apêndice D.

6.3 Programa Compilador

O processo de compilação dividiu-se no desenvolvimento do analisador léxico, do analisador sintático, do analisador semântico , do gerador de código intermediário e do gerador de código objeto.

6.3.1 Analisador Léxico

Devido à grande complexidade da análise léxica utilizou-se a ferramenta FLEX++, que automatiza o processo de criação do autômatos e o processo de reconhecimento de sentenças regulares a partir da especificação das expressões regulares correspondentes.

As especificações léxicas estão descritas nos parágrafos seguintes.

Cada átomo tem pelo menos um atributo chamado tipo.

Conforme o tipo do átomo, ele poderá ter outros tipos de atributos.

Se o átomo é uma palavra reservada, seu tipo é a própria palavra reservada.

Na Tabela 6.1 estão descritas as palavras reservadas reconhecidas pelo analisaor léxico.

Tabela 6.1: Palavras reservadas reconhecidas pelo analisador léxico

MAIN
PRINTF
SCANF
INT
CHAR
DO
IF
ELSE
WHILE
FOR

Se o átomo for um identificador, seu tipo é ID e o atributo é a cadeia de caracteres.

Constantes inteiras terão seu valor limitado em 1byte.

Cadeia de caracteres virão entre aspas duplas e terão seu tipo classificado como STRING LITERAL.

Caracteres virão entre aspas simples.

Na Tabela 6.2 são descritos os operadores reconhecidos, seus tipos e atributos.

Tabela 6.2: Reconhecimento dos operadores pelo analisador léxico

Átomo	Tipo	Atributo
+	OPAD	ADIÇÃO
-	OPAD	SUBTRAÇÃO
or	OPAD	OR
*	OPMULT	MULT
/	OPMULT	DIVISÃO
and	OPMULT	AND
~	OPNEG	NEG
<	OPREL	MENOR
<=	OPREL	MENORIGUAL
>	OPREL	MAIOR
>=	OPREL	MAIORIGUAL
==	OPREL	IGUAL
!=	OPREL	DIFER

- OPAD: operadores aditivos
- OPMULT: operadores multiplicativos
- OPNEG: operadores negativos
- OPREL: operadores relacionais

Os comentários são colocados entre * e *\.

Desenvolveu-se o analisador léxico utilizando o aplicativo FLEX++, devido à facilidade de se criar um scanner para a classificação dos tokens.

O arquivo analise_léxica.l, no qual encontram-se descritas as regras de tradução encontra-se no Apêndice E.

Para geração da aplicação analiselexica.exe executou-se os comandos descritos abaixo, já mencionado durante a introdução.

- >flex++ analise_léxica.l
- >g++ lex.yy.c -lfl -o analiselexica.exe

A execução do arquivo executável gera um arquivo tabelalexica.txt com a classificação de cada token reconhecido.

Além da classificação dos tokens é gerado também um arquivo erros.txt, onde são relatados os erros que ocorrem durante o processamento do arquivo de entrada.

Na Tabela 6.3 são apresentados os possíveis erros que podem ocorrer durante a análise léxica. O arquivo erros.txt é complementado durante a análise sintática.

Tabela 6.3: Possíveis erros léxicos

Erros	Descrição
<número> muito extenso	A especificação <número> deve estar entre 0 e 255
<id> identificador inválido	Identificadores que iniciam-se com números são inválidos
<caracter> caracter inválido	Caracteres maiores que 1 byte são inválidos
Arquivo terminado pelo comentário	Comentário inicializado, mas não finalizado

6.3.2 Analisador Sintático

O analisador sintático segue a gramática descrita na Figura 6.10. A gramática é livre de ambiguidades e de recursão à esquerda.

```
<prog> → MAIN "(" ")" "{" <corpo>"}"
<corpo> → <decl_list> <inst>
<decl_list> → <tipo> ID ";" | <decl_list>
<tipo> → INT | CHAR

<inst> → <atribuição> |
          IF "(" <exp_cond> ")" "{" <inst> "}" |
          IF " (" <exp_cond> ")" "{" <inst> "}" ELSE "{" <inst> "}" |
          DO "{" <inst> "}" WHILE " (" <exp_cond> ")" |
          WHILE " (" <exp_cond> ")"  "{" <inst> "}" |
          FOR " (" <exp_inicial> ";"  <exp_cond> ";"  <exp_inc> ")"

<exp> → <exp_simp>
<exp_cond>→ <exp_simp> | <exp_simp> OPREL <exp_simp>
<exp_simp> → <term> | <term> OPAD <exp_simp>
<term> → <fator> | <fator> OPMULT <term>
<fator> → ID | CTE | CARACTER | "(" <exp_simp>")" | OPNEG <fator>

<atribuição> → ID "=" <exp_simp>
```

Figura 6.10: Gramática seguida pelo analisador sintático

Inicialmente implementou-se somente duas instruções: as estruturas do IF e da atribuição.

A gramática será submetida a um analisador sintático preditor, onde cada átomo consegue decidir por qual produção deve seguir.

O analisador sintático baseou-se nos diagramas de transição de estados para cada não terminal.

Nas figuras a seguir serão apresentados os diagramas correspondentes a cada não terminal da gramática.

Para a implementação do diagrama de transição de estados, gerou-se um grafo orientado, onde os vértices representam os estados e sua informação representa o número do estado e os arcos representam as transições de estados. Os arcos carregam informações do vértice de destino, a condição para transição e ações que devem ser executadas. Na Figura 6.11 o arco é representado graficamente.

Primeiramente, na Figura 6.12 tem-se o autômoto principal, e na Figura 6.13 tem-se o diagrama de transição principal do analisador sintático.

Figura 6.11: Estrutura para o arco do diagrama de transição de estados

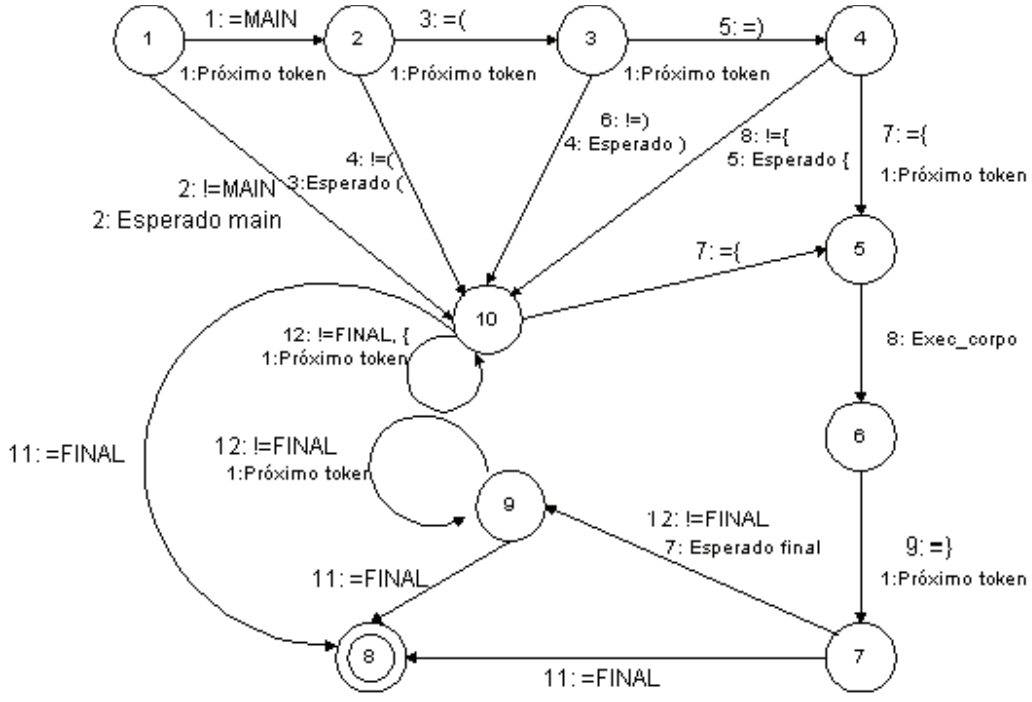


Figura 6.12: Autômato principal do analisador sintático

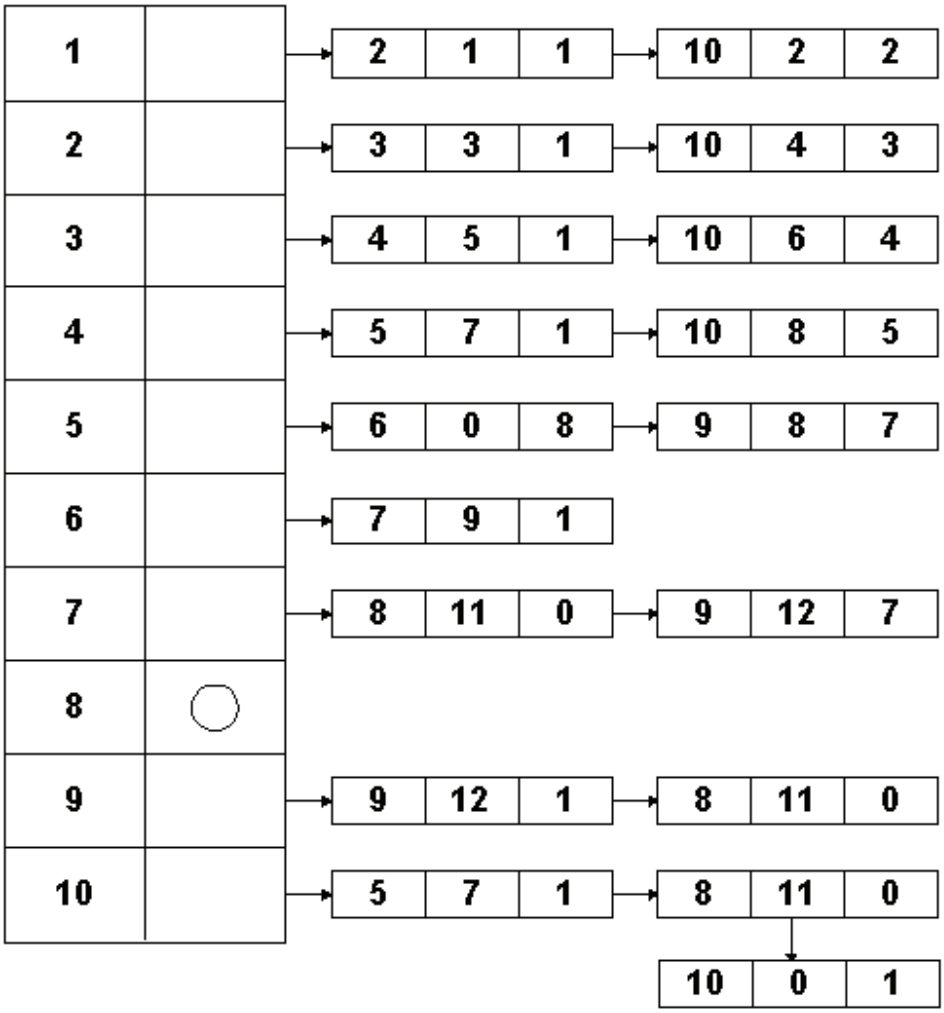


Figura 6.13: Diagrama de transição principal do analisador sintático

Nos passos seguintes apresenta-se em cada figura o autômoto e seu diagrama de transição, que corresponde à um não terminal da gramática e à uma ação na execução do analisador.

A Figura 6.14 corresponde à execução do não terminal <corpo>, que corresponde a ação 8, como apresentado no diagrama da Figura 6.13.

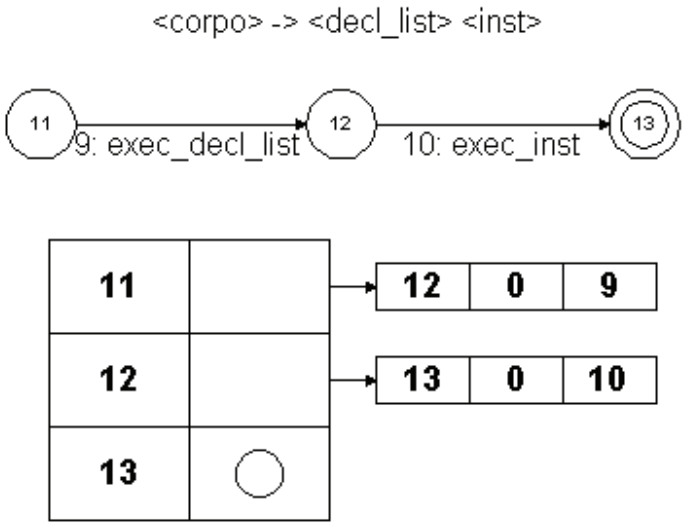


Figura 6.14: Ação 8: <corpo>

Na Figura 6.15 tem-se a ação 9, que corresponde ao não terminal <decl_list>.

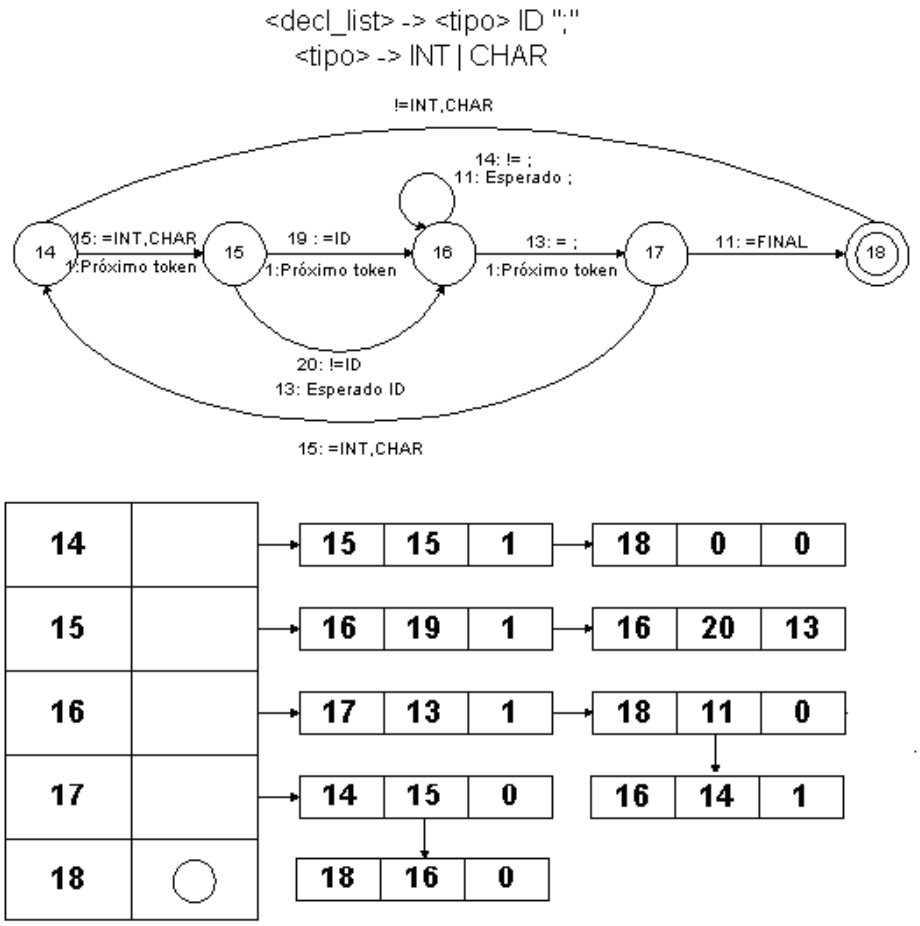


Figura 6.15: Ação 9: <decl_list>

Na Figura 6.16 tem-se a ação 10, que corresponde ao não terminal <inst>.

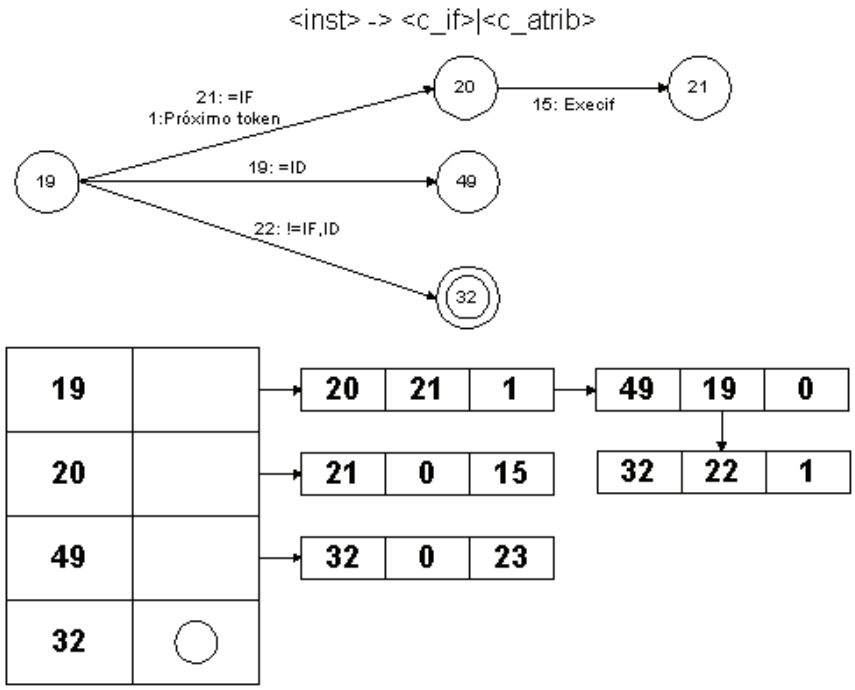
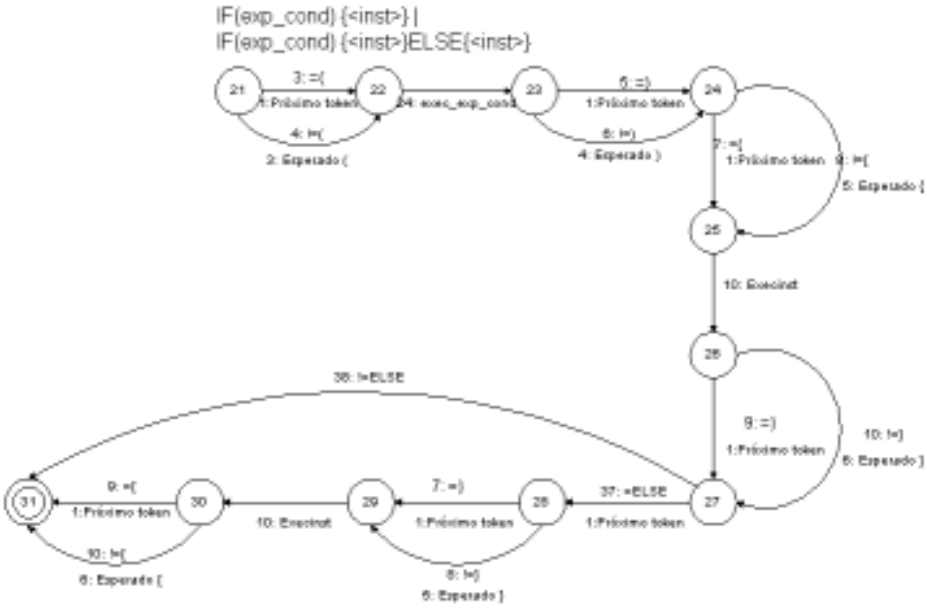


Figura 6.16: Ação 10: <inst>

Na Figura 6.17 tem-se a ação 15, que corresponde a execução da estrutura da instrução IF.



21		22	3	1	22	4	3
22		23	0	24			
23		24	5	1	24	6	4
24		25	7	1	25	8	5
25		26	49	10	26	22	0
26		27	9	1	27	10	6
27		28	37	1	31	38	0
28		29	7	1	29	8	5
29		30	49	10	30	22	0
30		31	9	1	31	10	0
31	○						

Figura 6.17: Ação 15: <if>

Na Figura 6.18 tem-se a ação 23, que corresponde à execução da estrutura da instrução de atribuição.

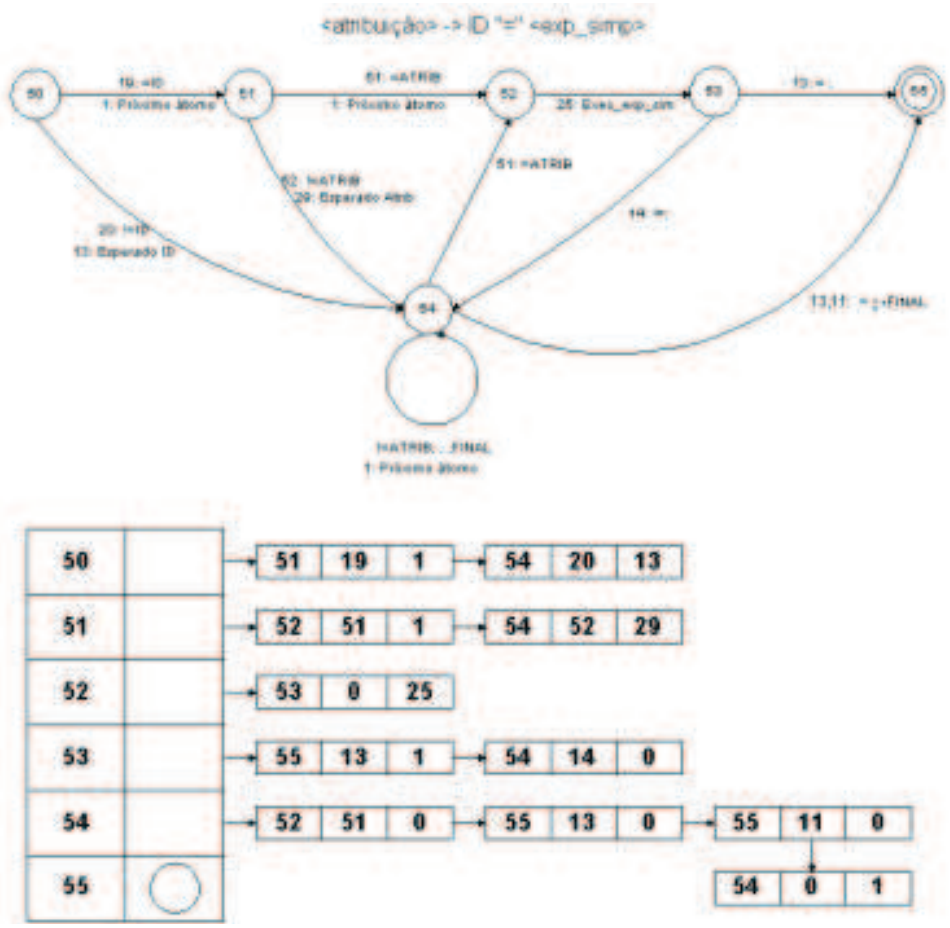


Figura 6.18: Ação 23: <atribuicao>

Na Figura 6.19 tem-se a ação 24, que corresponde à execução de uma expressão relacional.

Na Figura 6.20 tem-se a ação 25, que corresponde à execução de uma expressão aritmética aditiva.

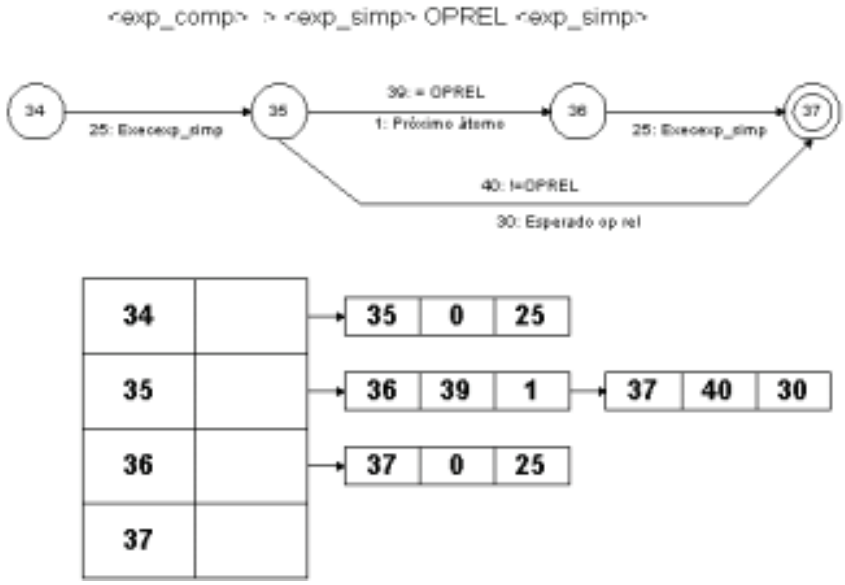


Figura 6.19: Ação 24: <exp_comp>

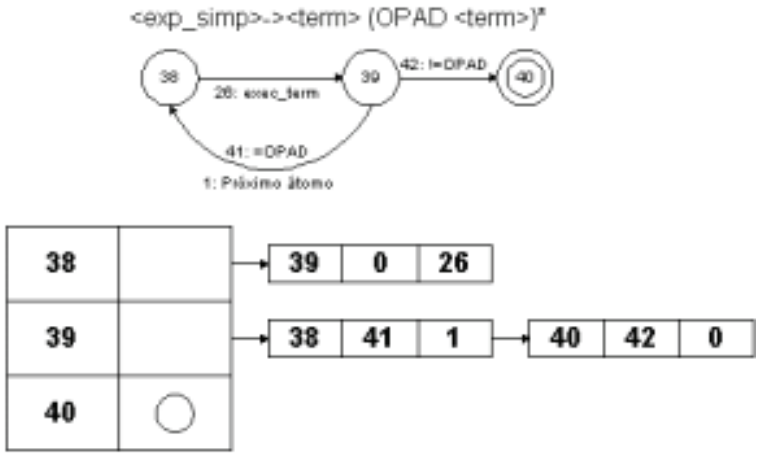


Figura 6.20: Ação 25: <exp_simp>

Na Figura 6.21 tem-se a ação 26, que corresponde à execução de uma expressão multiplicativa.

Na Figura 6.22 tem-se a ação 27, que leva a identificação dos terminais.

Para implementar o diagrama de transição de estados do analisador sintático gerou-se um objeto grafo apresentado no Apendice F.

O código do analisador sintático é apresentado no Apêndice G.

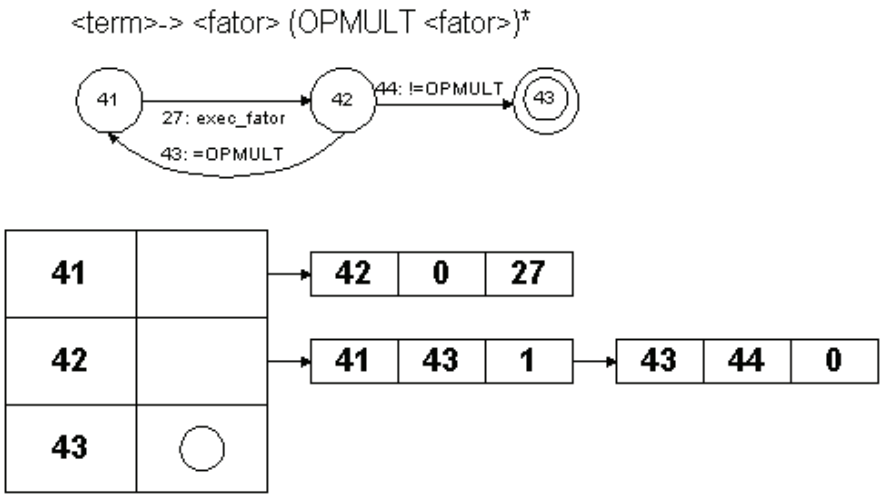


Figura 6.21: Ação 26: <term>

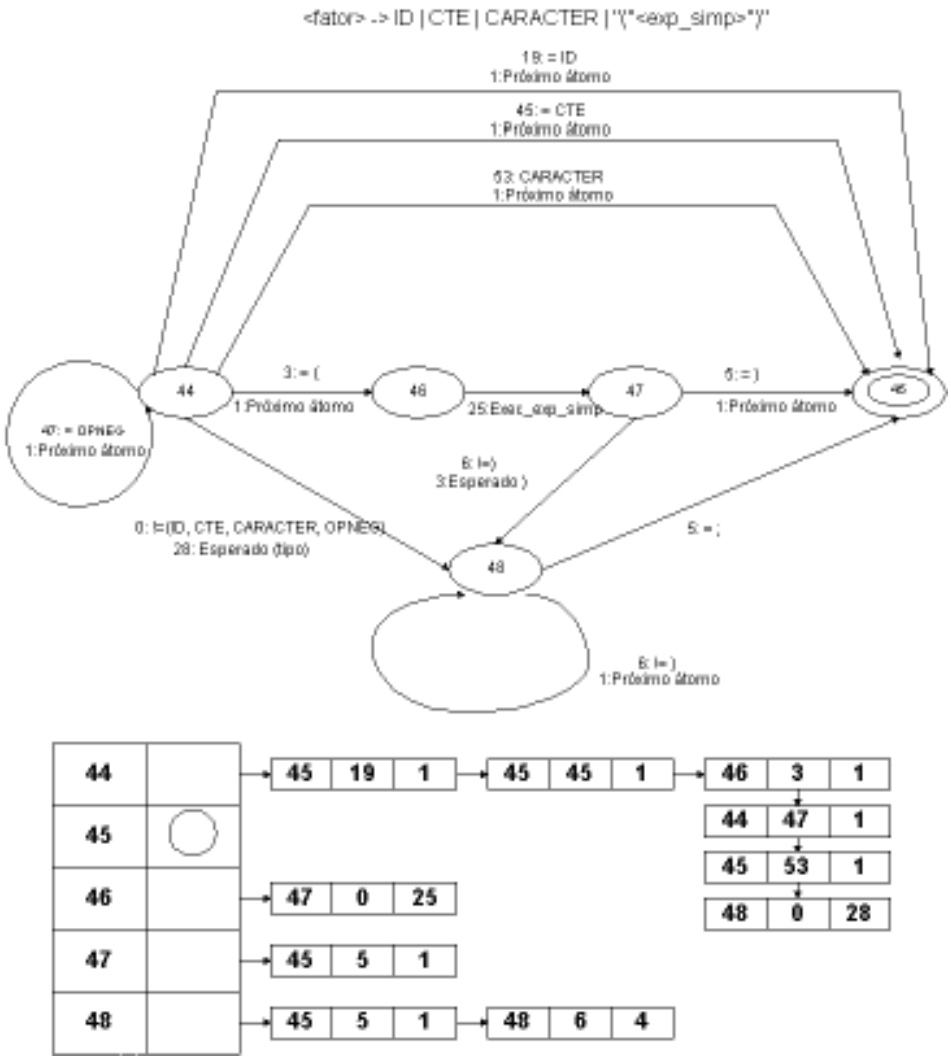


Figura 6.22: Ação 27: <fator>

6.3.3 Analisador Semântico

As ações semânticas implementadas estão relacionadas a criação e à manutenção da tabela de símbolos.

A tabela de símbolos é criada durante o processo de análise sintática, mais especificamente durante a declaração das variáveis.

Após sua criação a tabela é constantemente consultada, para verificação da existência de uma determinada variável e para verificação do seu tipo.

Dois erros semânticos são reportados durante o processamento do código fonte:

- Durante o procesamento do código fonte é verificado se todas as variáveis foram declaradas. Caso esta ação não seja verdadeira é emitido a mensagem: "Variável <nome da variável> não declarada!!"
- É verificado numa expressão se todos as variáveis são do mesmo tipo. Caso esta ação não seja verdadeira é emitida a mensagem: "Dados incompatíveis".

6.3.4 Gerador de código intermediário

Para gerar código intermediário a partir da árvore sintática obtida no processo de análise deve-se caminhar a árvore em profundidade (depth first), gerando código em quádruplas (ou triplas se for o caso) para cada uma das construções que aparecem na árvore.

Durante este caminhamento serão geradas variáveis temporárias necessárias para o cálculo das expressões que aparecem na árvore. Por exemplo:

$$A = B + C * D$$

Tem como árvore sintática a Figura 6.23.

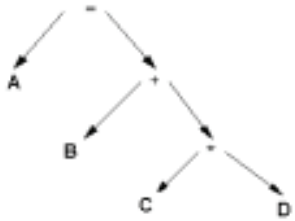


Figura 6.23: Árvore sintática para expressão A = B + C * D

Efetuando o caminhamento na árvore, o primeiro nodo que só contém folhas como filhos é o do operador "*". Neste caso é gerado o primeiro comando de código intermediário e uma variável temporária (temp1) como ilustrado na Tabela6.4.

Tabela 6.4: Primeira quádrupla da expressão A = B + C * D

Operação	Operando1	Operando2	Resultado
*	C	D	temp1

O nodo rotulado como "+" é visitado, gerando o código descrito na Tabela6.5.

Tabela 6.5: Quádruplas da expressão A = B + C * D

Operação	Operando1	Operando2	Resultado
*	C	D	temp1
+	B	temp1	temp2

Finalmente, o nodo rotulado como "=" gera o código da Tabela 6.6.

Tabela 6.6: Quádruplas da expressão A = B + C * D

Operação	Operando1	Operando2	Resultado
*	C	D	temp1
+	B	temp1	temp2
=	temp2		A

6.3.5 Regras semânticas para a Geração de Código Intermediário

As regras semânticas descritas na Tabela 6.7 definem a geração de código intermediário.

Tabela 6.7: Regras semânticas para geração de código intermediário

Produção	Regra Semântica
S → id := E	S.código := E.código | quádrupla(´:=´, E.lugar, ´´, id.lugar)
E → E1 + E2	E.lugar := novotemp; E.código := E1.código | E2.código | quádrupla(´+´, E1.lugar, E2.lugar, E.lugar)
E → E1 * E2	E.lugar := novotemp; E.código := E1.código | E2.código | quádrupla(´*´, E1.lugar, E2.lugar, E.lugar)
E →(E1)	E.lugar := E1.lugar E.código := E1.código
E → id	E.lugar := id.lugar E.código := ´´

6.3.6 Gerador de código objeto

A fase final do processo de compilação é a geração de código. Ela recebe a representação intermediária do programa fonte e as informações armazenadas na tabela de símbolos e produz como saída um programa objeto equivalente. As informações da tabela de símbolos são usadas para determinar os endereços de execução dos objetos de dados do programa.

Assume-se que, do ponto de vista do gerador de código, o programa já passou pela fase de análise e que já foi traduzido para uma representação intermediária. Portanto, o processo de geração de código pode prosseguir, assumindo que a sua entrada é isenta de erros.

Os requisitos impostos ao gerador de código são severos. A saída deve ser isenta de erros e de alta qualidade, assim como o próprio gerador de código deve executar de forma eficiente.

A saída do gerador de código é o programa objeto que pode assumir variadas formas: linguagem de máquina absoluta, linguagem de máquina relocável e linguagem assembly.

A produção de um código em linguagem assembly como saída do gerador de código faz com que o processo de geração de código seja muito mais simples.

6.3.7 Um gerador de código simples

O algoritmo de geração de código apresentado tem como entrada um conjunto de instruções em código intermediário constituindo um bloco básico. Para cada comando do código intermediário da forma x = y op z, devem ser aplicadas as seguintes ações descritas nos items abaixo.

1. Invocar a função getreg que determina a localização L onde o resultado da computação de y op z será armazenado. L pode ser uma posição de memória ou um registrador.

2. Consultar o descritor de endereços para determinar a localização de y e gerar a instrução MOV y,L.

3. Gerar a instrução op z, L, obtendo antes a localização de z. Atualizar os descritores para indicar que o novo valor de x encontra-se em L.

4. Se os valores correntes de y e/ou z não tiverem uso posteriormente no bloco e estiverem armazenados em registradores, liberar estes registradores para uso.

Devido à complexidade do sistema a geração de código intermediário e a geração de código objeto ainda não foram finalizadas.

Nos itens anteriores foram descritos algumas técnicas simplificadas, que poderão ser utilizadas para a fase de síntese do compilador.

Capítulo 7

Conclusões

O projeto iniciou-se com a construção do processador, que envolveu várias pesquisas para desenvolvimento de um processador com arquitetura simples, mas que atendesse as necessidades didáticas do projeto. Além da pesquisa, outros processos importantes como: desenvolvimento dos circuitos eletrônicos, simulações, testes e montagem do protótipo fizeram parte deste processo inicial. Finalizada a parte de hardware desenvolveram-se softwares para programação do processador.

Por meio do montador e do simulador o usuário pode gerar várias aplicações para a UDPM e verificar a execução da programação em assembly. Ao gerar uma aplicação o usuário poderá utilizar o simulador, para verificar a sua funcionalidade ou então gravar diretamente na memória da UDPM para execução.

Durante o desenvolvimento do processo de compilação, observou-se a grande complexidade das fases de construção de um compilador. As diversas variáveis que surgem no tratamento da gramática, dificultaram o desenvolvimento do processo de compilação. A fase de análise, constituída pela análise léxica, sintática e semântica foram completadas, mas a fase de síntese, constituída pela geração de código intermediário e geração de código ainda devem ser finalizadas.

Unindo-se o desenvolvimento da UPDM e do sistema integrado teremos um sistema, através do qual consegue-se descrever toda a concepção da tecnologia empregada no desenvolvimento de microprocessadores. Desde a etapa de projeto do hardware até a fase de desenvolvimento dos softwares.

O sistema integrado, constituído por um compilador, montador e simulador permite ao programador optar por desenvolver uma aplicação em alto nível ou em baixo nível para a UPDM.

Todos os processos envolvidos dão uma visão ampla de como ocorre o pro-

cessamento da informação pelo processador. O programador que possui esta visão desenvolve aplicações, aproveitando-se dos recursos oferecidos pela linguagem e pelo processador.

Referências Bibliográficas

[Aho95] Alfred V. Aho. *Compiladores: Princípios, Técnicas e Ferramentas*. Rio de Janeiro: Editora JC, 1995. 343p.

[DD98] H. M. Deitel and P. J. Deitel. *C++ How to Program*. Upper Sandle Rive, New Jersey: Prentice Hall, 1998. 1130p.

[Hay98] John P. Hayes. *Computer Architecture and Organization*. São Paulo: McGraw-Hill, 1998.

[JF90] G. G. Langdon Jr and E. Fregni. *Projeto de Computadores Digitais*. São Paulo: Edgard Blucher Ltda, 1990. 357p.

[Kha00] Munit Khan. Mingw: Minimalist gnu for windows, 2000. www.mingw.org.

[KL00] Edna M. Kanazawa and Wilian S. Lacerda. Descrição de uma unidade processadora discreta microprogramável. *Info Comp*, 2, Novembro 2000.

[Mal86] Albert Paul Malvino. *Microcomputadores e Microprocessadores*. São Paulo: McGraw-Hill, 1986. 577p.

[Sch90] H. Schildt. *C, Completo Total*. São Paulo: Makron Books, 1990. 889p.

[SM89] Valdemar W. Setzer and Inês S. H. Melo. *A construção de um Compilador*. Rio de Janeiro: Campus, 1989. 175p.

[Str00] Bjarne Stroustrup. *C++ a Linguagem de Programação*. Porto Alegre: Bookman, 2000. 823p.

[Tan84] A. S. Tanembaum. *Organização Estruturada de Computadores*. São Paulo: Prentice Hall do Brasil, 1984. 460p.

[Tau84] Hebert Taub. *Circuitos Digitais e Microprocessadores*. São Paulo: McGraw-Hill, 1984.

[Tok85] Roger L. Tokheim. *Introdução aos Microprocessadores*. São Paulo: McGraw-Hill, 1985. 431p.

[Uni95] University of California at Berkeley. *Flex, version 2.5 - A fast scanner generator*, 2.5 edition, march 1995.

[Zuf78] Jõao Antônio Zuffo. *Fundamentos da Arquitetura e Organização dos Microprocessadores*. São Paulo: Edgard Blucher Ltda, 1978. 419p.

Apêndice A

Interface gráfica gerada para o Sistema Integrado

A.1 Código fonte da interface gráfica gerada para o Sistema Integrado

A interface gráfica foi desenvolvida utilizando-se os recursos do compilador C++ Builder 3.0.

Projeto Principal - EditorSAP.cpp

```
// Projeto : Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//           Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("EditorSAP.res");
USEFORM("U_Principal.cpp", FormPrincipal);
USEFORM("U_Montador.cpp", FormMontador);
USEFORM("U_Simulador.cpp", FormSimulador);
USEFORM("ModuloPrincipal.cpp", FormTelaPrincipal);
USEFORM("U_DialogoASM.cpp", FormDialogoASM);
USEFORM("U_DialogoHex.cpp", FormDialogoSimulador);
USEFORM("U_Help.cpp", FormHelp);
```

```
USEFORM("U_Sobre.cpp", FormSobre);
USEFORM("U_EntradaDigital.cpp", FormEntradaDigital);
USEFORM("U_EntradaAnalogica.cpp", FormEntradaAnalogica);
USEFORM("U_EditorCompilador.cpp", FormEditorCompilador);
USEFORM("U_DialogoC.cpp", FormDialogoCompilador);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->Title = "SAP";
        Application->CreateForm(__classid(TFormTelaPrincipal), &FormTelaPrincipal);
        Application->CreateForm(__classid(TFormPrincipal), &FormPrincipal);
        Application->CreateForm(__classid(TFormMontador), &FormMontador);
        Application->CreateForm(__classid(TFormSimulador), &FormSimulador);
        Application->CreateForm(__classid(TFormHelp), &FormHelp);
        Application->CreateForm(__classid(TFormSobre), &FormSobre);
        Application->CreateForm(__classid(TFormEditorCompilador), &FormEditorCompilador);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------------
```

A.1.1 Tela Principal

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "ModuloPrincipal.h"
#include "U_DialogoASM.h"
```

```
#include "U_Principal.h"
#include "U_DialogoHex.h"
#include "U_Help.h"
#include "U_EditorCompilador.h"
#include "U_DialogoC.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormTelaPrincipal *FormTelaPrincipal;
//---------------------------------------------------------------------------
__fastcall TFormTelaPrincipal::TFormTelaPrincipal(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormTelaPrincipal::ButtonMontadorClick(TObject *Sender)
// se o botão montador for clicado abre-se o formulário de diálogo, onde deve ser
// informado o nome e o caminho do arquivo.asm
// o formulário de diálogo não é aberto em tempo de execução.
{
  Application->CreateForm(__classid(TFormDialogoASM), &FormDialogoASM);
  FormDialogoASM->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonFecharClick(TObject *Sender)
// botão fechar finaliza a aplicação
{
  Application->Terminate();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonEditorMontadorClick(
      TObject *Sender)
// se o botão Montador-Editor for clicado será aberto o editor do montador
// os arquivos do editor terão extensão .asm
{
  FormPrincipal->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonSimuladorClick(TObject *Sender)
```

```
// se o botão simulador for clicado o formulário de diálogo será aberto e o arquivo
//. hex deve ser informado.
{
  Application->CreateForm(__classid(TFormDialogoSimulador), &FormDialogoSimulador);
  FormDialogoSimulador->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonHelpClick(TObject *Sender)
{
// se o botão help for clicado o formulário de help será aberto
    FormHelp->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonEditorCompiladorClick(
      TObject *Sender)
{
FormEditorCompilador->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TFormTelaPrincipal::ButtonCompiladorClick(TObject *Sender)
{
  Application->CreateForm(__classid(TFormDialogoCompilador), &FormDialogoCompilador);
  FormDialogoCompilador->ShowModal();
}
//---------------------------------------------------------------------------
```

A.1.2 Editor Montador

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_Principal.h"
#include "U_Montador.h"
```

```
#include "monta.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormPrincipal *FormPrincipal;
//---------------------------------------------------------------------------
__fastcall TFormPrincipal::TFormPrincipal(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------


void __fastcall TFormPrincipal::ItemAbrirClick(TObject *Sender)
{

  if (MemoSap -> Modified)
    if (MessageBox(Handle,
        "O arquivo foi alterado. Deseja salvá-lo?",
        OpenDialogSAP->FileName.c_str(),
        MB_YESNO|MB_ICONQUESTION)==IDYES)
        ItemSalvarClick(this);
    if (OpenDialogSAP -> Execute()){
        MemoSap->Lines->LoadFromFile(OpenDialogSAP->FileName);
        SaveDialogSAP->FileName = OpenDialogSAP->FileName;
        Caption = ExtractFileName(OpenDialogSAP->FileName);
        MemoSap->Modified = false;
        Arquivo = OpenDialogSAP->FileName;
    }
}
//---------------------------------------------------------------------------
void __fastcall TFormPrincipal::ItemFecharClick(TObject *Sender)
{
  if (MemoSap -> Modified)
    if (MessageBox (Handle, "O arquivo foi alterado. Deseja salvá-lo?",
        OpenDialogSAP->FileName.c_str(),
        MB_YESNO|MB_ICONQUESTION) == IDYES)
        ItemSalvarClick(this);
        OpenDialogSAP->FileName = "";
        Caption = "SemNome - Memo1";
        MemoSap->Modified = false;
        MemoSap->Clear();
```

```
  Close();
}
//---------------------------------------------------------------------------

void __fastcall TFormPrincipal::Executar1Click(TObject *Sender)
{
/*    if (MemoSap -> Modified)
      if (MessageBox(Handle,
          "O arquivo foi alterado. Deseja salvá-lo?",
          OpenDialogSAP->FileName.c_str(),
          MB_YESNO|MB_ICONQUESTION)==IDYES)
          ItemSalvarClick(this );
          MemoSap->Modified = false;
          Arquivo = OpenDialogSAP->FileName;

    monta montador (Arquivo.c_str());
    FormMontador->Show();
    FormMontador->AtribuirCaminho(Arquivo);*/
}
//---------------------------------------------------------------------------


void __fastcall TFormPrincipal::ItemNovoClick(TObject *Sender)
{
if (MemoSap -> Modified)
    if (MessageBox (Handle, "O arquivo foi alterado. Deseja salvá-lo?",
          OpenDialogSAP->FileName.c_str(),
          MB_YESNO|MB_ICONQUESTION) == IDYES)
          ItemSalvarClick(this );
          OpenDialogSAP->FileName = "";
          Caption = "SemNome - Memo1";
          MemoSap->Modified = false;
          MemoSap->Clear();
}
//---------------------------------------------------------------------------


void __fastcall TFormPrincipal::ItemSalvarClick(TObject *Sender)
{
if (OpenDialogSAP->FileName == "")
    ItemSalvarcomoClick(this);
```

```
else{
    MemoSap->Lines->SaveToFile(OpenDialogSAP->FileName);
    MemoSap->Modified = false;
    Arquivo = OpenDialogSAP->FileName;
    }
}
//---------------------------------------------------------------------------


void __fastcall  TFormPrincipal::ItemSalvarcomoClick(TObject *Sender)
{
 if (SaveDialogSAP->Execute()){
    MemoSap->Lines->SaveToFile(SaveDialogSAP->FileName);
    OpenDialogSAP->FileName = SaveDialogSAP->FileName;
    Caption = ExtractFileName(OpenDialogSAP->FileName);
    Arquivo = OpenDialogSAP->FileName;
    MemoSap->Modified = false;
    }
}
//---------------------------------------------------------------------------
void __fastcall  TFormPrincipal::BitBtn1Click(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------------
```

A.1.3 Montador

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian  Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include  "U_Montador.h"
#include  "U_Simulador.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
```

```
#pragma resource "*.dfm"
TFormMontador *FormMontador;
//---------------------------------------------------------------------------
__fastcall TFormMontador::TFormMontador(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormMontador::FormResize(TObject *Sender)
{
  RichEditHex->Width = (FormMontador->Width/4);
  RichEditLog->Height = 89;
  BitBtnFechar->Left = (FormMontador->Width - BitBtnFechar->Left)/2;
}
//---------------------------------------------------------------------------
void __fastcall TFormMontador::ItemExecutarClick(TObject *Sender)
{
  FormSimulador->Show();
}

void __fastcall TFormMontador::AtribuirCaminho( String Caminho)
{
  String nome_sem_extensao;
  caminho_dos_arquivos = ExtractFilePath(Caminho);
  nome_dos_arquivos = ExtractFileName(Caminho);
  nome_sem_extensao = Obtem_Nome_do_Arquivo(nome_dos_arquivos);
//================Arquivo .hex============================
  String arquivo;
  String arquivoHex;
  arquivo = caminho_dos_arquivos+nome_sem_extensao+".hex";
  arquivoHex = arquivo;
  FormSimulador->RecebeArquivoHex(arquivo);
  RichEditHex->Lines->LoadFromFile(arquivo);

//================Arquivo .lst============================
  arquivo = "";
  arquivo = caminho_dos_arquivos+nome_sem_extensao+".lst";
  FormSimulador->RecebeArquivoLst(arquivo);
  RichEditLst->Lines->LoadFromFile(arquivo);

//================Arquivo .log============================
  arquivo = "";
```

```
  arquivo = caminho_dos_arquivos+nome_sem_extensao+".log";
  RichEditLog->Lines->LoadFromFile(arquivo);

//================Arquivo.sml=============================
  arquivo = "";
  arquivo = caminho_dos_arquivos+nome_sem_extensao+".sml";
  FormSimulador->RecebeArquivoSmlHex(arquivo,arquivoHex);
}
//---------------------------------------------------------------------------
String __fastcall TFormMontador::Obtem_Nome_do_Arquivo( String nome_do_arquivo )
{
  String arquivo = "";
  for(int i = 1; i < nome_do_arquivo.Length(); i ++)
  {
    if (nome_do_arquivo[i] == '.')
      break;
    else AppendStr(arquivo, nome_do_arquivo[i]);
  }
  return arquivo;
}

bool __fastcall TFormMontador::Verifica_Erros()
{
  String linha = "";
  int i = 0;
  String Erro = "Erro";
  bool Encontrou_erro = false;
  while ((i < RichEditLog->Lines->Count) && (!Encontrou_erro))
  {
    linha = (RichEditLog->Lines->Strings[i]).SubString(0, 4);
    if (Erro == linha ) Encontrou_erro = true;
    i++;
  }
  return !Encontrou_erro;
}
//---------------------------------------------------------------------------
void __fastcall TFormMontador::ItemFecharClick(TObject *Sender)
{
  BitBtnFechar->Click();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TFormMontador::BitBtnFecharClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
```

A.1.4 Caixa de diálogo para entrada do arquivo montador

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_DialogoASM.h"
#include "U_Montador.h"
#include "monta.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormDialogoASM *FormDialogoASM;
//---------------------------------------------------------------------------
__fastcall TFormDialogoASM::TFormDialogoASM(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormDialogoASM::BitBtnCancelarClick(TObject *Sender)
{
  DestroyWnd();
}
//---------------------------------------------------------------------------
void __fastcall TFormDialogoASM::FormClose(TObject *Sender,
      TCloseAction &Action)
{
  Action = caFree;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TFormDialogoASM::BitBtnProcurarClick(TObject *Sender)
{
  if (OpenDialogASM -> Execute()){
    EditArquivoASM->Text = OpenDialogASM->FileName;
    Arquivo = EditArquivoASM->Text;
  }
}
//---------------------------------------------------------------------------
 // ao clicar o botão montador no formulario principal é exibido
 // uma caixa de dialogo para entrada do nome do arquivo.asm
 // com o nome do arquivo é criado o objeto montador, que irá gerar
 // os arquivos .hex, . lst , . log , . map, .sml
 // após gerado os arquivos eles são exibidos no formulário do montador

void __fastcall TFormDialogoASM::BitBtnOKClick(TObject *Sender)
{
   if (EditArquivoASM->Text != ""){
     if ( FileExists(Arquivo)){
       monta montador (Arquivo.c_str());
       FormMontador->Show();//é aberto o formulário Montador
       FormMontador->AtribuirCaminho(Arquivo);//é enviado o caminho dos arquivos
       BitBtnCancelar->Click();
     }
     else{
       MessageBox(Handle,
         "O arquivo não encontrado !!!",
         "Erro",
         MB_OK);
     }
   }
   else{
       MessageBox(Handle,
         "Informe o nome do arquivo !!!",
         "Erro",
         MB_OK);
   }
}
//---------------------------------------------------------------------------
```

A.1.5 Simulador

```
// Projeto : Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//           Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_Simulador.h"
#include "U_EntradaDigital.h"
#include "U_EntradaAnalogica.h"
#include "celula_simb.h"
#include "lista_inst.h"
#include "lista_dire.h"
#include "celula_inst.h"
#include "celula_geral.h"
#include "celula_token.h"
#include "celula_hex.h"
#include "simul_tab.h"
#include "celu_simul.h"
#include <fstream>
#include <iomanip>
#include <vector>
#include <cstdio>

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormSimulador *FormSimulador;
//---------------------------------------------------------------------------
__fastcall TFormSimulador::TFormSimulador(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::BitBtnFecharClick(TObject *Sender)
{
    FormSimulador->Close();
    simu_inst.clear ();
    EditAC->Text = " ";
```

```
    EditCP->Text = "  ";
    EditAP->Text = "  ";
    EditEA->Text = "  ";
    EditED->Text = "  ";
    EditSA->Text = "  ";
    EditSD->Text = "  ";
    EditRE->Text = "  ";
    EditRI->Text = "  ";
    EditRB->Text = "  ";

    DBCheckBoxZero->Checked = false;
    DBCheckBoxIgual->Checked = false;
    DBCheckBoxVaiUm->Checked = false;
    DBCheckBoxSinal->Checked = false;

    BitBtnRun->Enabled = true;
    BitBtnPassoaPasso->Enabled = true;

    ListBoxHex->Items->Delete(ListBoxHex->ItemIndex);
    // FormEntradaDigital->EditEntradaDigital->Text = " ";
    simu_inst.clear ();
    end_cont.clear();



}
//---------------------------------------------------------------------------
void __fastcall  TFormSimulador::RecebeArquivoHex(String ArqHex){
        // RichEditHex->Lines->LoadFromFile(ArqHex);
        GroupBoxHex->Caption = ExtractFileName(ArqHex);
}
//---------------------------------------------------------------------------
void __fastcall  TFormSimulador::RecebeArquivoLst(String ArqLst){
        RichEditLst->Lines->LoadFromFile(ArqLst);
        GroupBoxLst->Caption = ExtractFileName(ArqLst);

}
//---------------------------------------------------------------------------
void __fastcall  TFormSimulador::RecebeArquivoSmlHex(String ArqSml,String ArqHexa){

   /*vector <celula_geral> simu_inst;
   celula_geral  simu_celula;
```

```
celula_hex cel_endcont;
celula_hex inter;
vector <celula_hex> end_cont  ;*/
arquivohexadecimal = ArqHexa;
ifstream arquivo_hex (ArqHexa.c_str());

string sep;
int endereco;
int conteudo;
string endhex;
string conthex;

while (!arquivo_hex.eof()){

      arquivo_hex » hex » endereco » sep » hex » conteudo;
      cel_endcont.setcelula_hex(endereco, conteudo);
      end_cont.push_back(cel_endcont);
      arquivo_hex.get();
}
arquivo_hex.close();

string strgeral;
ifstream arquivohex (ArqHexa.c_str());
while (!arquivohex.eof()){

      arquivohex » endhex » sep » conthex;
      strgeral = endhex+": "+conthex;
      ListBoxHex->Items->Add(strgeral.c_str());
      arquivohex.get();
}
arquivohex.close();

//criação de vetor de célula geral inst_program
//   vector <celula_geral> simu_inst;
//celula_geral simu_celula;
//   void setcelulageral(string instrucao, int codigo,
// int endereco, string operando = "@", int end_op = -1, int operando_real = -1);
/*
      arquivo_sml « inst_program[cont].getinst ()«" "
                  « inst_program[cont].getcodigo () « " "
                  « inst_program[cont].getendereco () « " "
                  « inst_program[cont].getoperando () « " "
```

```
                    « inst_program[cont].getendereco_op () « " "
                    «  inst_program[cont].getoperando_geral()«endl;
    */
    ifstream  arquivo_sml (ArqSml.c_str());
    string  instrucao_sml;
    int  codigo_sml;
    int  endereco_sml;
    string  operando_sml;
    int  op_end_sml;
    int  op_real_sml;



    while (!arquivo_sml.eof()){

    arquivo_sml » instrucao_sml » codigo_sml »  endereco_sml » operando_sml » op_end_sml » op_real
    simu_celula.setcelulageral(instrucao_sml, codigo_sml , endereco_sml , operando_sml , op_end_sml , o
    simu_inst.push_back(simu_celula);
    arquivo_sml.get();

    }

}

//---------------------------------------------------------------------------
void __fastcall  TFormSimulador::executa(int simul_acao){
    int  exec_acao = simul_acao;
    int  endereco_int;
    String  teste;
    // instrução  LDA, ACC <- [endereco]
    // endereco = simu_inst[cont].getoperando_geral();
    // percorre o arquivo.hex e  verifica  a  linha  do endereco dada pelo linha-1
    // é pego seu conteúdo e colocado no acumulador
    switch (exec_acao){
    case 1:
        RE = CP;//registrador de endereços recebe o conteúdo do contador de programas
        AtualizaRegistradores(); // atualiza  os valores dos registradores
        RE = simu_inst[cont].getoperando_geral();//o  registrador  de endereços recebe o operando do instru
        AtualizaRegistradores(); // atualiza  os valores dos registradores
        ACC = procura_cont(simu_inst[cont].getoperando_geral());//o acumulador recebe o conteúdo do end
        AtualizaRegistradores(); // atualiza  os valores dos registradores
        CP = CP + 1; //contador de programa aponta para a próxima instrução
```

```
        AtualizaRegistradores(); // atualiza  os valores  dos registradores
        break;
    // instrução : ADI <dado> <– ACC = ACC + dado
    case 2:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,simu_inst[cont].getoperando_geral());
        ACC = ACC + simu_inst[cont].getoperando_geral();
        ativa_flag_sinal (ACC);
        if  (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução : ADD <endereço> <– ACC = ACC + [endereco]
    case 3:
        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        RB =  procura_cont(simu_inst[cont].getoperando_geral());
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,procura_cont(simu_inst[cont].getoperando_geral()));
        ACC = ACC + procura_cont(simu_inst[cont].getoperando_geral());
        ativa_flag_sinal (ACC);
        if  (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;

    // instrução : SUBI <dado> <– ACC = ACC – dado
    case 4:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
```

```
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,(~(simu_inst[cont].getoperando_geral())));
        ACC = ACC – simu_inst[cont].getoperando_geral();
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: SUB <endereco> <– ACC = ACC – [endereco]
    case 5:
        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        RB = procura_cont(simu_inst[cont].getoperando_geral());
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,(~(procura_cont(simu_inst[cont].getoperando_geral()))));
        ACC = ACC – procura_cont(simu_inst[cont].getoperando_geral());
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: CMP <dado>
    // compara ACC com o dado e seta flag igual se forem iguais
    case 6:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        if (ACC == simu_inst[cont].getoperando_geral())
            flag_igual = 1;
        else flag_igual = 0;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
```

```
        break;
    //instrução: INC <– ACC+1
    case 7:
        ativa_flag_vaium(ACC,1);
        ACC = ACC + 1;
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        break;
    //instrução:DCC <– ACC–1
    case 8:
        ativa_flag_vaium(ACC,~1);
        ACC = ACC – 1;
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        break;
    //instrução:LDI <dado> <– ACC = dado
    case 9:
        RE = CP;
        AtualizaRegistradores();
        ACC = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    //instrução: STA <endereço> <– endereço = ACC
    case 10:

        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        FormSimulador–>ListBoxHex–>Items–>Delete(RE);
        teste = converte_hexa(RE)+" : "+converte_hexa(ACC);
        FormSimulador–>ListBoxHex–>Items–>Insert(RE, teste);
        //procura o endereço no vetor do arquivo hex
        endereco_int = procura_indice(simu_inst[cont].getoperando_geral());
```

```
        // seta seu novo conteúdo
        end_cont[endereco_int].setConteudo(ACC);
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução : AND <dado> <– ACC = ACC and dado
    case 11:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC & simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução : OR <dado> <– ACC = ACC or dado
    case 12:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC | simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução:XOR <dado> <– ACC = ACC xor dado
    case 13:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC ^ simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        break;
    // instrução : CLR <– ACC = 0
    case 14:
        ACC = 0;
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
```

```
        AtualizaRegistradores();
        break;
    //instrução: NOT <– ACC = /ACC
    case 15:
        ACC = ~ACC;
        AtualizaRegistradores();
        break;
    //instrução: JMP <endereco> <– CP = endereco
    case 16:
        RE = CP;
        AtualizaRegistradores();
        CP = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        break;
    //instrução: JZ <endereço> <– if /ZERO = 0 CP = endereco
    //else CP = CP++;
    case 17:
        RE = CP;
        AtualizaRegistradores();
        if (flag_zero == 0){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else {
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    //instrução: JC <endereço> <– if /VAI–UM = 0 CP = endereco
    //else CP = CP++;
    case 18:
        RE = CP;
        AtualizaRegistradores();
        if (flag_vai_um == 0){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
```

```
// instrução : JS <endereço> <– if SINAL = 1 CP = endereco
// else  CP = CP++;
case 19:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_sinal  == 1){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else{
           CP = CP + 1;
           AtualizaRegistradores();
     }
     break;
// instrução : JEQ <endereço> <– if IGUAL = 1 CP = endereco
// else  CP = CP++;
case 20:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_igual  == 1){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else {
          CP = CP + 1;
          AtualizaRegistradores();
     }
     break;
// instrução : JNZ <endereço> <– if /ZERO = 1 CP = endereco
// else  CP = CP++;
case 21:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_zero == 1){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else{
          CP = CP + 1;
          AtualizaRegistradores();
     }
     break;
```

```
// instrução : JNC <endereço> <– if /VAI–UM = 1 CP = endereco
// else  CP = CP++;
case 22:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_vai_um == 1){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else{
          CP = CP + 1;
          AtualizaRegistradores();
     }
     break;
// instrução : JNS <endereço> <– if SINAL = 0 CP = endereco
// else  CP = CP++;
case 23:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_sinal  == 0){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else{
          CP = CP + 1;
          AtualizaRegistradores();
     }
     break;
// instrução : JNEQ <endereço> <– if IGUAL = 0 CP = endereco
// else  CP = CP++;
case 24:
     RE = CP;
     AtualizaRegistradores();
     if  ( flag_igual  == 0){
          CP = simu_inst[cont].getoperando_geral();
          AtualizaRegistradores();
     }
     else{
          CP = CP + 1;
          AtualizaRegistradores();
     }
     break;
```

```
// instrução: CALL <endereço> <– [AP] = CP
// CP = endereço   AP <– AP – 1;
case 25:
    RE = AP;
    AtualizaRegistradores();
    RE = CP;
    simu_inst[cont].getoperando_geral();
     inter.setcelula_hex(AP, CP);
    end_cont.push_back(inter);
    CP = simu_inst[cont].getoperando_geral();
    AtualizaRegistradores();
    AP = AP –1;
    AtualizaRegistradores();
    break;
// instrução: RET <– CP = [AP]
// AP = AP+1
case 26:
    AP = AP + 1;
    AtualizaRegistradores();
    RE = AP;
    CP = procura_cont(AP);
    AtualizaRegistradores();
    CP = CP + 1;
    AtualizaRegistradores();
    break;
// instrução: INA <– ACC = entrada analógica
case 27:
    Application–>CreateForm(__classid(TFormEntradaAnalogica), &FormEntradaAnalogica);
    FormEntradaAnalogica–>ShowModal();
    ACC = entrada_analogica;
    AtualizaRegistradores();
    break;
// instrução: IND <– ACC = entrada digital
case 28:
    Application–>CreateForm(__classid(TFormEntradaDigital), &FormEntradaDigital);
    FormEntradaDigital–>ShowModal();
    ACC = entrada_digital;
    AtualizaRegistradores();
    break;
// instrução: OUTA <– saida analógica = ACC
case 29:
    saida_analogica = ACC;
```

```
            AtualizaRegistradores();
            break;
        //instrução: OUTD <– saida digital = ACC

        case 30:
            saida_digital = ACC;
            AtualizaRegistradores();
            break;
        //instrução: HALT
        case 31:
            cont_r = simu_inst.size();
            cont_pp = simu_inst.size();
            break;
        //instrução: PUSH
        case 32:
            RE = AP;
            AtualizaRegistradores();
            FormSimulador–>ListBoxHex–>Items–>Delete(RE);
            teste = converte_hexa(RE)+" : "+converte_hexa(ACC);
            FormSimulador–>ListBoxHex–>Items–>Insert(RE, teste);
            //procura o endereço no vetor do arquivo hex
            endereco_int = procura_indice(AP);
            //seta seu novo conteúdo
            end_cont[endereco_int].setConteudo(ACC);
            AP = AP –1;
            AtualizaRegistradores();
            break;
        //instrução POP
        case 33:
            AP = AP + 1;
            AtualizaRegistradores();
            RE = AP;
            AtualizaRegistradores();
            ACC = procura_cont(procura_indice(AP));
            AtualizaRegistradores();
            break;
        }//fim switch
}

//  vetor de células e conteúdos
//  celula_hex cel_endcont;
//  vector <celula_hex> end_cont;
```

```
int __fastcall TFormSimulador::procura_cont(int endere){
    int parada;
    for (int i = 0; i < end_cont.size(); i++){
        if (endere == end_cont[i].getEnd()){
            parada = i;
            i = end_cont.size();

        }
    }
    return (end_cont[parada].getConteudo());
}

int __fastcall TFormSimulador::procura_indice(int endere){
    int parada;
    for (int i = 0; i < end_cont.size(); i++){
        if (endere == end_cont[i].getEnd()){
            parada = i;
            i = end_cont.size();

        }
    }
    return (parada);
}

void __fastcall TFormSimulador::ativa_flag_vaium(int op1, int op2){
    int operando1[8];
    int operando2[8];
    int result [9];
    result [0] = 0;

    for (int i = 0; i < 8; i++){
        operando1[i] = op1%2;
        op1 = op1/2;
    }

    for (int i = 0; i < 8; i++){
        operando2[i] = op2%2;
        op2 = op2/2;
    }

    for (int i = 0; i < 8; i++)
```

```
            result [ i +1] = ( result [ i ] &( operando1[i] ^ operando2[i ])) | ( operando1[i] & operando2[i]);

        flag_vai_um = result [8];


    }

    void __fastcall TFormSimulador::ativa_flag_sinal(int op1){
        int operando1[8];

        for ( int i = 0; i < 8; i++){
            operando1[i] = op1%2;
            op1 = op1/2;
        }

        flag_sinal = operando1[7];

    }

    void __fastcall TFormSimulador::BitBtnRunClick(TObject *Sender)
    {
        cont_r = 0;
        ACC = 0;
        AP = 255;
        entrada_analogica = 0;
        entrada_digital = 0;
        saida_analogica = 0;
        saida_digital = 0;
        CP_ant = 0;
        cont = 0;

        // int acao;
        // string str;
        // simul_tab acao_tab;

        while (cont_r < simu_inst.size ()){
            simu_celula = simu_inst[cont_r]; // celula_geral
            cont = cont_r;
            str = simu_celula.getinst ();
            if (( str != "DB") || (str != "DS")){
                acao = acao_tab.procura_simul(str);
                RE = CP;//primeiro ciclo de busca
```

```
                AtualizaRegistradores();
                CP = CP + 1; //segundo ciclo de busca
                AtualizaRegistradores();
                RI = simu_celula.getcodigo();
                AtualizaRegistradores(); // terceiro  ciclo  de busca
                for (int i = 0; i < 1000; i++){
                }
                CP_ant = CP;
                executa(acao); //executa cada instrução
                if (CP != CP_ant+1){
                    cont_r = procuraCP(CP);
                }
                else{
                    AtualizaRegistradores();
                    cont_r++;
                }
            }
            else {
                cont_r++;
                 }
        }
        BitBtnRun->Enabled = false;
        BitBtnPassoaPasso->Enabled = false;
}
//---------------------------------------------------------------------------
int __fastcall TFormSimulador::procuraCP (int end){
        int i = 0;
        while (i < simu_inst.size ()){
          if (simu_inst[i].getendereco() == end){
            return i;
         }
         else i++;
        }
}
//---------------------------------------------------------------------------
String __fastcall TFormSimulador::converte_hexa (int valor){
        vector <int> inteiros ;
        String num;
        if (valor < 10){
            num = "0" + IntToStr(valor);
            return num;
        }
```

```
    else{
        do{
            inteiros.push_back(valor % 16);
            valor = valor / 16;
        }while (valor != 0);


        for(int i = 0; i < inteiros.size(); i++){
            if ( inteiros[inteiros.size() - i -1] < 10)
                num = num + IntToStr(inteiros[inteiros.size() - i -1]);
            else{
                switch(inteiros[inteiros.size() - i -1]){
                    case 10: num = num + "A";
                             break;
                    case 11: num = num + "B";
                             break;
                    case 12: num = num + "C";
                             break;
                    case 13: num = num + "D";
                             break;
                    case 14: num = num + "E";
                             break;
                    case 15: num = num + "F";
                             break;
                    } // fim switch

                } // fim else
        } // fim for
        if (num.Length()==1)
            num = "0"+num;
        return num;
    } // fim else

}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::RadioGroupValorClick(TObject *Sender)
{
 if (RadioGroupValor -> ItemIndex == 0){
    EditAC->Text = IntToStr(ACC);
    EditCP->Text = IntToStr(CP);
    EditAP->Text = IntToStr(AP);
    EditEA->Text = IntToStr(entrada_analogica);
```

```
      EditED->Text = IntToStr(entrada_digital);
      EditSA->Text = IntToStr(saida_analogica);
      EditSD->Text = IntToStr(saida_digital);
      EditRE->Text = IntToStr(RE);
      EditRI->Text = IntToStr(RI);
      EditRB->Text = IntToStr(RB);
}
 if (RadioGroupValor -> ItemIndex == 1){
     EditAC->Text = converte_hexa(ACC);
     EditCP->Text = converte_hexa(CP);
     EditAP->Text = converte_hexa(AP);
     EditEA->Text = converte_hexa(entrada_analogica);
     EditED->Text = converte_hexa(entrada_digital);
     EditSA->Text = converte_hexa(saida_analogica);
     EditSD->Text = converte_hexa(saida_digital);
     EditRE->Text = converte_hexa(RE);
     EditRI->Text = converte_hexa(RI);
     EditRB->Text = converte_hexa(RB);
}
}
//---------------------------------------------------------------------------

void __fastcall TFormSimulador::BitBtnResetClick(TObject *Sender)
{
      cont_r = 0;
      cont_pp = 0;
      ACC = 0;
      AP = 255;
      CP = 0;
      RE = 0;
      RI = 0;
      RB = 0;
      entrada_analogica = 0;
      entrada_digital  = 0;
      saida_analogica = 0;
      saida_digital  = 0;
      BitBtnPassoaPasso->Enabled = true;
      BitBtnRun->Enabled = true;
      flag_zero  = 0;  // flag  / zero
      flag_vai_um = 0;  // flag  / vai-um
      flag_igual  = 0;  // flag  igual
      flag_sinal  = 0;  // flag  sinal
```

```
    AtualizaRegistradores();

    ListBoxHex->Clear();
    // FormEntradaDigital->EditEntradaDigital->Text = " ";
    string  strgeral ;
    string  endhex, conthex;
    string  sep;
    ifstream   arquivohex (arquivohexadecimal.c_str());
    while (!arquivohex.eof()){

        arquivohex » endhex » sep » conthex;
        strgeral = endhex+": "+conthex;
        ListBoxHex->Items->Add(strgeral.c_str());
        arquivohex.get();
    }
    arquivohex.close();
    end_cont.clear();
    ifstream   arquivo_hex (arquivohexadecimal.c_str());
    int endereco;
    int conteudo;
    while (!arquivo_hex.eof()){

        arquivo_hex » hex » endereco » sep » hex » conteudo;
        cel_endcont.setcelula_hex(endereco, conteudo);
        end_cont.push_back(cel_endcont);
        arquivo_hex.get();
    }
    arquivo_hex.close();


}
//---------------------------------------------------------------------------

void __fastcall TFormSimulador::BitBtnPassoaPassoClick(TObject *Sender)
{
    BitBtnRun -> Enabled = false; //se a opção passo a passo o botão run é desbilitado até o término do p
    simu_celula = simu_inst[cont_pp];//celula_geral contém os dados de cada instrução do programa fonte
    cont = cont_pp;//seta o contador_principal com o contador do passo a passo.
    str = simu_celula.getinst();
    if (( str != "DB") || (str != "DS")){
        acao = acao_tab.procura_simul(str);
        RE = CP;//primeiro ciclo de busca
```

```
            AtualizaRegistradores();
            CP = CP + 1; //segundo ciclo de busca
            AtualizaRegistradores();
            RI = simu_celula.getcodigo();
            AtualizaRegistradores(); // terceiro  ciclo  de busca
            executa(acao);  //executa cada instrução
            if (CP != CP_ant+1){
                  cont_pp = procuraCP(CP);
            }
            else{
                  AtualizaRegistradores();
                  cont_pp++;
            }
      }
      else cont_pp++;
      if (cont_pp >= simu_inst.size())
            BitBtnPassoaPasso->Enabled = false;

}

//---------------------------------------------------------------------------
void __fastcall TFormSimulador::AtualizaRegistradores(){
int i = 0;
while (i < 1000000)//atraso
      i++ ;
if (RadioGroupValor -> ItemIndex == 0){
    EditAC->Text = IntToStr(ACC);
    EditCP->Text = IntToStr(CP);
    EditAP->Text = IntToStr(AP);
    EditEA->Text = IntToStr(entrada_analogica);
    EditED->Text = IntToStr(entrada_digital);
    EditSA->Text = IntToStr(saida_analogica);
    EditSD->Text = IntToStr(saida_digital);
    EditRE->Text = IntToStr(RE);
    EditRI->Text = IntToStr(RI);
    EditRB->Text = IntToStr(RB);
}
if (RadioGroupValor -> ItemIndex == 1){
      EditAC->Text = converte_hexa(ACC);
      EditCP->Text = converte_hexa(CP);
      EditAP->Text = converte_hexa(AP);
      EditEA->Text = converte_hexa(entrada_analogica);
```

```
    EditED->Text = converte_hexa(entrada_digital);
    EditSA->Text = converte_hexa(saida_analogica);
    EditSD->Text = converte_hexa(saida_digital);
    EditRE->Text = converte_hexa(RE);
    EditRI->Text = converte_hexa(RI);
    EditRB->Text = converte_hexa(RB);
}

ListBoxHex->ItemIndex = RE;

DBCheckBoxZero->Checked = flag_zero;
DBCheckBoxIgual->Checked = flag_igual;
DBCheckBoxVaiUm->Checked = flag_vai_um;
DBCheckBoxSinal->Checked = flag_sinal;
}
//---------------------------------------------------------------------------
```

A.1.6 Simulador - Caixa de diálogo para entrada do arquivo.hex

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_DialogoHex.h"
#include "U_Montador.h"
#include "U_Simulador.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormDialogoSimulador *FormDialogoSimulador;
//---------------------------------------------------------------------------
__fastcall TFormDialogoSimulador::TFormDialogoSimulador(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
// na caixa de diálogo para entrada do arquivo.hex
```

```
void __fastcall TFormDialogoSimulador::BitBtnProcurarClick(TObject *Sender)
{
 if (OpenDialogHex -> Execute()){
    EditArquivoHex->Text = OpenDialogHex->FileName;
    Arquivo = EditArquivoHex->Text;
    }
}
//---------------------------------------------------------------------------
 //
void __fastcall TFormDialogoSimulador::BitBtnOkClick(TObject *Sender)
{
 String caminho_arquivo;
 String nome_arquivo;
 String nome;
 int i = 0;
 if (EditArquivoHex->Text != ""){
     if ( FileExists(Arquivo)){
       caminho_arquivo = ExtractFilePath(Arquivo);
       nome_arquivo = ExtractFileName(Arquivo);
       FormMontador->AtribuirCaminho(Arquivo);
       FormSimulador->Show();
       BitBtnCancelar->Click();
     }
     else{
       MessageBox(Handle,
         "O arquivo não encontrado !!!",
         "Erro",
         MB_OK);
     }
  }
  else{
       MessageBox(Handle,
         "Informe o nome do arquivo !!!",
         "Erro",
         MB_OK);
  }
}
//---------------------------------------------------------------------------

void __fastcall TFormDialogoSimulador::BitBtnCancelarClick(TObject *Sender)
{
  DestroyWnd();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TFormDialogoSimulador::FormClose(TObject *Sender,
      TCloseAction &Action)
{
 Action = caFree;
}
//---------------------------------------------------------------------------
```

A.1.7 Simulador - Caixa de diálogo para entrada analógica

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_EntradaAnalogica.h"
#include "U_Simulador.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormEntradaAnalogica *FormEntradaAnalogica;
//---------------------------------------------------------------------------
__fastcall TFormEntradaAnalogica::TFormEntradaAnalogica(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TFormEntradaAnalogica::EditEntradaAnalogicaKeyPress(TObject *Sender,
      char &Key)
{
 if (Key > '9'||Key < '0'){
     Key = 0;
     MessageBeep(0);
     }
}
//---------------------------------------------------------------------------

void __fastcall TFormEntradaAnalogica::BitBtnOKClick(TObject *Sender)
{
     if  (StrToInt(EditEntradaAnalogica->Text) > 5)
         MessageBox(Handle,
```

```
            "Valor Inválido !!!",
            "Erro",
            MB_OK);
      else FormSimulador->entrada_analogica = (255 * (StrToInt(EditEntradaAnalogica->Text)))/5;
}
//---------------------------------------------------------------------------
```

A.1.8 Simulador - Caixa de diálogo para entrada digital

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_EntradaDigital.h"
#include "U_Simulador.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormEntradaDigital *FormEntradaDigital;
//---------------------------------------------------------------------------
__fastcall TFormEntradaDigital::TFormEntradaDigital(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormEntradaDigital::EditEntradaDigitalKeyPress(
      TObject *Sender, char &Key)
{
 if (Key > '9'||Key < '0'){
    Key = 0;
    MessageBeep(0);
    }
}
//---------------------------------------------------------------------------
void __fastcall TFormEntradaDigital::BitBtnOKClick(TObject *Sender)
{
 if  (StrToInt(EditEntradaDigital->Text) > 255)
    MessageBox(Handle,
        "Valor Inválido !!!",
        "Erro",
        MB_OK);
```

```
else FormSimulador->entrada_digital = StrToInt(EditEntradaDigital->Text);
}
//---------------------------------------------------------------------------
```

A.1.9 Ajuda

```
// Projeto: Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//          Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_Help.h"
#include "U_Sobre.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormHelp *FormHelp;
//---------------------------------------------------------------------------
__fastcall TFormHelp::TFormHelp(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TFormHelp::BitBtnFecharClick(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------------

void __fastcall TFormHelp::MduloCMS1Click(TObject *Sender)
{
FormSobre->Show();
}
//---------------------------------------------------------------------------
```

```
void __fastcall  TFormHelp::Fechar1Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
```

A.1.10 Ajuda - Sobre

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include  "U_Sobre.h"
 // Projeto :  Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
 //            Simulação para Programação de um Processador Discreto Microprogramável
 // Orientada: Edna Mie Kanazawa
 // Orientador: Wilian  Soares Lacerda
 // Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource  "*.dfm"
TFormSobre *FormSobre;
//---------------------------------------------------------------------------
 __fastcall  TFormSobre::TFormSobre(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall  TFormSobre::BitBtnOKClick(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
```

Apêndice B

Programa Montador

B.1 Código do Programa Montador

O programa montador é um objeto que necessita de um arquivo.asm como entrada.
Como resultado tem-se a geração dos arquivos: .hex, .lst, .sml, .map e .log.

```
/*
Projeto : Desenvovlvimento de um Sistema Integrado compilação, montagem e simulação
Fase 1: Desenvolvimento do Montador
Desenvolvedor: Edna Mie Kanazawa
Orientador: Wilian Soares Lacerda
*/

#ifndef MONTA_H
#define MONTA_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <ctype.h>
#include <iomanip>
#include "celula_simb.h"
#include "lista_inst.h"
#include "lista_dire.h"
#include "celula_inst.h"
#include "celula_geral.h"
#include "celula_token.h"
```

```
class monta{

public:

    monta(string nome_arquivo);//, string  arq_map, string arq_hex, string  arq_lst ,  string  arq_log);
    vector<celula_geral> getVetor();
    string  getArquivoHex();



private:
    int  verifica_rotulo ( string  rotulo );
    celula_inst  verifica_instrucao ( string  instrucao );
    int  verifica_diretiva ( string  diretiva ,  string  op, int contador);
    int  procura_linha_token(string pal );
    celula_geral  celinst_geral ;
    celula_geral  celinst_geral2 ;
    vector <celula_geral> inst_program;
    vector <celula_token> t_linha ;
    vector<string> linha ;
    string  simuArquivoHex;
};

 // CONSTRUTOR a partir do arquivo
monta::monta(string nome_arquivo){//, string  arq_map, string arq_hex, string  arq_lst ,  string  arq_log){
    // o primeiro :  nome_arquivo é o programa fonte
    // as outras  strings  sao nomes dos arquivos que devem ser abertos para insercao de dados
    string  arq_principal  = nome_arquivo;
    string  outros_arquivos = nome_arquivo.erase(nome_arquivo.find('.'));
    ifstream  entrada(arq_principal.c_str ());              //  Objeto arquivo

     //  Caso o arquivo não puder ser aberto é emitido uma mensagem de erro
     if  (! entrada){
         cerr  << "\n IMPOSSIVEL ABRIR ARQUIVO!"« endl;
         exit (1);
    }


    // o nome do arquivo.hex deve ser armqzenado numa string para ser utilizado no simulador
    simuArquivoHex = outros_arquivos+".hex";//arq_hex;
```

```
// no vetor de saida temos cada linha do programa, representadas como uma string
string str_temp;
int resp_ponto;

do{
     getline (entrada,str_temp);
     linha . push_back(str_temp);//vetor com o conteúdo de cada linha do arquivo fonte

}while(!entrada.eof ());
 // O arquivo fonte é fechado
entrada.close ();

// os comentarios são retirados e colocados no vetor linha2
// em cada linha é lido caracter por caracter e se houver um ; o comentario é retirado
int cont = 0;              // contador de linhas
string temp;               // string que recebe o conteudo de cada linha
int cont2 = 0;             // contador de caracteres da linha
vector < string > linha2 ; // vetor de linhas sem comentarios
string forma_linha;        // string que forma linha de instrucao sem comentarios

while (cont < linha . size ()){  // percorre todas as linhas do vetor de linhas
     temp = linha[cont];
     while (cont2 < temp.size ()){  // percorre cada caractere de cada linha
          if (temp[cont2] == ';'){ //se for encontrado um ; temos que seguir para outra linha
               cont2 = temp.size();
          }
          else {
               cont2++;      // caso contrario é formada a linha de instruções sem comentarios
               forma_linha = forma_linha + temp[cont2-1];
          }

     }
     linha2 . push_back(forma_linha);//a linha formada é armazenada no vetor secundario
     forma_linha.erase();
     cont2 = 0;
     cont++;

}

 // nao esquecer que o vetor linha contem todas as linhas do programa, mas com os comentarios
 // e o vetor linha 2 contem todas as linhas sem o comentario.
//-------------------------------------------------------------------------
```

```
    //abertura dos arquivos
    string  arq_map = outros_arquivos+".map";
    string  arq_hex = outros_arquivos+".hex";
    string  arq_lst  = outros_arquivos+".lst";
    string  arq_log = outros_arquivos+".log";
    string  arq_sml = outros_arquivos+".sml";

    ofstream arquivo_map (arq_map.c_str()); //arquivo de simbolos
    ofstream arquivo_hexa (arq_hex.c_str()); // arquivo de enderecos e dados
    ofstream arquivo_lst  ( arq_lst . c_str ());   // arquivo com o programa fonte + arquivo hex
    ofstream arquivo_log (arq_log.c_str ());   // arquivos com erros e sucessos
    ofstream arquivo_sml (arq_sml.c_str ());  // arquivo simulador;

//-------------------------------------------------------------------------
    // passo1
    //  inicialiazar   o contador de programa PC em 00
    // montar a tabela de simbolos em seus respectivos enderecos
    // ORG e EQU não sao diretivas
    // EQU somente atribue valor aos labels
    // ORG atualiza o contador de programa
    int  PC = 0;                  //  inicializa   PC
    cont  = 0;                    // contador de linhas  linhas
    int  contador = 0;
    string   str ;                // forma string
    vector < string > tokens;  // vetor  de tokens
    celula_simb simb;          // celula  de simbolos (contem o rotulo e seu endereco)
    vector <celula_simb> simbolos;//vetor de simbolos
    celula_inst  resp_inst ;    //  resposta da  verifica  instrucao
    int  resp = 0;
    celula_token token_linha;



    // forma os tokens, que são armazenados no vetor de tokens
    while (contador < linha2 . size ()){   // enquanto existir  linhas  no vetores
        temp = linha2[contador] + " "; //temp recebe cada linha do vetor
        while (cont < temp.size ()){
            if  (( temp[cont] == '\t')||(temp[cont] == ' ')||(temp[cont] == '\0')){
                if  ((! str . empty ()) || (  str . length  () != 0)){   //  str  forma as strings  label ,  instrucao ,  dire
                    tokens.push_back(str);  // vetor  com todos os tokens
                    token_linha.setToken(str ,  contador);
                    t_linha . push_back(token_linha);
```

```
                }
                str.erase();
                cont++;
            } // fim if
            else {
                str = str + temp[cont];
                cont ++;
            } // fim else
        } // fim while
        contador++;
        cont = 0;
    } // fim while

    // verifica se o último token é END
    if (tokens[tokens.size()-1] != "END")
        arquivo_log << "Programa não finalizado" << endl;

    // inicio real do passo1
    // um tokens é identificado como rotulo quando este tiver o caracter :
    // todas os tokens são percorridos identificados
    // quando um label é identificado é armazenado em sua celula_simb o nome do label e seu endereco
    // quando uma instrução é identificada é armazenado em sua celula_geral(instrução, código, endereço,
    // o operando pode estar representado por um abel ou por um numero.
    // no primeiro passo esse operando real é -1, pois somente no passo 2 é definido seu valor correto
    cont = 0;
    while (cont < tokens.size ()){
        resp = verifica_rotulo (tokens[cont]);
        if (resp == 1){ // verifica se a string é um rotulo
            str = tokens[cont];
            str = str.erase(str.find(':'));
            simb.setcelula_simb(str, PC);//seta a celula com o valor do rotulo e seu endereco
            simbolos.push_back(simb);//insere cada rotulo no vetor
        }
        // verifica se a string é uma instrucao
        resp_inst = verifica_instrucao (tokens[cont]);
        if (resp_inst.getCodigo() > 0){ // verifica se a instrução possui código
            if (resp_inst.getOperando() > 0)//e se o número de operando é maior que 0
                celinst_geral.setcelulageral(tokens[cont], resp_inst.getCodigo(), PC, tokens[cont+1], PC+
            else
                celinst_geral.setcelulageral(tokens[cont], resp_inst.getCodigo(), PC);

            inst_program.push_back(celinst_geral);
```

```
            PC = PC + (resp_inst.getOperando()) + 1;

        }

        if (cont > 1){
            if (tokens[cont-1] == "ORG")
                PC = (atoi(tokens[cont].c_str ()));

            if (tokens[cont-1] == "EQU"){
                simb.setcelula_simb(tokens[cont - 2], (atoi(tokens[cont].c_str ()))); // seta a celula com o
                simbolos.push_back(simb);//insere cada rotulo no vetor
            }

            // verifica se s string é uma diretiva DB ou DS
            if ((tokens[cont-1] == "DB") || (tokens[cont-1] == "DS")){
                if (tokens[cont-1] == "DB"){
                    celinst_geral.setcelulageral("DB", -1, PC, tokens[cont], PC);
                    inst_program.push_back(celinst_geral);
                }

                if (tokens[cont-1] == "DS"){
                    celinst_geral.setcelulageral("DS", -2, PC, tokens[cont], PC);
                    inst_program.push_back(celinst_geral);
                }


                PC = verifica_diretiva (tokens[cont-1], tokens[cont], PC);


            }
        }

        cont ++;

    } // fim while

    // se o contador de programa ultrapassar 256 posições de memória
    // é lançado no arquivo de log um aviso!!

    if (PC > 256)
        arquivo_log << "Espaço de memória insuficiente"«endl;
```

```
    // finalização do passo 1
    arquivo_log << "Passo1: Concluído " « endl;
//-------------------------------------------------------------------------
    // arquivo .map recebera o vetor de simbolos
    cont = 0;
    arquivo_map.setf(ios::uppercase);
    while (cont < simbolos.size()){
        simb = simbolos[cont];
        arquivo_map « setw(20) « setfill (' ') « simb.getSimb() << '\t';
        arquivo_map « setw(2) « setfill ('0') «hex « simb.getEnd() « endl;
        cont ++;
    }

    arquivo_log << "Arquivo.map: Construído" « endl;
//-------------------------------------------------------------------------
    // passo 2
    // no passo 1 foram montados 2 vetores:
    // um vetor de labels com seus respectivos valores
    // um vetor de instruções com seus respectivos codigos, endereços, operandos.

    // no passo 2
    // vector <celula_simb> simbolos – vetor simbolos;
    // vector <celula_geral> inst_program – vetor de instrucoes;
    // no vetor inst_program temos celula_geral
    // em cada celula_geral temos um campo onde temos os operandos
    // se a instrução não exigir operando por default = @
    // se o operando for um label este é procurado na tabela de simbolos
    // se for um valor inteiro é verificado se o operando é correto.
    // antes de entrar no passo 2 verificar se todos os operandos estão corretos
    // ou seja, se existe algum operando como instrução



    string str_2;
    string str_3;
    int cont_2 = 0;
    int num;
    int num_cont = 0;
    int resp_2 = 1;
    int linha_prog;
```

```
celula_simb simb2;
cont = 0;

while (cont < inst_program.size ()){   //  vetor  de instruções do programa
     celinst_geral2    = inst_program[cont]; // celula  do vetor  de instruções do programa
     str_2  =  celinst_geral2 .getoperando();  // string  2  recebe o operando da instrução
     if  ( str_2 != "@"){                        // se o tamanho da string é maior que zero significa  que temos um
          while (cont_2 < simbolos.size ()){ // portanto temos que procurar o operando no vetor de simbolo
               simb2 = simbolos[cont_2];
               str_3 = simb2.getSimb();

               if  ( str_2 == str_3 ){  // se o operando for encontrado no vetor de simbolos
                    cont_2 = simbolos.size (); // o laco  termina
                    inst_program[cont].setoperandoreal(simb2.getEnd()); //e o operando real é setado no ve

               } // fim  if
               else cont_2++;
          } // fim  while
          if  (( cont_2 >= simbolos.size()) && (str_2  !=  str_3 )){
               num = str_2.length ();
               while (num_cont < num){
                    resp_2 =  isdigit (str_2[num_cont]);
                    if  ( resp_2 == 0){
                         linha_prog = procura_linha_token(str_2);
                         arquivo_log  <<"Erro na linha: " « linha_prog  + 1 <<": "<<"operando i
                         num_cont = num;
                    } // fim  if
                    else num_cont++;
               } // fim  while
               // se  for  uma cadeia de digitos
               //  verifica –se o numero formado
               // se o numero formado for maior que 256
               // o numero é inválido
               if  ( resp_2 > 0){
                    if  ( atoi (str_2 . c_str ()) < 256)
                         inst_program[cont].setoperandoreal(atoi(str_2. c_str  ()));   // e o operando real é set
                    else{
                         linha_prog = procura_linha_token(str_2);
                         arquivo_log  << "Erro na linha: " « (linha_prog  + 1 )<<": "<<"operand
                         }
               } // fim  if
```

```
                num_cont = 0;
            } // fim  if

        } // fim  if
        else {
            resp_inst =  verifica_instrucao  (inst_program[cont]. getinst ());
            if  (resp_inst .getCodigo() > 0){
                if  (resp_inst .getOperando() > 0)
                    arquivo_log  <<"Erro na linha: " « (procura_linha_token(str_2) + 1)<< ":" <<

            } // fim  if
        } // fim  else
        cont++;
        cont_2 = 0;


} // fim  while  principal
arquivo_log  << "Passo2: Concluído" « endl;

// montagem do arquivo .hex
// PC é o contador de programa
// PC sempre caminha ate o valor de getendereco e getendereco_op
// e so apos caminhar todos os endereco do vetor de instrucoes ele vai para a proxima linha
// as  diretivas  DB e DS possuem código de instrucao −1 e −2
// quando −1 é encontrado temos que atribuir ao endereco_op o conteúdo real
// quando −2 temos que incrementar contador PC quantas vezes o operando indicar
cont = 0;
int  contador_PC = 0;
int  pula_linha = 1;
arquivo_hexa.setf(ios :: uppercase);
arquivo_hexa « setw (2) «   setfill  ('0')« hex « contador_PC  <<" : ";
while (cont < inst_program.size()){
    celinst_geral  = inst_program[cont];
    // contador_PC deve pecorrer primeiro ate o endereco da instrucao e depois ate o endereco do oper
    while (contador_PC <= celinst_geral.getendereco()){
        if  (contador_PC >= (1∗pula_linha)){
            pula_linha++;
            arquivo_hexa « endl;
            arquivo_hexa « setw (2) «   setfill  ('0')«hex « contador_PC  <<" : ";
        }
        if  (contador_PC == celinst_geral.getendereco()){
            if  ( celinst_geral .getcodigo() == −1){
```

```
                arquivo_hexa « setw (2) «   setfill ('0')«hex «inst_program[cont].getoperando_geral(

                contador_PC++;
            }
            else if ( celinst_geral .getcodigo() !=-2){
                arquivo_hexa « setw (2) «   setfill ('0')«hex « inst_program[cont].getcodigo() << '
                contador_PC++;
            }
            else { // diretiva  DS
                int dec_DS = inst_program[cont].getoperando_geral();
                do{
                    if (contador_PC >= (1*pula_linha)){
                        pula_linha++;
                        arquivo_hexa «endl;
                        arquivo_hexa « setw (2) «   setfill ('0')«hex « contador_PC <<" : ";
                    }

                    arquivo_hexa <<"00"<<'\t';
                    contador_PC++;
                    dec_DS--;
                }while (dec_DS > 0);
            }

        } // fim  if
        else{
             arquivo_hexa <<"00" <<'\t';
             contador_PC++;
        }

    }
    while (contador_PC <= celinst_geral.getendereco_op()){
        if (contador_PC >= (1*pula_linha)){
            pula_linha++;
            arquivo_hexa «endl;
            arquivo_hexa « setw (2) «   setfill ('0')«hex « contador_PC <<" : ";
        }
        if (contador_PC == celinst_geral.getendereco_op()){
            arquivo_hexa « setw (2) «   setfill ('0')«hex «inst_program[cont].getoperando_geral() <
            contador_PC++;
        }
        else{
            arquivo_hexa <<"00"<<'\t';
```

```
                contador_PC++;
            }

        }
        cont++;
    } // fim while
    while (contador_PC < 256) {
        if (contador_PC >= (1*pula_linha)){
                pula_linha++;
                arquivo_hexa «endl;
                arquivo_hexa « setw (2) «  setfill ('0')«hex « contador_PC <<" : ";
        }
        else{
                 arquivo_hexa <<"00" <<'\t';
                 contador_PC++;
            }
    }

    arquivo_log << "Arquivo.hex: Construído"«endl;

    // montagem do arquivo.lst
    // utiliza-se o vetor vector <string> linha2;
    // este vetor possui todas as linhas do programa fonte se a existencia dos comentarios
    // utiliza-se tambem o vetor inst_program
    // e suas funcoes:
    //  getinst ()
    //  getcodigo()
    //  getendereco()
    //  getoperando_geral()
    cont = 0;
    int contador_linhas = 0;
    int contador_ant=0;
    int t;
    string str_lst ;
    int resp_lst;
    arquivo_lst.setf(ios::uppercase);
    while ( contador_linhas < inst_program.size()){
        celinst_geral = inst_program[contador_linhas];
        str_lst = celinst_geral.getinst (); // str_lst recebe a instrucao do vetor inst_program
        cont = contador_ant;
        while (cont < linha.size ()){
            resp_lst = linha[cont].find( str_lst ); // pocura nas linhas do programa a str_lst
```

```
        if (resp_lst > 0){ // se for encontrada verifica-se o codigo da instrução
            if (inst_program[contador_linhas].getcodigo() == -1){//se codigo for -1 que dizer que não
                // quando for DB temos que imprimir o endereco e seu conteúdo
                arquivo_lst << setw(2) << setfill('0')<< hex << inst_program[contador_linhas].getende
                arquivo_lst << linha[cont] << endl;

            }
                // abaixo temos que verificar a DS
            if (inst_program[contador_linhas].getcodigo() == -2){
                int dec_DS_lst = inst_program[contador_linhas].getoperando_geral();
                t = inst_program[contador_linhas].getendereco();
                arquivo_lst << t <<": 00"<<" " << "\t\t" << linha[cont] << endl;
                dec_DS_lst--;
                t++;
                while (dec_DS_lst > 0){
                    arquivo_lst << t <<": 00"<< endl;
                    dec_DS_lst--;
                    t++;
                }
            }

            if (inst_program[contador_linhas].getcodigo() > 0){
                if (inst_program[contador_linhas].getoperando_geral() >= 0){
                    arquivo_lst << setw(2) << setfill('0')<< hex << inst_program[contador_linhas].gete
                    arquivo_lst << linha[cont] << endl;
                }
                else {
                    arquivo_lst << setw(2) << setfill('0')<< hex << inst_program[contador_linhas].gete
                    arquivo_lst << linha[cont] << endl;
                }
            }
            contador_ant = cont+1;
            cont = linha.size();
        }
        else {
            arquivo_lst << '\t' << linha[cont] << endl;
            cont++;

        }
    }
    t = 0;
    contador_linhas++;
```

```
    }
    arquivo_lst << linha[linha.size()-1] << endl;
    arquivo_log << "Arquivo.lst: Construído"<<endl;

        // monta arquivo de instruções do programa fonte para ser utilizado no simulador
    /*
        string getinst();
    int getcodigo();
    int getendereco();
    int getoperando_geral();
    string getoperando();
    int getendereco_op();*/
    cont = 0;
    while (cont < inst_program.size()){
        arquivo_sml << inst_program[cont].getinst()<<" "<<inst_program[cont].getcodigo() << " " << inst
            << inst_program[cont].getoperando() << " " <<inst_program[cont].getendereco_op() << "
            << inst_program[cont].getoperando_geral()<<endl;
        cont++;
    }


} // fim montador


// a string é identificada como rotulo quando possui : no final da string
int monta::verifica_rotulo(string rotulo){

    string vr_rotulo;
    vr_rotulo = rotulo;
    int x = vr_rotulo.find(":");
    if (x < 0)
        return 0;
    else return 1;

} // fim verifica_rotulo

// verifica se a string é uma instrucao
celula_inst monta::verifica_instrucao(string instrucao){
    string vr_instrucao = instrucao;
    lista_inst lista;
    return (lista.procura_inst(vr_instrucao));
} // fim verifica_instrucao
```

```
// verifica se a string é uma diretiva
// se a string for uma diretiva o endereço é atualizado e retornado
int monta:: verifica_diretiva ( string diretiva , string op, int contador){
      string vr_diretiva = diretiva ;
      string vr_op = op;
      int vr_contador = contador ;
      lista_dire diretivas ;
      int teste;

      int resp_diretiva = diretivas .procura_dire( vr_diretiva );

      if ( resp_diretiva == 1){
           teste = diretivas . atualiza_dire ( vr_diretiva , atoi (vr_op.c_str ()), vr_contador);
           return teste;

      }
      else return (vr_contador);
} // fim verifica_diretiva

// vector <celula_token> t_linha;
// celula_token ( string , int );
// string getToken();
// int getLinha();
int monta::procura_linha_token(string pal){
      int tamanho = 0;
      string p_token;
      p_token = pal;
      int lin ;

      while (tamanho < t_linha.size ()){
           if (( t_linha [tamanho].getToken()) == p_token){
                lin = tamanho;
                tamanho = t_linha.size ();

           }
           else
                tamanho++;
      }

      return t_linha [ lin ]. getLinha ();
} // fim procura_linha_token
```

```
vector<celula_geral> monta::getVetor(){
    return inst_program;
}

string monta::getArquivoHex(){
    return simuArquivoHex;
}
#endif

// no arquivo.log
// erros descritos
//– verifica –se se o programa fonte foi finalizado , ou seja , verifica –se o END é a última string do vetor d
//–sucesso: Passo1 – Concluído
//– verifica –se os operandos: operandos validos sao menores que 256(FF)
//– operandos inválidos: label que nao estao na tabela de simbolos, montado no passo1, são verificados
//–se são digitos .
//–se for encontrado um caracter no meio dos digitos no arquivo.log teremos um aviso
//–se a cadeia de digitos for formada, mas for maior que 256, o operando também é inválido
//– verifica –se também o contador de programa, este deve ter somente 256 posições, pois a memória ende
//–sucesso: arquivo.map, arquivo.hex, arquivo.lst – Construídos
//– quando um operando invalido é detectado, no arquivo.log temos a descrição da linha onde o erro esta o
```

B.2 celula_simb.h

```
#ifndef CELULA_SIMB_H
#define CELULA_SIMB_H

#include <iostream>
#include <string>

class celula_simb{

public:

    celula_simb();
    void setcelula_simb(string label , int endereco);
    string getSimb();
    int getEnd();
```

```
private:
      string  label_simb;
      int  end_simb;

};

celula_simb::celula_simb(){

}

void celula_simb::setcelula_simb (string  label , int  endereco){
      label_simb = label;
      end_simb = endereco;
}

string  celula_simb::getSimb(){
      return label_simb;
}

int   celula_simb::getEnd(){
      return end_simb;
}

#endif
```

B.3 lista_inst.h

```
#ifndef LISTA_INST_H
#define LISTA_INST_H

#include <iostream>
#include <string>
#include <vector>
#include  "celula_inst.h"

class  lista_inst {

public:

       lista_inst  ();
```

```
        celula_inst  procura_inst( string   str );
        void imprimi ();

    private:
        vector<celula_inst> tab_instrucoes;
        celula_inst  inst_cod1;
        string  instrucao;
        int  codigo;

    };

    lista_inst  ::  lista_inst  (){
        inst_cod1.setInstrucao("LDA",9,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("ADI",5,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("ADD",32,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("SUBI",6,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("SUB",33,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("CMP",25,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("INC",7,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("DCC",8,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("LDI",9,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("STA",16,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("AND",17,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("OR",18,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("XOR",19,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("CLR",20,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("NOT",21,0);
        tab_instrucoes.push_back(inst_cod1);
```

```
        inst_cod1.setInstrucao("JMP",22,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JZ",130,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JC",145,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JS",167,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JEQ",229,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JNZ",179,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JNC",193,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JNS",214,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("JNEQ",244,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("CALL",23,1);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("RET",24,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("INA",34,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("IND",35,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("OUTA",36,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("OUTD",37,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("HLT",38,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("PUSH",39,0);
        tab_instrucoes.push_back(inst_cod1);
        inst_cod1.setInstrucao("POP",40,0);
        tab_instrucoes.push_back(inst_cod1);

}

// procura a string do programa na tabela de instrucões
// se a instrução for encontrada retorna o código da instrução
// senão retorna -1
```

```
celula_inst  lista_inst ::procura_inst(string  str ){
    string  lista_str  =  str ;
    string  str1 ;
    int  resp = 0;
    int  contador = 0;
    celula_inst  temp;

    //percorre a tabela de instruções a procura da instrução
    while ((resp == 0) && (contador < tab_instrucoes.size ())){
        temp = tab_instrucoes[contador];
        str1  = temp.getInstrucao();
        if  ( lista_str  == str1 )
            resp = 1;
        else contador++;
    }

    // verifica  se o codigo foi  encontrado ou não
    if  ((resp  == 1)||( contador < tab_instrucoes.size ())){
        return temp;
    }
    else{
        temp.setInstrucao(" ",-1,-1);
        return temp;
    }
}

void  lista_inst ::imprimi (){
    celula_inst  temp;
    for ( int  i  = 0;  i  < tab_instrucoes.size ();  i++){
        temp = tab_instrucoes[i ];
        cout  «  temp.getInstrucao()  <<  "  " « temp.getCodigo() « endl;

    }
}


#endif
```

B.4 lista_dire.h

```
#ifndef LISTA_DIRE_H
#define LISTA_DIRE_H
```

```
#include <iostream>
#include <string>
#include <vector>
#include "celula_dire.h"

class lista_dire {

public:

      lista_dire ();
      int procura_dire(string str );
      int atualiza_dire ( string str1 , int endereco, int op_seg);


private:
      vector<celula_dire> tab_dire;
      celula_dire dire_geral;

};

lista_dire :: lista_dire (){
      dire_geral. setdiretiva ("DB");
      tab_dire.push_back(dire_geral);
      dire_geral. setdiretiva ("DS");
      tab_dire.push_back(dire_geral);
}

// procura diretiva teremos que encontrar a diretiva correspondente
// a funcção procura ira receber 3 dados (diretiva procurada, endereço atual,
// e um inteiro )

int lista_dire :: procura_dire(string str ){

      string p_str = str;
      string str1;
      int resp = 0;
      int contador = 0;
      celula_dire temp ;

      // percorre a tabela de diretivas a procura da diretiva
      while ((resp == 0) && (contador < tab_dire.size ())){
```

```
            temp = tab_dire[contador];
            str1 = temp.getdiretiva ();
            if (p_str == str1)
                  return resp = 1;
            else contador++;
      }

      if (resp == 0)
            return -1;

}

int lista_dire :: atualiza_dire(string str1, int op_seg, int endereco){
      int p_end = endereco;
      int p_op = op_seg;
      string diretiva = str1;
      int resposta;

      if ( diretiva == "DB")
            resposta = p_end + 1;

      if ( diretiva == "DS")
            resposta = p_end + p_op;

      return resposta;

}

#endif
```

B.5 celula_inst.h

```
#ifndef CELULA_INST_H
#define CELULA_INST_H

#include <iostream>
#include <string>

class celula_inst{

public:
      celula_inst ();
```

```
        void setInstrucao(string instrucao, int codigo, int op);
        int getCodigo();
        int getOperando();
        string getInstrucao();


    private:
        string c_instrucao;
        int c_codigo;
        int c_op;

    };

    celula_inst :: celula_inst (){

    }

    void celula_inst :: setInstrucao(string instrucao, int codigo, int op){
        c_instrucao = instrucao;
        c_codigo = codigo;
        c_op = op;
    }

    int celula_inst :: getCodigo(){
        return c_codigo;
    }

    int celula_inst :: getOperando(){
        return c_op;
    }

    string celula_inst :: getInstrucao(){
        return c_instrucao;
    }

    #endif
```

B.6 celula_geral.h

```
// celula geral objetiva representar as instrucoes do programa a ser montado.
// cada celula contem a instrucao, o codigo e o endereco
// um vetor de celula geral é montado quando o programa é compilado
```

```
// apos armazenar todos os dados no vetor o vetor é consultado para montar o
// mapeamento do program


#ifndef CELULA_GERAL_H
#define CELULA_GERAL_H

#include <iostream>
#include <string>

class celula_geral{

public:

    celula_geral ();
    void setcelulageral( string instrucao, int codigo, int endereco, string operando = "@", int end_op
    string getinst ();
    int getcodigo();
    int getendereco();
    int getoperando_geral();
    string getoperando();
    int getendereco_op();
    void setoperandoreal(int op_real);


private:
    int op_real_geral;
    int codigo_geral;
    int end_geral;
    int end_operando;
    string op_geral;
    string inst_geral;



};

celula_geral :: celula_geral (){

}

void celula_geral :: setcelulageral ( string instrucao, int codigo, int endereco, string operando, int end_
```

```
        inst_geral  = instrucao;
        codigo_geral = codigo;
        end_geral = endereco;
        op_geral = operando;
        op_real_geral = operando_real;
        end_operando = end_op;
}

string  celula_geral :: getinst (){
        return inst_geral;
}

int   celula_geral :: getcodigo(){
        return codigo_geral;
}


int   celula_geral :: getendereco(){
        return end_geral;
}

string  celula_geral :: getoperando(){
        return op_geral;
}
int  celula_geral :: getendereco_op(){
        return end_operando;
}


int  celula_geral :: getoperando_geral(){
        return op_real_geral;
}


void celula_geral :: setoperandoreal(int op_real){
        op_real_geral = op_real;

}

#endif
```

B.7 celula_token.h

```
// a célula instrução constitui-se da string da instrução e inteiro representando o codigo
// duas funções: uma para setar a celula e outro para acessar o elementos da célula
// na diretiva temos que ter a string que representa a diretiva e quanto que o
// endereco deve retornar ou assumir

#ifndef CELULA_TOKEN_H
#define CELULA_TOKEN_H

#include <iostream>
#include <string>

class celula_token{

public:
    celula_token();
    void setToken(string palavra, int linha);
    string getToken();
    int getLinha();

private:
    string t_palavra;
    int num_linha;
};

celula_token::celula_token(){

}

void celula_token::setToken(string palavra, int linha){
    t_palavra = palavra;
    num_linha = linha;

}

string celula_token::getToken(){
    return t_palavra;
}

int celula_token::getLinha(){
```

```
    return num_linha;
}

#endif
```

Apêndice C

Exemplos em linguagem assembly para a UPDM

C.1 Soma de dois números

```
; Programa para somar dois números armazenados nas posições de
; memória 20d e 21d e armazenar o resultado na posição de memória 22d

    RESET: JMP INICIO       ; INICIO = 10


    TEMP EQU 100



            ORG 10
INICIO:     LDA OP1     ; primeiro operando (OP1 = 20d)
            ADD OP2     ;soma Op1 e Op2 (OP2 = 21d)
            STA SOMA    ; armazena o resultado (SOMA = 22d)
FIM:        HLT


            ORG 20
OP1:        DB      5   ; valor do primeiro operando
OP2:        DB      12  ; valor do segundo operando
SOMA:       DS      1   ; posição do resultado
            END
```

C.1.1 Soma de dois números - exemplosoma.hex

```
00 : 16
01 : 0A
02 : 00
03 : 00
04 : 00
05 : 00
06 : 00
07 : 00
08 : 00
09 : 00
0A : 09
0B : 14
0C : 20
0D : 15
0E : 10
0F : 16
10 : 26
11 : 00
12 : 00
13 : 00
14 : 05
15 : 0C
16 : 00
17 : 00
18 : 00
19 : 00
1A : 00
1B : 00
1C : 00
1D : 00
1E : 00
1F : 00
20 : 00
21 : 00
22 : 00
23 : 00
24 : 00
25 : 00
26 : 00
27 : 00
```

```
28 : 00
29 : 00
2A : 00
2B : 00
2C : 00
2D : 00
2E : 00
2F : 00
30 : 00
31 : 00
32 : 00
33 : 00
34 : 00
35 : 00
36 : 00
37 : 00
38 : 00
39 : 00
3A : 00
3B : 00
3C : 00
3D : 00
3E : 00
3F : 00
40 : 00
41 : 00
42 : 00
43 : 00
44 : 00
45 : 00
46 : 00
47 : 00
48 : 00
49 : 00
4A : 00
4B : 00
4C : 00
4D : 00
4E : 00
4F : 00
50 : 00
51 : 00
```

```
52 : 00
53 : 00
54 : 00
55 : 00
56 : 00
57 : 00
58 : 00
59 : 00
5A : 00
5B : 00
5C : 00
5D : 00
5E : 00
5F : 00
60 : 00
61 : 00
62 : 00
63 : 00
64 : 00
65 : 00
66 : 00
67 : 00
68 : 00
69 : 00
6A : 00
6B : 00
6C : 00
6D : 00
6E : 00
6F : 00
70 : 00
71 : 00
72 : 00
73 : 00
74 : 00
75 : 00
76 : 00
77 : 00
78 : 00
79 : 00
7A : 00
7B : 00
```

```
7C : 00
7D : 00
7E : 00
7F : 00
80 : 00
81 : 00
82 : 00
83 : 00
84 : 00
85 : 00
86 : 00
87 : 00
88 : 00
89 : 00
8A : 00
8B : 00
8C : 00
8D : 00
8E : 00
8F : 00
90 : 00
91 : 00
92 : 00
93 : 00
94 : 00
95 : 00
96 : 00
97 : 00
98 : 00
99 : 00
9A : 00
9B : 00
9C : 00
9D : 00
9E : 00
9F : 00
A0 : 00
A1 : 00
A2 : 00
A3 : 00
A4 : 00
A5 : 00
```

A6 : 00
A7 : 00
A8 : 00
A9 : 00
AA : 00
AB : 00
AC : 00
AD : 00
AE : 00
AF : 00
B0 : 00
B1 : 00
B2 : 00
B3 : 00
B4 : 00
B5 : 00
B6 : 00
B7 : 00
B8 : 00
B9 : 00
BA : 00
BB : 00
BC : 00
BD : 00
BE : 00
BF : 00
C0 : 00
C1 : 00
C2 : 00
C3 : 00
C4 : 00
C5 : 00
C6 : 00
C7 : 00
C8 : 00
C9 : 00
CA : 00
CB : 00
CC : 00
CD : 00
CE : 00
CF : 00

```
D0 : 00
D1 : 00
D2 : 00
D3 : 00
D4 : 00
D5 : 00
D6 : 00
D7 : 00
D8 : 00
D9 : 00
DA : 00
DB : 00
DC : 00
DD : 00
DE : 00
DF : 00
E0 : 00
E1 : 00
E2 : 00
E3 : 00
E4 : 00
E5 : 00
E6 : 00
E7 : 00
E8 : 00
E9 : 00
EA : 00
EB : 00
EC : 00
ED : 00
EE : 00
EF : 00
F0 : 00
F1 : 00
F2 : 00
F3 : 00
F4 : 00
F5 : 00
F6 : 00
F7 : 00
F8 : 00
F9 : 00
```

```
FA : 00
FB : 00
FC : 00
FD : 00
FE : 00
FF : 00
```

C.1.2 Soma de dois números - exemplosoma.lst

```
        ; Programa para somar dois números armazenados nas posições de
        ; memória 20d e 21d e armazenar o resultado na posição de memória 22d

00: 16 0A       RESET: JMP INICIO       ; INICIO = 10


            TEMP EQU 100



                    ORG 10
0A: 09 14       INICIO:     LDA OP1     ; primeiro operando (OP1 = 20d)
0C: 20 15                   ADD OP2     ;soma Op1 e Op2 (OP2 = 21d)
0E: 10 16                   STA SOMA    ; armazena o resultado (SOMA = 22d)
10: 26          FIM:        HLT


                    ORG 20
14: 05          OP1:        DB      5   ; valor do primeiro operando
15: 0C          OP2:        DB      12  ; valor do segundo operando
16: 00          SOMA:       DS      1   ; posição do resultado
                END
```

C.1.3 Soma de dois números - exemplosoma.map

```
RESET               00
TEMP            64
INICIO              0A
FIM             10
OP1             14
OP2             15
SOMA            16
```

C.2 Multiplicacao entre dois números

```
; Programa para multiplicar dois números armazenados nas posições de
; memória 80d e 81d e armazenar o resultado na posição de memória 83d

    RESET: JMP INICIO       ; INICIO = 10


                ORG 10
    INICIO:     LDA OP2     ; verifica  se é uma multiplicação por zero
                CMP 0
                JEQ MULT0
                CMP 1       ; verifica  se é uma multiplicação por um
                JEQ MULT1
    LOOP:       LDA OP1     ; inicia  o processo de multiplicação; carrega acumulador com o operando1
                ADD RESP    ;soma o acumulador com o resultado parcial
                STA RESP    ; atualiza  o resultado  parcial
                LDA OP2     ;decrementa o multiplicador, que corresponde ao operando 2
                DCC
                STA OP2     ; atualiza  o valor  do multiplicador
                CMP 0       ; verifica  se as somas sucessivas chegaram ao final
                JEQ RESF
                JMP LOOP




    MULT0:      LDI 0
                STA REST
                JMP FINAL

    MULT1:      LDA OP1
                STA REST
                JMP FINAL

    RESF:       LDA RESP
                STA REST
                JMP FINAL

    FINAL:      HLT
```

```
            ORG 80
OP1:        DB      2   ; valor do primeiro operando 1
OP2:        DB      2   ; valor do segundo operando 2
RESP:       DB      0   ; resultado parcial da multiplicação
REST:       DS      1   ; resultado final da multiplicação
            END
```

C.2.1 Multiplicacao entre dois números - exemplomult.hex

```
00 : 16
01 : 0A
02 : 00
03 : 00
04 : 00
05 : 00
06 : 00
07 : 00
08 : 00
09 : 00
0A : 09
0B : 51
0C : 19
0D : 00
0E : E5
0F : 25
10 : 19
11 : 01
12 : E5
13 : 2B
14 : 09
15 : 50
16 : 20
17 : 52
18 : 10
19 : 52
1A : 09
1B : 51
1C : 08
1D : 10
1E : 51
1F : 19
20 : 00
```

```
21 : E5
22 : 31
23 : 16
24 : 14
25 : 09
26 : 00
27 : 10
28 : 53
29 : 16
2A : 37
2B : 09
2C : 50
2D : 10
2E : 53
2F : 16
30 : 37
31 : 09
32 : 52
33 : 10
34 : 53
35 : 16
36 : 37
37 : 26
38 : 00
39 : 00
3A : 00
3B : 00
3C : 00
3D : 00
3E : 00
3F : 00
40 : 00
41 : 00
42 : 00
43 : 00
44 : 00
45 : 00
46 : 00
47 : 00
48 : 00
49 : 00
4A : 00
```

```
4B : 00
4C : 00
4D : 00
4E : 00
4F : 00
50 : 02
51 : 02
52 : 00
53 : 00
54 : 00
55 : 00
56 : 00
57 : 00
58 : 00
59 : 00
5A : 00
5B : 00
5C : 00
5D : 00
5E : 00
5F : 00
60 : 00
61 : 00
62 : 00
63 : 00
64 : 00
65 : 00
66 : 00
67 : 00
68 : 00
69 : 00
6A : 00
6B : 00
6C : 00
6D : 00
6E : 00
6F : 00
70 : 00
71 : 00
72 : 00
73 : 00
74 : 00
```

```
75 : 00
76 : 00
77 : 00
78 : 00
79 : 00
7A : 00
7B : 00
7C : 00
7D : 00
7E : 00
7F : 00
80 : 00
81 : 00
82 : 00
83 : 00
84 : 00
85 : 00
86 : 00
87 : 00
88 : 00
89 : 00
8A : 00
8B : 00
8C : 00
8D : 00
8E : 00
8F : 00
90 : 00
91 : 00
92 : 00
93 : 00
94 : 00
95 : 00
96 : 00
97 : 00
98 : 00
99 : 00
9A : 00
9B : 00
9C : 00
9D : 00
9E : 00
```

```
9F : 00
A0 : 00
A1 : 00
A2 : 00
A3 : 00
A4 : 00
A5 : 00
A6 : 00
A7 : 00
A8 : 00
A9 : 00
AA : 00
AB : 00
AC : 00
AD : 00
AE : 00
AF : 00
B0 : 00
B1 : 00
B2 : 00
B3 : 00
B4 : 00
B5 : 00
B6 : 00
B7 : 00
B8 : 00
B9 : 00
BA : 00
BB : 00
BC : 00
BD : 00
BE : 00
BF : 00
C0 : 00
C1 : 00
C2 : 00
C3 : 00
C4 : 00
C5 : 00
C6 : 00
C7 : 00
C8 : 00
```

C9 : 00
CA : 00
CB : 00
CC : 00
CD : 00
CE : 00
CF : 00
D0 : 00
D1 : 00
D2 : 00
D3 : 00
D4 : 00
D5 : 00
D6 : 00
D7 : 00
D8 : 00
D9 : 00
DA : 00
DB : 00
DC : 00
DD : 00
DE : 00
DF : 00
E0 : 00
E1 : 00
E2 : 00
E3 : 00
E4 : 00
E5 : 00
E6 : 00
E7 : 00
E8 : 00
E9 : 00
EA : 00
EB : 00
EC : 00
ED : 00
EE : 00
EF : 00
F0 : 00
F1 : 00
F2 : 00

```
F3 : 00
F4 : 00
F5 : 00
F6 : 00
F7 : 00
F8 : 00
F9 : 00
FA : 00
FB : 00
FC : 00
FD : 00
FE : 00
FF : 00
```

C.2.2 Multiplicacao entre dois números - exemplomult.lst

```
        ; Programa para multiplicar dois números armazenados nas posições de
        ; memória 80d e 81d e armazenar o resultado na posição de memória 83d

00: 16 0A       RESET: JMP INICIO       ; INICIO = 10


                       ORG 10
0A: 09 51       INICIO:     LDA OP2     ; verifica  se é uma multiplicação por zero
0C: 19 00                   CMP 0
0E: E5 25                   JEQ MULT0
10: 19 01                   CMP 1       ; verifica  se é uma multiplicação por um
12: E5 2B                   JEQ MULT1
14: 09 50       LOOP:       LDA OP1     ; inicia  o processo de multiplicação; carrega acumulador com o op
16: 20 52                   ADD RESP    ;soma o acumulador com o resultado parcial
18: 10 52                   STA RESP    ;atualiza  o resultado  parcial
1A: 09 51                   LDA OP2     ;decrementa o multiplicador, que corresponde ao operando 2
1C: 08                      DCC
1D: 10 51                   STA OP2     ;atualiza  o valor  do multiplicador
1F: 19 00                   CMP 0       ; verifica  se as somas sucessivas chegaram ao final
21: E5 31                   JEQ RESF
23: 16 14                   JMP LOOP




25: 09 00       MULT0:      LDI 0
```

```
27: 10 53                STA REST
29: 16 37                JMP FINAL

2B: 09 50      MULT1:    LDA OP1
2D: 10 53                STA REST
2F: 16 37                JMP FINAL

31: 09 52      RESF:     LDA RESP
33: 10 53                STA REST
35: 16 37                JMP FINAL

37: 26         FINAL:    HLT


                  ORG 80
50: 02         OP1:      DB    2    ; valor do primeiro operando 1
51: 02         OP2:      DB    2    ; valor do segundo operando 2
52: 00         RESP:     DB    0    ; resultado parcial da multiplicação
53: 00         REST:     DS    1    ; resultado final da multiplicação
               END
```

C.2.3 Multiplicacao entre dois números - exemplomult.map

```
RESET          00
INICIO         0A
LOOP       14
MULT0          25
MULT1          2B
RESF       31
FINAL          37
OP1        50
OP2        51
RESP       52
REST       53
```

Apêndice D

Programa Simulador

D.1 Código do Programa Simulador

O código do programa simulador foi desenvolvido juntamente com o desenvolvimento da interface gráfica do Sistema Integrado.

A simulação é iniciada com a seleção de um arquivo.hex, gerado pelo processo de montagem.

```
// Projeto : Desenvolvimento de um Sistema Integrado de Compilação, MOntagem e
//           Simulação para Programação de um Processador Discreto Microprogramável
// Orientada: Edna Mie Kanazawa
// Orientador: Wilian  Soares Lacerda
// Período de desenvolvimento: agosto/2000 a julho/2001

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_Simulador.h"
#include "U_EntradaDigital.h"
#include "U_EntradaAnalogica.h"
#include "celula_simb.h"
#include "lista_inst.h"
#include "lista_dire.h"
#include "celula_inst.h"
#include "celula_geral.h"
#include "celula_token.h"
#include "celula_hex.h"
```

```
#include "simul_tab.h"
#include "celu_simul.h"
#include <fstream>
#include <iomanip>
#include <vector>
#include <cstdio>

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormSimulador *FormSimulador;
//---------------------------------------------------------------------------
__fastcall TFormSimulador::TFormSimulador(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::BitBtnFecharClick(TObject *Sender)
{
    FormSimulador->Close();
    simu_inst.clear ();
    EditAC->Text = " ";
    EditCP->Text = " ";
    EditAP->Text = " ";
    EditEA->Text = " ";
    EditED->Text = " ";
    EditSA->Text = " ";
    EditSD->Text = " ";
    EditRE->Text = " ";
    EditRI->Text = " ";
    EditRB->Text = " ";

    DBCheckBoxZero->Checked = false;
    DBCheckBoxIgual->Checked = false;
    DBCheckBoxVaiUm->Checked = false;
    DBCheckBoxSinal->Checked = false;

    BitBtnRun->Enabled = true;
    BitBtnPassoaPasso->Enabled = true;

    ListBoxHex->Items->Delete(ListBoxHex->ItemIndex);
    //FormEntradaDigital->EditEntradaDigital->Text = " ";
```

```
    simu_inst.clear ();
    end_cont.clear();



}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::RecebeArquivoHex(String ArqHex){
        // RichEditHex->Lines->LoadFromFile(ArqHex);
        GroupBoxHex->Caption = ExtractFileName(ArqHex);
}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::RecebeArquivoLst(String ArqLst){
        RichEditLst->Lines->LoadFromFile(ArqLst);
        GroupBoxLst->Caption = ExtractFileName(ArqLst);

}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::RecebeArquivoSmlHex(String ArqSml,String ArqHexa){

   /*vector <celula_geral> simu_inst;
   celula_geral simu_celula;
   celula_hex cel_endcont;
   celula_hex inter;
   vector <celula_hex> end_cont  ;*/
   arquivohexadecimal = ArqHexa;
   ifstream arquivo_hex (ArqHexa.c_str());

   string sep;
   int endereco;
   int conteudo;
   string endhex;
   string conthex;

   while (!arquivo_hex.eof()){

        arquivo_hex » hex » endereco » sep » hex » conteudo;
        cel_endcont.setcelula_hex(endereco, conteudo);
        end_cont.push_back(cel_endcont);
        arquivo_hex.get();
    }
    arquivo_hex.close();
```

```
string strgeral;
ifstream arquivohex (ArqHexa.c_str());
while (!arquivohex.eof()){

    arquivohex » endhex » sep » conthex;
    strgeral = endhex+": "+conthex;
    ListBoxHex->Items->Add(strgeral.c_str());
    arquivohex.get();
}
arquivohex.close();

//criação de vetor de célula geral inst_program
//   vector <celula_geral> simu_inst;
//celula_geral simu_celula;
//   void setcelulageral(string instrucao, int codigo,
//  int endereco, string operando = "@", int end_op = -1, int operando_real = -1);
/*
    arquivo_sml « inst_program[cont].getinst ()«" "
                « inst_program[cont].getcodigo () « " "
                « inst_program[cont].getendereco () « " "
                « inst_program[cont].getoperando () « " "
                « inst_program[cont].getendereco_op () « " "
                « inst_program[cont].getoperando_geral()«endl;
*/
ifstream arquivo_sml (ArqSml.c_str());
string instrucao_sml;
int codigo_sml;
int endereco_sml;
string operando_sml;
int op_end_sml;
int op_real_sml;



while (!arquivo_sml.eof()){

arquivo_sml » instrucao_sml » codigo_sml » endereco_sml » operando_sml » op_end_sml » op_real
simu_celula.setcelulageral(instrucao_sml, codigo_sml , endereco_sml , operando_sml , op_end_sml , o
simu_inst.push_back(simu_celula);
arquivo_sml.get();
```

```
    }

}

//---------------------------------------------------------------------------
void __fastcall TFormSimulador::executa(int simul_acao){
    int exec_acao = simul_acao;
    int endereco_int;
    String teste;
    // instrução LDA, ACC <- [endereco]
    // endereco = simu_inst[cont].getoperando_geral();
    // percorre o arquivo.hex e verifica a linha do endereco dada pelo linha-1
    // é pego seu conteúdo e colocado no acumulador
    switch (exec_acao){
    case 1:
        RE = CP;//registrador de endereços recebe o conteúdo do contador de programas
        AtualizaRegistradores(); // atualiza os valores dos registradores
        RE = simu_inst[cont].getoperando_geral();//o registrador de endereços recebe o operando do instru
        AtualizaRegistradores(); // atualiza os valores dos registradores
        ACC = procura_cont(simu_inst[cont].getoperando_geral());//o acumulador recebe o conteúdo do enc
        AtualizaRegistradores(); // atualiza os valores dos registradores
        CP = CP + 1; //contador de programa aponta para a próxima instrução
        AtualizaRegistradores(); // atualiza os valores dos registradores
        break;
    // instrução: ADI <dado> <- ACC = ACC + dado
    case 2:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,simu_inst[cont].getoperando_geral());
        ACC = ACC + simu_inst[cont].getoperando_geral();
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: ADD <endereço> <- ACC = ACC + [endereco]
    case 3:
```

```
        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        RB =  procura_cont(simu_inst[cont].getoperando_geral());
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,procura_cont(simu_inst[cont].getoperando_geral()));
        ACC = ACC + procura_cont(simu_inst[cont].getoperando_geral());
        ativa_flag_sinal (ACC);
        if  (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;

   // instrução : SUBI <dado> <– ACC = ACC – dado
  case 4:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,(~(simu_inst[cont].getoperando_geral())));
        ACC = ACC – simu_inst[cont].getoperando_geral();
        ativa_flag_sinal (ACC);
        if  (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
   // instrução : SUB <endereco> <– ACC = ACC – [endereco]
   case 5:
        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        RB =  procura_cont(simu_inst[cont].getoperando_geral());
        AtualizaRegistradores();
        ativa_flag_vaium(ACC,(~(procura_cont(simu_inst[cont].getoperando_geral()))));
```

```
        ACC = ACC – procura_cont(simu_inst[cont].getoperando_geral());
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: CMP <dado>
    // compara ACC com o dado e seta flag igual se forem iguais
    case 6:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        if (ACC == simu_inst[cont].getoperando_geral())
            flag_igual = 1;
        else flag_igual = 0;
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: INC <– ACC+1
    case 7:
        ativa_flag_vaium(ACC,1);
        ACC = ACC + 1;
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        break;
    // instrução:DCC <– ACC–1
    case 8:
        ativa_flag_vaium(ACC,~1);
        ACC = ACC – 1;
        ativa_flag_sinal (ACC);
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
```

```
        break;
    //instrução:LDI <dado> <– ACC = dado
    case 9:
        RE = CP;
        AtualizaRegistradores();
        ACC = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    //instrução: STA <endereço> <– endereço = ACC
    case 10:

        RE = CP;
        AtualizaRegistradores();
        RE = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        FormSimulador->ListBoxHex->Items->Delete(RE);
        teste = converte_hexa(RE)+" : "+converte_hexa(ACC);
        FormSimulador->ListBoxHex->Items->Insert(RE, teste);
        //procura o endereço no vetor do arquivo hex
        endereco_int = procura_indice(simu_inst[cont].getoperando_geral());
        //seta seu novo conteúdo
        end_cont[endereco_int].setConteudo(ACC);
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    //instrução: AND <dado> <– ACC = ACC and dado
    case 11:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC & simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    //instrução: OR <dado> <– ACC = ACC or dado
    case 12:
        RE = CP;
        AtualizaRegistradores();
```

```
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC | simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: XOR <dado> <– ACC = ACC xor dado
    case 13:
        RE = CP;
        AtualizaRegistradores();
        RB = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        ACC = ACC ^ simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        break;
    // instrução: CLR <– ACC = 0
    case 14:
        ACC = 0;
        if (ACC == 0)
            flag_zero = 0;
        else flag_zero = 1;
        AtualizaRegistradores();
        break;
    // instrução: NOT <– ACC = /ACC
    case 15:
        ACC = ~ACC;
        AtualizaRegistradores();
        break;
    // instrução: JMP <endereco> <– CP = endereco
    case 16:
        RE = CP;
        AtualizaRegistradores();
        CP = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        break;
    // instrução: JZ <endereço> <– if /ZERO = 0 CP = endereco
    // else CP = CP++;
    case 17:
        RE = CP;
        AtualizaRegistradores();
        if (flag_zero == 0){
```

```
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else {
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    //instrução: JC <endereço> <– if /VAI–UM = 0 CP = endereco
    //else CP = CP++;
    case 18:
        RE = CP;
        AtualizaRegistradores();
        if (flag_vai_um == 0){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    //instrução: JS <endereço> <– if SINAL = 1 CP = endereco
    //else CP = CP++;
    case 19:
        RE = CP;
        AtualizaRegistradores();
        if (flag_sinal == 1){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
             CP = CP + 1;
             AtualizaRegistradores();
        }
        break;
    //instrução: JEQ <endereço> <– if IGUAL = 1 CP = endereco
    //else CP = CP++;
    case 20:
        RE = CP;
        AtualizaRegistradores();
        if (flag_igual == 1){
```

```
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else {
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    // instrução : JNZ <endereço> <– if /ZERO = 1 CP = endereco
    // else CP = CP++;
    case 21:
        RE = CP;
        AtualizaRegistradores();
        if (flag_zero == 1){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    // instrução : JNC <endereço> <– if /VAI–UM = 1 CP = endereco
    // else CP = CP++;
    case 22:
        RE = CP;
        AtualizaRegistradores();
        if (flag_vai_um == 1){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    // instrução : JNS <endereço> <– if SINAL = 0 CP = endereco
    // else CP = CP++;
    case 23:
        RE = CP;
        AtualizaRegistradores();
        if ( flag_sinal == 0){
```

```
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    //instrução: JNEQ <endereço> <– if IGUAL = 0 CP = endereco
    //else CP = CP++;
    case 24:
        RE = CP;
        AtualizaRegistradores();
        if ( flag_igual == 0){
            CP = simu_inst[cont].getoperando_geral();
            AtualizaRegistradores();
        }
        else{
            CP = CP + 1;
            AtualizaRegistradores();
        }
        break;
    //instrução: CALL <endereço> <– [AP] = CP
    //CP = endereço   AP <– AP – 1;
    case 25:
        RE = AP;
        AtualizaRegistradores();
        RE = CP;
        simu_inst[cont].getoperando_geral();
        inter.setcelula_hex(AP, CP);
        end_cont.push_back(inter);
        CP = simu_inst[cont].getoperando_geral();
        AtualizaRegistradores();
        AP = AP –1;
        AtualizaRegistradores();
        break;
    //instrução: RET <– CP = [AP]
    //AP = AP+1
    case 26:
        AP = AP + 1;
        AtualizaRegistradores();
        RE = AP;
```

```
        CP = procura_cont(AP);
        AtualizaRegistradores();
        CP = CP + 1;
        AtualizaRegistradores();
        break;
    // instrução: INA <– ACC = entrada analógica
    case 27:
        Application->CreateForm(__classid(TFormEntradaAnalogica), &FormEntradaAnalogica);
        FormEntradaAnalogica->ShowModal();
        ACC = entrada_analogica;
        AtualizaRegistradores();
        break;
    // instrução: IND <– ACC = entrada digital
    case 28:
        Application->CreateForm(__classid(TFormEntradaDigital), &FormEntradaDigital);
        FormEntradaDigital->ShowModal();
        ACC = entrada_digital;
        AtualizaRegistradores();
        break;
    // instrução: OUTA <– saida analógica = ACC
    case 29:
        saida_analogica = ACC;
        AtualizaRegistradores();
        break;
    // instrução: OUTD <– saida digital = ACC

    case 30:
        saida_digital  = ACC;
        AtualizaRegistradores();
        break;
    // instrução: HALT
    case 31:
        cont_r = simu_inst.size();
        cont_pp = simu_inst.size();
        break;
    // instrução: PUSH
    case 32:
        RE = AP;
        AtualizaRegistradores();
        FormSimulador->ListBoxHex->Items->Delete(RE);
        teste = converte_hexa(RE)+" : "+converte_hexa(ACC);
        FormSimulador->ListBoxHex->Items->Insert(RE, teste);
```

```
            // procura o endereço no vetor do arquivo hex
            endereco_int = procura_indice(AP);
            // seta seu novo conteúdo
            end_cont[endereco_int].setConteudo(ACC);
            AP = AP – 1;
            AtualizaRegistradores();
            break;
        // instrução POP
        case 33:
            AP = AP + 1;
            AtualizaRegistradores();
            RE = AP;
            AtualizaRegistradores();
            ACC = procura_cont(procura_indice(AP));
            AtualizaRegistradores();
            break;
        } // fim switch
}

//  vetor de células e conteúdos
//  celula_hex cel_endcont;
//  vector <celula_hex> end_cont;

int __fastcall TFormSimulador::procura_cont(int endere){
    int parada;
    for (int i = 0; i < end_cont.size(); i++){
        if (endere == end_cont[i].getEnd()){
            parada = i;
            i = end_cont.size();

        }
    }
    return (end_cont[parada].getConteudo());
}

int __fastcall TFormSimulador::procura_indice(int endere){
    int parada;
    for (int i = 0; i < end_cont.size(); i++){
        if (endere == end_cont[i].getEnd()){
            parada = i;
            i = end_cont.size();
```

```
            }
        }
        return (parada);
}

void __fastcall TFormSimulador::ativa_flag_vaium(int op1, int op2){
        int operando1[8];
        int operando2[8];
        int result [9];
        result [0] = 0;

        for ( int i = 0; i < 8; i++){
            operando1[i] = op1%2;
            op1 = op1/2;
        }

        for ( int i = 0; i < 8; i++){
            operando2[i] = op2%2;
            op2 = op2/2;
        }

        for ( int i = 0; i < 8; i++)
            result [ i +1] = ( result [ i ] &( operando1[i] ^ operando2[i ])) | ( operando1[i] & operando2[i]);

        flag_vai_um = result [8];


}

void __fastcall TFormSimulador::ativa_flag_sinal(int op1){
        int operando1[8];

        for ( int i = 0; i < 8; i++){
            operando1[i] = op1%2;
            op1 = op1/2;
        }

        flag_sinal = operando1[7];

}

void __fastcall TFormSimulador::BitBtnRunClick(TObject *Sender)
```

```
{
    cont_r = 0;
    ACC = 0;
    AP = 255;
    entrada_analogica = 0;
    entrada_digital  = 0;
    saida_analogica = 0;
    saida_digital  = 0;
    CP_ant = 0;
    cont = 0;

    // int  acao;
    // string  str;
    // simul_tab acao_tab;

    while (cont_r < simu_inst.size ()){
        simu_celula = simu_inst[cont_r]; // celula_geral
        cont = cont_r;
        str = simu_celula.getinst ();
        if (( str != "DB") || (str != "DS")){
            acao = acao_tab.procura_simul(str);
            RE = CP;//primeiro ciclo de busca
            AtualizaRegistradores();
            CP = CP + 1; //segundo ciclo de busca
            AtualizaRegistradores();
            RI = simu_celula.getcodigo();
            AtualizaRegistradores(); // terceiro ciclo de busca
            for (int i = 0; i < 1000; i++){
            }
            CP_ant = CP;
            executa(acao); // executa cada instrução
            if (CP != CP_ant+1){
                cont_r = procuraCP(CP);
            }
            else{
                AtualizaRegistradores();
                cont_r++;
            }
        }
        else {
            cont_r++;
        }
```

```
    }
    BitBtnRun->Enabled = false;
    BitBtnPassoaPasso->Enabled = false;
}
//---------------------------------------------------------------------------
int __fastcall TFormSimulador::procuraCP (int end){
    int i = 0;
    while (i < simu_inst.size ()){
     if (simu_inst[i].getendereco() == end){
        return i;
     }
     else i++;
    }
}
//---------------------------------------------------------------------------
String __fastcall TFormSimulador::converte_hexa (int valor){
    vector <int> inteiros ;
    String num;
    if (valor < 10){
        num = "0" + IntToStr(valor);
        return num;
    }
    else{
        do{
            inteiros.push_back(valor % 16);
            valor = valor / 16;
        }while (valor != 0);


        for(int i = 0; i < inteiros.size (); i++){
            if ( inteiros [ inteiros.size() - i -1] < 10)
                num = num + IntToStr(inteiros[ inteiros.size() - i -1]);
            else{
                switch(inteiros[ inteiros.size() - i -1]){
                    case 10: num = num + "A";
                             break;
                    case 11: num = num + "B";
                             break;
                    case 12: num = num + "C";
                             break;
                    case 13: num = num + "D";
                             break;
```

```
                        case 14: num = num + "E";
                                 break;
                        case 15: num = num + "F";
                                 break;
                        } // fim switch

                    } // fim else
                } // fim for
                if (num.Length()==1)
                    num = "0"+num;
                return num;
            } // fim else

}
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::RadioGroupValorClick(TObject *Sender)
{
 if (RadioGroupValor -> ItemIndex == 0){
     EditAC->Text = IntToStr(ACC);
     EditCP->Text = IntToStr(CP);
     EditAP->Text = IntToStr(AP);
     EditEA->Text = IntToStr(entrada_analogica);
     EditED->Text = IntToStr(entrada_digital);
     EditSA->Text = IntToStr(saida_analogica);
     EditSD->Text = IntToStr(saida_digital);
     EditRE->Text = IntToStr(RE);
     EditRI->Text = IntToStr(RI);
     EditRB->Text = IntToStr(RB);
}
 if (RadioGroupValor -> ItemIndex == 1){
    EditAC->Text = converte_hexa(ACC);
    EditCP->Text = converte_hexa(CP);
    EditAP->Text = converte_hexa(AP);
    EditEA->Text = converte_hexa(entrada_analogica);
    EditED->Text = converte_hexa(entrada_digital);
    EditSA->Text = converte_hexa(saida_analogica);
    EditSD->Text = converte_hexa(saida_digital);
    EditRE->Text = converte_hexa(RE);
    EditRI->Text = converte_hexa(RI);
    EditRB->Text = converte_hexa(RB);
}
}
```

```
//---------------------------------------------------------------------------

void __fastcall TFormSimulador::BitBtnResetClick(TObject *Sender)
{
    cont_r = 0;
    cont_pp = 0;
    ACC = 0;
    AP = 255;
    CP = 0;
    RE = 0;
    RI = 0;
    RB = 0;
    entrada_analogica = 0;
    entrada_digital = 0;
    saida_analogica = 0;
    saida_digital = 0;
    BitBtnPassoaPasso->Enabled = true;
    BitBtnRun->Enabled = true;
    flag_zero = 0; // flag /zero
    flag_vai_um = 0; // flag /vai-um
    flag_igual = 0; // flag igual
    flag_sinal = 0; // flag sinal
    AtualizaRegistradores();

    ListBoxHex->Clear();
    // FormEntradaDigital->EditEntradaDigital->Text = " ";
    string strgeral ;
    string endhex, conthex;
    string sep;
    ifstream arquivohex (arquivohexadecimal.c_str());
    while (!arquivohex.eof()){

        arquivohex » endhex » sep » conthex;
        strgeral = endhex+" : "+conthex;
        ListBoxHex->Items->Add(strgeral.c_str());
        arquivohex.get();
    }
    arquivohex.close();
    end_cont.clear();
    ifstream arquivo_hex (arquivohexadecimal.c_str());
    int endereco;
    int conteudo;
```

```
    while (!arquivo_hex.eof()){

        arquivo_hex » hex » endereco » sep » hex » conteudo;
        cel_endcont.setcelula_hex(endereco, conteudo);
        end_cont.push_back(cel_endcont);
        arquivo_hex.get();
    }
    arquivo_hex.close();


}
//---------------------------------------------------------------------------

void __fastcall TFormSimulador::BitBtnPassoaPassoClick(TObject *Sender)
{
    BitBtnRun -> Enabled = false; //se a opção passo a passo o botão run é desbilitado até o término do p
    simu_celula = simu_inst[cont_pp]; //celula_geral contém os dados de cada instrução do programa fonte
    cont = cont_pp;//seta o contador_principal com o contador do passo a passo.
    str = simu_celula.getinst ();
    if (( str != "DB") || (str != "DS")){
        acao = acao_tab.procura_simul(str);
        RE = CP;//primeiro ciclo de busca
        AtualizaRegistradores();
        CP = CP + 1; //segundo ciclo de busca
        AtualizaRegistradores();
        RI = simu_celula.getcodigo();
        AtualizaRegistradores(); // terceiro ciclo de busca
        executa(acao); //executa cada instrução
        if (CP != CP_ant+1){
            cont_pp = procuraCP(CP);
        }
        else{
            AtualizaRegistradores();
            cont_pp++;
        }
    }
    else cont_pp++;
    if (cont_pp >= simu_inst.size())
        BitBtnPassoaPasso->Enabled = false;

}
```

```
//---------------------------------------------------------------------------
void __fastcall TFormSimulador::AtualizaRegistradores(){
int i = 0;
while (i < 1000000)//atraso
      i++ ;
if (RadioGroupValor -> ItemIndex == 0){
    EditAC->Text = IntToStr(ACC);
    EditCP->Text = IntToStr(CP);
    EditAP->Text = IntToStr(AP);
    EditEA->Text = IntToStr(entrada_analogica);
    EditED->Text = IntToStr(entrada_digital);
    EditSA->Text = IntToStr(saida_analogica);
    EditSD->Text = IntToStr(saida_digital);
    EditRE->Text = IntToStr(RE);
    EditRI->Text = IntToStr(RI);
    EditRB->Text = IntToStr(RB);
}
if (RadioGroupValor -> ItemIndex == 1){
     EditAC->Text = converte_hexa(ACC);
     EditCP->Text = converte_hexa(CP);
     EditAP->Text = converte_hexa(AP);
     EditEA->Text = converte_hexa(entrada_analogica);
     EditED->Text = converte_hexa(entrada_digital);
     EditSA->Text = converte_hexa(saida_analogica);
     EditSD->Text = converte_hexa(saida_digital);
     EditRE->Text = converte_hexa(RE);
     EditRI->Text = converte_hexa(RI);
     EditRB->Text = converte_hexa(RB);
}

ListBoxHex->ItemIndex = RE;

DBCheckBoxZero->Checked = flag_zero;
DBCheckBoxIgual->Checked = flag_igual;
DBCheckBoxVaiUm->Checked = flag_vai_um;
DBCheckBoxSinal->Checked = flag_sinal;
}
//---------------------------------------------------------------------------
```

Apêndice E

Analisador léxico

E.1 Código analisador léxico

```
/*
Projeto : Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
Fase 3: Desenvolvimento do Compilador – Analisador Léxico
Desenvolvedor: Edna Mie Kanazawa
Orientador: Wilian Soares Lacerda
*/

%{
#include <iostream>
#include <cmath>
#include <fstream>
#include <string>
using namespace std;


#define MAIOR 1
#define MENOR 2
#define IGUAL 3
#define NEG 4
#define MAIORIGUAL 5
#define MENORIGUAL 6
#define DIFER 7
#define ADICAO 8
#define SUBTRACAO 9
#define MULT 10
```

```
#define DIVISAO 11
#define AND 12
#define OR 13
#define ID 15
#define NUM 16
#define IF 17
#define ELSE 19
#define WHILE 20
#define DO    21
#define CASE   35
#define SWITCH 22
#define RETURN 23
#define BREAK   24
#define ABPAR 25
#define FEPAR 26
#define ABCH 27
#define FECH 28
#define PVIR 29
#define PONTO 30
#define VIRG 31
#define STRING_LITERAL 32
#define SCANF 33
#define PRINTF 34
#define CTE     36
#define OPAD   37
#define OPREL  38
#define OPNEG 39
#define ASPAS 40
#define INVALIDO 41
#define MAIN   42
#define INT      43
#define CHAR  44
#define ATRIB  45
#define CARACTER 46
#define CARACINV 47
#define ERR    48


%}

delim     [ \ t\n]
```

```
ws        {delim}+
digito    [0-9]
letra     [A-Za-z]
num       {digito}+
id        {letra}({letra}|{digito})*
idinvalido  {digito}({letra}|{digito})*


%%
{ws}      {}
"/*"      {register int c;
          for (;;){
               while ((c = yyinput()) != '*' && c!=EOF);
               if (c == '*'){
                    while((c = yyinput()) =='*');
                    if (c=='/')
                         break;
               }
               if (c = EOF){
                    return ERR;//erros « " arquivo terminado pelo comentario"«endl;
                    break;
               }

          }
          }
main      {return MAIN;}
if        {return IF;}
else      {return ELSE;}
while     {return WHILE;}
do        {return DO;}
case      {return CASE;}
switch    {return SWITCH;}
return    {return RETURN;}
break     {return BREAK;}
printf    {return PRINTF;}
scanf     {return SCANF;}
int       {return INT;}
char      {return CHAR;}
{id}      {return ID;} // se o átomo for um identificador o tipo = ID e o atributo =cadeia de caracteres
{num}     {return NUM;}//se o átomo for um cadeia de números a tipo será CTE e atributo é a cadeia de núm
{idinvalido} {return INVALIDO;}
"\""{id}"\""   {return(STRING_LITERAL); }
```

```
"\'"{letra}"\'" {return(CARACTER); }
"\'"{id}"\'" {return CARACINV;}
"<"     {return MENOR;}
">"     {return MAIOR;}
"="     {return ATRIB;}
"=="        {return IGUAL;}
"~"     {return NEG;}
"<="    {return MENORIGUAL;}
">="    {return MAIORIGUAL;}
"!="    {return DIFER;}
"+"     {return ADICAO;}
"-"     {return SUBTRACAO;}
"*"     {return MULT;}
"/"     {return DIVISAO;}
"&&"        {return AND;}
"||"        {return OR;}
"("     {return ABPAR;}
")"     {return FEPAR;}
"{"     {return ABCH;}
"}"     {return FECH;}
";"     {return PVIR;}
"."     {return PONTO;}
","     {return VIRG;}


%%

int main(int argc, char *argv[]) {
    FILE *f_in;
    if (argc == 2) {
        if (f_in = fopen(argv[1],"r"))
            yyin = f_in;
        else
            perror(argv[0]);
        }
    else
        yyin = stdin;

    int i;
    string path = argv[1];

    path = path.erase(path.find_last_of("\\")+1);
```

```
string cad; // string utilizada para cadeia

string caminho = path + "tabelalexica.txt";
string caminho_erros = path + "erros.txt";


ofstream lexica(caminho.c_str()); // tabela léxica atomo/tipo/ atributo
ofstream erros(caminho_erros.c_str()); // arquivo de erros

lexica <<"atomo" << "\t" "tipo " << "\t" << "Atributo" « endl;


while (i = yylex ()){

    switch (i ) {
    case 1:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "MAIOR" « endl;
        break;
    case 2:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "MENOR" « endl;
        break;
    case 3:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "IGUAL" « endl;
        break;
    case 4:
        lexica « yytext <<"\t"<< "OPNEG" << "\t" << "NEG" « endl;
        break;
    case 5:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "MAIORIGUAL" « endl;
        break;
    case 6:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "MENORIGUAL" « endl;
        break;
    case 7:
        lexica « yytext <<"\t"<< "OPREL" << "\t" << "DIFER" « endl;
        break;
    case 8:
        lexica « yytext <<"\t"<< "OPAD" << "\t" << "ADICAO" « endl;
        break;
    case 9:
        lexica « yytext <<"\t"<< "OPAD" << "\t" << "SUBTRACAO" « endl;
```

```
        break;
    case 10:
        lexica « yytext <<"\t"<< "OPMULT" << "\t" << "MULT" « endl;
        break;
    case 11:
        lexica « yytext <<"\t"<< "OPMULT" << "\t" << "DIVISAO" « endl;
        break;
    case 12:
        lexica « yytext <<"\t"<< "OPMULT" << "\t" << "AND" « endl;
        break;
    case 13:
        lexica « yytext <<"\t"<< "OPMULT" << "\t" << "OR" « endl;
        break;
    case 15:
        lexica « yytext <<"\t"<< "ID" << "\t" « yytext « endl;
        break;
    case 16:
        if ( atoi (yytext) > 255)
            erros « yytext << " : numero muito extenso" « endl;
        else
            lexica « yytext <<"\t"<< "CTE" << "\t" « yytext « endl;
        break;
    case 17:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 19:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 20:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 21:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 22:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 23:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
        break;
    case 24:
        lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
```

```
    break;
case 25:
    lexica « yytext <<"\t"<< "ABPAR"<< "\t" << "@" « endl;
    break;
case 26:
    lexica « yytext <<"\t"<< "FEPAR"<< "\t" << "@" « endl;
    break;
case 27:
    lexica « yytext <<"\t"<< "ABCH"<< "\t" << "@" « endl;
    break;
case 28:
    lexica « yytext <<"\t"<< "FECH"<< "\t" << "@" « endl;
    break;
case 29:
    lexica « yytext <<"\t"<< "PVIRG"<< "\t" << "@" « endl;
    break;
case 30:
    lexica « yytext <<"\t"<< "PONTO"<< "\t" << "@" « endl;
    break;
case 31:
    lexica « yytext <<"\t"<< "VIRG"<< "\t" << "@" « endl;
    break;
case 32:
    cad = yytext;
    cad = cad.substr(1,cad.length()-2);
    lexica « yytext <<"\t"<< "CADEIA"<< "\t" « cad « endl;
    break;
case 33:
    lexica « yytext <<"\t"<< "SCANF"<< "\t" << "@" « endl;
    break;
case 34:
    lexica « yytext <<"\t"<< "PRINTF"<< "\t" << "@" « endl;
    break;
case 35:
    lexica « yytext <<"\t"« yytext << "\t" << "@" « endl;
    break;
case 40:
    lexica « yytext <<"\t"<< "ASPAS"<< "\t" << "@" « endl;
    break;
case 41:
    lexica « yytext <<"\t"<< "ID" << "\t" « yytext « endl;
    erros « yytext << " : identificador inválido" « endl;
```

```
            break;
        case 42:
            lexica « yytext <<"\t"<< "MAIN"<< "\t" << "@" « endl;
            break;
        case 43:
            lexica « yytext <<"\t"<< "INT"<< "\t" << "@" « endl;
            break;
        case 44:
            lexica « yytext <<"\t"<< "CHAR"<< "\t" << "@" « endl;
            break;
        case 45:
            lexica « yytext <<"\t"<< "ATRIB"<< "\t" << "@" « endl;
            break;
        case 46:
            cad = yytext;
            cad = cad.substr(1,cad.length()-2);
            lexica « yytext <<"\t"<< "CARACTER"<< "\t" « cad « endl;
            break;
        case 47:
            cad = yytext;
            cad = cad.substr(1,cad.length()-2);
            erros « cad << " : caracter invalido" « endl;
            break;
        case 48:
            erros << "arquivo terminado pelo comentario"«endl;
            break;

        } // fim switch

    } // fim while
    lexica << "FINAL" << "\t"<< "FINAL" <<"\t"<< "@" « endl;


}
```

Apêndice F

Objeto grafo

F.1 Diagrama de transicão de estados - analisador sintático

```
// Projeto : Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
// Fase 3: Desenvolvimento do Compilador – Analisador Sintático
// Desenvolvedor: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda

// CELULA DE VERTICE, CONDICAO, E ACAO

#ifndef GRAFO_SIN_H
#define GRAFO_SIN_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "celula_vca.h"


class grafo_sin{

public:

 grafo_sin ();
 vector <celula_vca> retornacelulas(int vert );
```

```
private:
 celula_vca temp;
 celula_vca temp1;
 celula_vca temp2;
 vector <vector <celula_vca> > vertice;
 vector <celula_vca> celulas;

};


grafo_sin :: grafo_sin (){
// corpo principal
// vertice 0 – nao faz nada, somente vai para o vertice 1
 temp.setcelula_vca(1,0,0);
 celulas.push_back(temp);//celula 1
 vertice.push_back(celulas);
 celulas.clear ();

// vertice 1 – verifica MAIN
 temp.setcelula_vca(2,1,1); // = MAIN vai para o vertice 2 e pega proximo átomo (acao 1)
 celulas.push_back(temp);//
 temp.setcelula_vca(10,2,2); // != MAIN vai para vertice 10 para encontrar um ponto e virgula ou FINAL qu
 celulas.push_back(temp);//
 vertice.push_back(celulas);
 celulas.clear ();
// vertice 2
 temp.setcelula_vca(3,3,1); // = ( vai para vertice 3 e pega proximo átomo
 celulas.push_back(temp);//celula 4
 temp.setcelula_vca(10,4,3); // != ( vai para vértice procura por ; ou FINAL acão = esperado (
 celulas.push_back(temp);//celula 5
 vertice.push_back(celulas);
  celulas.clear ();
// vertice 3
 temp.setcelula_vca(4,5,1); // = ) e próximo átomo
 celulas.push_back(temp); //
 temp.setcelula_vca(10,6,4); // != ) e erro "esperado )" vai para o vertice 10
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear ();
// vertice 4
 temp.setcelula_vca(5,7,1);   // ={ e próximo vertice
```

```
 celulas.push_back(temp);
 temp.setcelula_vca(10,8,5);   //!= {  e erro "esperado {"
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();
// vertice 5
 temp.setcelula_vca(6,0,8);  // acao 8: exec_decl_list
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();
// vertice 6
 temp.setcelula_vca(7,9,1);   // = }  próximo atomo
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();

// vertice 7
 temp.setcelula_vca(8,11,400);// encontrado FINAL acao 400: parada =1 sai do loop principal
 celulas.push_back(temp);
 temp.setcelula_vca(9,12,7); //!= FINAL esperado final
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();

// vertice 8
 temp.setcelula_vca(0,11,400);  // encontradp final  finaliza  LOOP
 celulas.push_back(temp);//15
 vertice.push_back(celulas);
 celulas.clear();

// vertice 9
 temp.setcelula_vca(9,12,1);  // procura por  final
 celulas.push_back(temp);//16
 temp.setcelula_vca(8,11,400);//se encontra vai  para o  vértice  8  onde o programa é finalizado
 celulas.push_back(temp);//17
 vertice.push_back(celulas);
 celulas.clear();

// vertice 10
 temp.setcelula_vca(5,6,1); // condição 13: =;  for  satisfeita  é porque existe instruções ou FINAL
 celulas.push_back(temp);
 temp.setcelula_vca(8,11,400);//=FINAL finaliza  programa
```

```
 celulas.push_back(temp);
 temp.setcelula_vca(10,0,1);
 celulas.push_back(temp);//19
 vertice.push_back(celulas);
 celulas.clear();
// fim do corpo principal


// inicio do corpo
// vertice 11
 temp.setcelula_vca(12,15,9); // = INT, CHAR
 celulas.push_back(temp);//20
 temp.setcelula_vca(12,16,0); // != INT, CHAR
 celulas.push_back(temp);//20
 vertice.push_back(celulas);
 celulas.clear();

// vertice 12
 temp.setcelula_vca(13,49,10);//=IF, WHILE, FOR, ....
 celulas.push_back(temp);//21
 temp.setcelula_vca(13,50,0); //!= IF, WHILE, FOR
 celulas.push_back(temp);//21
 vertice.push_back(celulas);
 celulas.clear();

// vertice 13
 temp.setcelula_vca(0,0,405);// finaliza parada_corpo
 celulas.push_back(temp);//22
 vertice.push_back(celulas);
 celulas.clear();
// fim do corpo


// inicio do <decl_list>
// vertice 14
 temp.setcelula_vca(15,15,1);
 celulas.push_back(temp);//23
 temp.setcelula_vca(17,0,0);
 celulas.push_back(temp);//24
 vertice.push_back(celulas);
 celulas.clear();
// vertice 15
```

```
 temp.setcelula_vca(16,19,1);
 celulas.push_back(temp);//25
 temp.setcelula_vca(16,20,13);
 celulas.push_back(temp);//25
 vertice.push_back(celulas);
 celulas.clear();
// vertice 16
 temp.setcelula_vca(17,13,1);
 celulas.push_back(temp);//26
 temp.setcelula_vca(18,11,14);
 celulas.push_back(temp);//27
 temp.setcelula_vca(16,14,1);
 celulas.push_back(temp);//27
 vertice.push_back(celulas);
 celulas.clear();
// vertice 17
 temp.setcelula_vca(14,15,0);
 celulas.push_back(temp);//28
 temp.setcelula_vca(18,16,0);
 celulas.push_back(temp);//28
 vertice.push_back(celulas);
 celulas.clear();
// vertice 18
 temp.setcelula_vca(0,0,402);
 celulas.push_back(temp);//29
 vertice.push_back(celulas);
 celulas.clear();
// fim do <decl_list>

// inicio da inst
// vertice 19
 temp.setcelula_vca(20,21,1); //=IF
 celulas.push_back(temp);//30
 temp.setcelula_vca(49,19,0); //=ID
 celulas.push_back(temp);//30
 /* complementar com outras instruções
 temp.setcelula_vca(20,25,18);
 celulas.push_back(temp);//30
 temp.setcelula_vca(20,27,20);
 celulas.push_back(temp);//30
 temp.setcelula_vca(20,29,21);
 celulas.push_back(temp);//30
```

```
 temp.setcelula_vca(20,32,22);
 celulas.push_back(temp);//30
 temp.setcelula_vca(20,33,16);
 celulas.push_back(temp);//30
 temp.setcelula_vca(20,35,19);
 celulas.push_back(temp);//30
 temp.setcelula_vca(6,9,0);//
 celulas.push_back(temp);//30
*/
 temp.setcelula_vca(32,22,0); // termina <inst>
 celulas.push_back(temp);//30
 vertice.push_back(celulas);
 celulas.clear();

// vertice 20
 temp.setcelula_vca(32,0,15); // acao 15 = exec_if
 celulas.push_back(temp);//32
 vertice.push_back(celulas);
 celulas.clear();

// inicio  do exec_if
// vertice 21
 temp.setcelula_vca(22,3,1);
 celulas.push_back(temp);//34
 temp.setcelula_vca(22,4,3);
 celulas.push_back(temp);//34
 vertice.push_back(celulas);
 celulas.clear();
// vertice 22
 temp.setcelula_vca(23,0,24);//acao 24: exec_expcomp deve ser uma operacao relacional vai para o vértice
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();

// vertice 23
 temp.setcelula_vca(24,5,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(24,6,4);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();
```

```
// vertice 24
 temp.setcelula_vca(25,7,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(25,8,5);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 25
 temp.setcelula_vca(26,49,10); //<inst>
 celulas.push_back(temp);//38
 temp.setcelula_vca(26,22,0);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 26
 temp.setcelula_vca(27,9,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(27,10,6);
 celulas.push_back(temp);//37
 vertice.push_back(celulas);
 celulas.clear();
// vertice 27
 temp.setcelula_vca(28,37,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(31,38,0);
 celulas.push_back(temp);//37
 vertice.push_back(celulas);
 celulas.clear();
// vertice 28
 temp.setcelula_vca(29,7,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(29,8,5);
 celulas.push_back(temp);//37
 vertice.push_back(celulas);
 celulas.clear();
// vertice 29
 temp.setcelula_vca(30,49,10);
 celulas.push_back(temp);//38
 temp.setcelula_vca(30,22,0);
 celulas.push_back(temp);//38
```

```
 vertice.push_back(celulas);
 celulas.clear();

// vertice 30
 temp.setcelula_vca(31,9,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(31,10,6);
 celulas.push_back(temp);//37
 vertice.push_back(celulas);
 celulas.clear();

// vertice 31
 temp.setcelula_vca(0,0,404);//fim do if
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 32
 temp.setcelula_vca(19,49,0);
 celulas.push_back(temp);//38
 temp.setcelula_vca(0,22,403);//fim do inst
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 33
 temp.setcelula_vca(6,9,400);//geral
 celulas.push_back(temp);//38
 temp.setcelula_vca(19,10,400);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// inicio do exec_expcomp.
// vertice 34
 temp.setcelula_vca(35,0,25); //acao 25: exec_esimp
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 35
 temp.setcelula_vca(36,39,1);
```

```
 celulas.push_back(temp);//38
 temp.setcelula_vca(37,40,30);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 36
 temp.setcelula_vca(37,0,25); // acao 25 exec_expsimp
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 37
 temp.setcelula_vca(0,0,401);//fim do expcomp
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// exec_expsimp
// vertice 38
 temp.setcelula_vca(39,0,26);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 39
 temp.setcelula_vca(38,41,1); //=OPAD
 celulas.push_back(temp);//38
 temp.setcelula_vca(40,42,0);//!=OPAD
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 40
 temp.setcelula_vca(0,0,406);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// exec_term
// vertice 41
 temp.setcelula_vca(42,0,27);
```

```
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 42
 temp.setcelula_vca(41,43,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(43,44,0);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 43
 temp.setcelula_vca(0,0,407);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// exec_fat
// vertice 44
 temp.setcelula_vca(45,19,1);//=ID
 celulas.push_back(temp);
 temp.setcelula_vca(45,45,1);//=CTE
 celulas.push_back(temp);
 temp.setcelula_vca(46,3,1);//=(
 celulas.push_back(temp);
 temp.setcelula_vca(44,47,1);
 celulas.push_back(temp);//=OPNEG
 temp.setcelula_vca(45,53,1);
 celulas.push_back(temp);//=CARACTER
 temp.setcelula_vca(48,0,28); //!= dos anteriores
 celulas.push_back(temp);
 vertice.push_back(celulas);
 celulas.clear();

// vertice 45
 temp.setcelula_vca(0,0,408);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 46
```

```
 temp.setcelula_vca(47,0,25);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 47
 temp.setcelula_vca(45,5,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(48,6,4);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 48
 temp.setcelula_vca(45,13,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(45,11,0);
 celulas.push_back(temp);//38
 temp.setcelula_vca(48,14,1);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// o vertice 49 vem da chamada à atribuição do vértice 19
// vertice 49
 temp.setcelula_vca(32,0,23); // acao 23 = exec_atrib
 celulas.push_back(temp);//32
 vertice.push_back(celulas);
 celulas.clear();

// vertice 50
// execatrib
 temp.setcelula_vca(51,19,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(54,20,13);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 51
 temp.setcelula_vca(52,51,1);
 celulas.push_back(temp);//38
```

```
 temp.setcelula_vca(54,52,29);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 52
 temp.setcelula_vca(53,0,25);  //  acao :execexp_simp
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 53
 temp.setcelula_vca(55,13,1);
 celulas.push_back(temp);//38
 temp.setcelula_vca(54,14,0);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 54
 temp.setcelula_vca(52,51,0);
 celulas.push_back(temp);//38
 temp.setcelula_vca(55,13,0);
 celulas.push_back(temp);//38
 temp.setcelula_vca(55,11,0);
 celulas.push_back(temp);//38
 temp.setcelula_vca(54,0,1);
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 55
 temp.setcelula_vca(0,0,409);  //  acao 24:execexp
 celulas.push_back(temp);//38
 vertice.push_back(celulas);
 celulas.clear();

// vertice 56


}
```

```
vector <celula_vca> grafo_sin::retornacelulas(int vert){
 return vertice[vert];
}

/*
celula_vca grafo_sin::retornacelula(int vert){
 vector <celula_vca> r_celulas;
 celula_vca r_celula;
 r_celulas = retornacelula(int vert);
 r_celula =
}*/

#endif
```

Apêndice G

Analisador sintático

G.1 Código analisador sintático

```
// Projeto : Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
// Fase 3: Desenvolvimento do Compilador – Analisador Sintático
// Desenvolvedor: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda

#ifndef ANASIN_H
#define ANASIN_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "grafo_sin.h"
#include "celula_vca.h"
#include "simbolo.h"
using namespace std;

class anasin{

public:

    anasin(string nome_arquivo);

private:
    string atomo, tipo , atributo ;
```

```
        ofstream erros;
        ifstream  entrada;
        void montagrafosin();
        int  verificacondicao ( int  codigo_condicao);
        void executa(int acao);
        int  parada;
        grafo_sin  grafo ;  // objeto  grafo
        void execcorpo (int novo_vertice, int  n_num);
        int  parada_corpo;
        void execdecl_list  ( int  novo_vertice_decl_list , int  n_num_decl_list);
        int  parada_decl_list;
        void execinst ( int  novo_vertice_inst , int  n_num_inst);
        int  parada_inst;
        void execif  ( int  novo_vertice_if , int  n_num_if);
        int  parada_if;
        void execexpsimp (int novo_vertice_expsimp, int n_num_expsimp);
        int  parada_expsimp;
        void execterm (int novo_vertice_term, int n_num_term);
        int  parada_term;
        void execfator ( int  novo_vertice_fator , int  n_num_fator);
        int  parada_fator;
        void execatrib ( int  novo_vertice_atrib , int  n_num_atrib);
        int  parada_atrib;
        void execexpcomp (int novo_vertice_expcomp, int n_num_expcomp);
        int  parada_expcomp;

        vector <simbolo> tabelasimb;
        void imprimitabelasimbolos (vector <simbolo> tabela);
        ofstream tabelasimbolos;

        int  verificaexistencia  ( string  str );
        int  verificacaodetipos( vector < string > variaveis );
        vector < string > var_exp_atrib; // vetor  que armazena os tipos de dados numa atribuiçao
        vector < string > var_exp_comp;

        string  retornotipodoid ( string  atomo);
        string  caminho;
};


anasin::anasin(string  nome_arquivo){
```

```
    string nome = nome_arquivo;
    entrada.open (nome.c_str(),ios::in );  // abre a tabela léxica para leitura
    caminho = nome.erase(nome.find_last_of("\\")+1);
    string caminho_erros = caminho + "erros.txt";
    erros.open(caminho_erros.c_str(), ios::app);      // abre o arquivo de erros para inserir os erros da ana
    entrada » atomo » tipo » atributo ; // a primeira linha da tabela léxica corresponde ao título
    entrada » atomo » tipo » atributo ; // primeiro átomo tipo e atributo

    montagrafosin(); // inicia o encaminhamento nao grafo

}
void anasin::montagrafosin(){


    // inicialização no vertice
    int vert = 1;
    int proximovertice = 1;
    int num = 0; // todas as vezes que um novo vertice é setado o num = 0;
    vector <celula_vca> vca_geral;
    celula_vca cel_vca_geral;
    int resp_geral;
    // caminhamento no vértice

    // primeira celula VCA do vertice 1 e primeira célula
    parada = 0;
    while(parada == 0){
        vca_geral = grafo.retornacelulas(proximovertice);
        cel_vca_geral = vca_geral[num];
        resp_geral = verificacondicao(cel_vca_geral.getCondicao());
        if (resp_geral == 0){// se a resposta da condição for verdadeira executa a ação e vai para o próxim
            executa(cel_vca_geral.getAcao());
            proximovertice = cel_vca_geral.getVertice ();
            num = 0;
        }
        if (resp_geral == 1)// se a resposta for falsa caminhamos para a próxima célula
            num++;

 }

}
// condicões
int anasin::verificacondicao (int codigo_condicao){
```

```
switch(codigo_condicao){
    case 0: return 0;
            break;
    case 1: if (atomo == "MAIN"|| atomo == "main") return 0;
            else return 1;
            break;
    case 2: if (atomo != "MAIN"|| atomo != "main") return 0;
            else return 1;
            break;
    case 3: if (atomo == "(") return 0;
            else return 1;
            break;
    case 4: if (atomo != "(") return 0;
            else return 1;
            break;
    case 5: if (atomo == ")") return 0;
            else return 1;
            break;
    case 6: if (atomo != ")") return 0;
            else return 1;
            break;
    case 7: if (atomo == "{") return 0;
            else return 1;
            break;
    case 8: if (atomo != "{") return 0;
            else return 1;
            break;
    case 9: if (atomo == "}") return 0;
            else return 1;
            break;
    case 10: if (atomo != "}") return 0;
            else return 1;
            break;
    case 11: if (atomo == "FINAL") return 0;
            else return 1;
            break;
    case 12: if (atomo != "FINAL") return 0;
            else return 1;
            break;
    case 13: if (atomo == ";") return 0;
            else return 1;
            break;
```

```
case 14: if (atomo != ";") return 0;
         else return 1;
         break;
case 15: if (atomo == "INT"|atomo == "CHAR"|atomo == "int"|atomo == "char") return
         else return 1;
         break;
case 16: if (atomo != "INT"|atomo != "CHAR"|atomo != "int"|atomo != "char") return
         else return 1;
         break;
case 17: if (atomo == ",") return 0;
         else return 1;
         break;
case 18: if (atomo != ",") return 0;
         else return 1;
         break;
case 19: if (tipo == "ID") return 0;
         else return 1;
         break;
case 20: if (tipo != "ID") return 0;
         else return 1;
         break;
case 21: if (atomo == "IF"|atomo == "if") return 0;
         else return 1;
         break;
case 22: if (atomo != "IF"|atomo != "if") return 0;
         else return 1;
         break;
case 23: if (atomo == "SWITCH") return 0;
         else return 1;
         break;
case 24: if (atomo != "SWITCH") return 0;
         else return 1;
         break;
case 25: if (atomo == "DO") return 0;
         else return 1;
         break;
case 26: if (atomo != "DO") return 0;
         else return 1;
         break;
case 27: if (atomo == "PRINTF") return 0;
         else return 1;
         break;
```

```
case 28: if (atomo != "PRINTF") return 0;
         else return 1;
         break;

case 29: if (atomo == "SCANF") return 0;
         else return 1;
         break;
case 30: if (atomo != "SCANF") return 0;
         else return 1;
         break;
case 31: if (atomo == "CASE") return 0;
         else return 1;
         break;
case 32: if (atomo != "CASE") return 0;
         else return 1;
         break;

case 33: if (atomo == "WHILE") return 0;
         else return 1;
         break;
case 34: if (atomo != "WHILE") return 0;
         else return 1;
         break;

case 35: if (atomo == "FOR") return 0;
         else return 1;
         break;
case 36: if (atomo != "FOR") return 0;
         else return 1;
         break;
case 37: if (atomo == "ELSE" |atomo == "else" ) return 0;
         else return 1;
         break;
case 38: if (atomo != "ELSE"|atomo != "else") return 0;
         else return 1;
         break;
case 39: if ( tipo == "OPREL") return 0;
         else return 1;
         break;
case 40: if ( tipo != "OPREL") return 0;
         else return 1;
         break;
```

```
case 41: if ( tipo == "OPAD") return 0;
         else return 1;
         break;
case 42: if ( tipo != "OPAD") return 0;
         else return 1;
         break;

case 43: if ( tipo == "OPMULT") return 0;
         else return 1;
         break;

case 44: if ( tipo != "OPMULT") return 0;
         else return 1;
         break;

case 45: if ( tipo == "CTE") return 0;
         else return 1;
         break;

case 46: if ( tipo != "CTE") return 0;
         else return 1;
         break;

case 47: if ( tipo == "OPNEG") return 0;
         else return 1;
         break;

case 48: if ( tipo != "OPNEG") return 0;
         else return 1;
         break;

case 49: if (atomo == "if"|atomo == "IF"|atomo == "for"|atomo == "FOR"|
             atomo == "while"| atomo == "WHILE"| atomo == "case"|atomo == "CASE"|
             atomo == "switch"|atomo == "SWITCH"|atomo == "case"|atomo == "sacanf
             atomo == "SCANF"|atomo == "printf"|atomo == "PRINTF"|tipo == "ID") retu
         else return 1;
         break;

case 50: if (atomo != "if") return 0;
         else return 1;
         break;
```

```
            case 51: if ( tipo == "ATRIB") return 0;
                     else return 1;
                     break;

            case 52: if ( tipo != "ATRIB") return 0;
                     else return 1;
                     break;

            case 53: if ( tipo == "CARACTER") return 0;
                     else return 1;
                     break;

            case 54: if ( tipo != "CARACTER") return 0;
                     else return 1;
                     break;



        } // fim switch
} // fim verificacondicao

// ações
void anasin::executa(int acao){
    switch (acao){
        case 0: break;

        case 1: entrada » atomo » tipo » atributo ;
                break;

        case 2: erros << "main esperado " « endl;
                break;

        case 3: erros << "esperado (" « endl;
                break;

        case 4: erros << "esperado ) " « endl;
                break;

        case 5: erros << "esperado {" « endl;
                break;

        case 6: erros << "esperado }" « endl;
```

```
            break;

case 7: erros << "esperado FINAL " « endl;
            break;

case 8: execcorpo(11, 0);
            parada = 0;
            break;

case 9: execdecl_list (14, 0); // exec_decl_list
            parada_decl_list = 0;
            break;

case 10: execinst (19, 0); // exec_inst
             parada_inst = 0;
             break; // inst

case 11: erros << "esperado ; " « endl;

case 12: //não esta sendo utilizado . é para implementar lista de variáveis
             break;

case 13: erros << "esperado identificador" « endl;
             break;

case 14: erros << "esperado ;" « endl;
             break;

case 15: execif (21, 0);
             parada_if = 0;
             break; // exec_if

case 16: break; //exec_while

case 17: break; //exec_switch

case 18: break; //exec_do

case 19: break; //exec_for

case 20: break; //exec_printf
```

```
case 21: break; //exec_scanf

case 22: break; //exec_case

case 23: execatrib(50, 0);
         parada_atrib = 0;
         break; //exec_atribuição

case 24: execexpcomp(34, 0);
         parada_expcomp = 0;
         break;

case 25: execexpsimp(38, 0);
         parada_expsimp = 0;
         break;

case 26: execterm(41, 0);
         parada_term = 0;
         break; //exec_term

case 27: execfator(44, 0);
         parada_fator = 0;
         break; //exec_fator

case 28: erros << "não esperado " « tipo « endl;
         break;

case 29: erros << "esperado atribuicao" « endl;
         break;

case 30: erros << "esperado operador relacional" « endl;
         break;

case 31: erros << "variável " « atomo << " nao declarada!!" « endl;
         break;


case 32: erros << "dados incompativeis" « endl;
         break;

case 33: erros << "variavel " « atomo <<" ja declarada!!" « endl;
         break;
```

```
            case 400: parada = 1;
                        break;

            case 401: parada_expcomp = 1;
                        break;

            case 402: parada_decl_list = 1;
                        break;

            case 403: parada_inst = 1;
                        break;

            case 404: parada_if = 1;
                        break;

            case 405: parada_corpo = 1;
                        break;

            case 406: parada_expsimp = 1;
                        break;

            case 407: parada_term = 1;
                        break;

            case 408: parada_fator = 1;
                        break;

            case 409: parada_atrib = 1;
                        break;


        } // fim  switch

} // fim  executa


void anasin::execcorpo (int novo_vertice, int  n_num){
        vector <celula_vca> vca;
        celula_vca cel_vca;
        int  resp_corpo;
        int  vert_corpo, num_corpo;
```

```
        vert_corpo = novo_vertice;
        num_corpo = n_num;
        parada_corpo = 0;
        while(parada_corpo == 0){
            vca = grafo.retornacelulas(vert_corpo);
            cel_vca = vca[num_corpo];
            resp_corpo = verificacondicao(cel_vca.getCondicao());
            if (resp_corpo == 0){
                executa(cel_vca.getAcao());
                vert_corpo = cel_vca.getVertice();
                num_corpo = 0;
            }
            if (resp_corpo == 1)
                num_corpo++;

        }


    } // fim execcorpo


    void anasin::execexpcomp (int novo_vertice_expcomp, int n_num_expcomp){
        int vet_expcomp = novo_vertice_expcomp;
        int num_expcomp = n_num_expcomp;
        int resp_expcomp;
        vector <celula_vca> vca_expcomp;
        celula_vca cel_vca_expcomp;
        parada_expcomp = 0;
        while(parada_expcomp == 0){
            vca_expcomp = grafo.retornacelulas(vet_expcomp);
            cel_vca_expcomp = vca_expcomp[num_expcomp];
            resp_expcomp = verificacondicao(cel_vca_expcomp.getCondicao());
            if (resp_expcomp == 0){
                if (cel_vca_expcomp.getCondicao()==19){
                    // é verificado se a variavel esta na tabela de simbolos
                    if ( verificaexistencia (atomo)==1)
                        executa(31); // caso a variavel nao esteja declarada é inserido um erro
                    else{
                        var_exp_comp.push_back(retornotipodoid(atomo));
                        }
                }
                // todos os dados de uma atribuição são armazenados num vetor para verificação dos tipos
```

```
            if (cel_vca_expcomp.getCondicao()== 45|cel_vca_expcomp.getCondicao()==53)//CTE ou CAR
                var_exp_comp.push_back(tipo);
            //antes da atribuição terminar é verificado se os dados são compatíveis
            if (cel_vca_expcomp.getAcao()==401){
                if (verificacaodetipos(var_exp_comp)==1)
                    executa(32);
            }

            executa(cel_vca_expcomp.getAcao());
            vet_expcomp = cel_vca_expcomp.getVertice();
            num_expcomp = 0;
        }
        if (resp_expcomp == 1)
            num_expcomp++;

    } //fim do while
    var_exp_comp.clear();
    var_exp_atrib.clear ();


} //fim execexp_comp

void anasin::execdecl_list (int novo_vertice_decl_list, int n_num_decl_list){
    int vet_decl_list = novo_vertice_decl_list;
    int num_decl_list = n_num_decl_list;
    int resp_decl_list;
    string tipo_anterior;
    simbolo c_simbolo;

    vector <celula_vca> vca_decl_list;
    celula_vca cel_vca_decl_list;
    parada_decl_list = 0;
    while(parada_decl_list == 0){
        vca_decl_list = grafo.retornacelulas( vet_decl_list );
        cel_vca_decl_list = vca_decl_list[num_decl_list];
        resp_decl_list = verificacondicao(cel_vca_decl_list.getCondicao());
        if (resp_decl_list == 0){
            if (cel_vca_decl_list.getCondicao() == 15)
                //se a condição for 15 indica que temos um int ou char
                tipo_anterior = tipo;
            if (cel_vca_decl_list.getCondicao() == 19){
                if ( verificaexistencia (atomo) == 1){// verifica se a variavel ja foi declarada
```

```
                    c_simbolo.setSimbolo(atomo, tipo_anterior);
                    tabelasimb.push_back (c_simbolo);
                }
                else executa(33);
            }

            executa(cel_vca_decl_list.getAcao());
             vet_decl_list  = cel_vca_decl_list.getVertice ();
            num_decl_list = 0;
        }
        if ( resp_decl_list == 1)
            num_decl_list++;
    }
    imprimitabelasimbolos(tabelasimb);
} // fim  exec_decl_list

void anasin::execinst ( int  novo_vertice_inst, int  n_num_inst){
    int  vet_inst  = novo_vertice_inst;
    int  num_inst = n_num_inst;
    int  resp_inst;

    vector <celula_vca> vca_inst;
    celula_vca cel_vca_inst;
    parada_inst = 0;
    while(parada_inst == 0){
        vca_inst = grafo.retornacelulas( vet_inst );
        cel_vca_inst = vca_inst[num_inst];
        resp_inst = verificacondicao(cel_vca_inst.getCondicao());
        if ( resp_inst == 0){
            executa(cel_vca_inst.getAcao());
            vet_inst  = cel_vca_inst.getVertice ();
            num_inst = 0;
        }
        if ( resp_inst == 1)
            num_inst++;

    }
} // fim  execinst

void anasin::execif ( int  novo_vertice_if, int  n_num_if){
    int  vet_if  = novo_vertice_if;
    int  num_if = n_num_if;
```

```
    int resp_if;
    vector <celula_vca> vca_if;
    celula_vca cel_vca_if;
    parada_if = 0;
    while(parada_if == 0){
        vca_if = grafo.retornacelulas( vet_if );
        cel_vca_if = vca_if[num_if];
        resp_if = verificacondicao(cel_vca_if.getCondicao());
        if ( resp_if == 0){
            executa(cel_vca_if.getAcao());
            vet_if = cel_vca_if.getVertice ();
            num_if = 0;
        }
        if ( resp_if == 1)
            num_if++;

    }
} // fim execinst


void anasin::execexpsimp (int novo_vertice_expsimp, int n_num_expsimp){
    int vet_expsimp = novo_vertice_expsimp;
    int num_expsimp = n_num_expsimp;
    int resp_expsimp;
    vector <celula_vca> vca_expsimp;
    celula_vca cel_vca_expsimp;
    parada_expsimp = 0;
    while(parada_expsimp == 0){
        vca_expsimp = grafo.retornacelulas(vet_expsimp);
        cel_vca_expsimp = vca_expsimp[num_expsimp];
        resp_expsimp = verificacondicao(cel_vca_expsimp.getCondicao());
        if ( resp_expsimp == 0){
            executa(cel_vca_expsimp.getAcao());
            vet_expsimp = cel_vca_expsimp.getVertice();
            num_expsimp = 0;
        }
        if ( resp_expsimp == 1)
            num_expsimp++;

    }

} // fim execexpsimp
```

```
void anasin::execterm (int novo_vertice_term, int n_num_term){
     int vet_term = novo_vertice_term;
     int num_term = n_num_term;
     int resp_term;
     vector <celula_vca> vca_term;
     celula_vca cel_vca_term;
     parada_term = 0;
     while(parada_term == 0){
          vca_term = grafo.retornacelulas(vet_term);
          cel_vca_term = vca_term[num_term];
          resp_term = verificacondicao(cel_vca_term.getCondicao());
          if (resp_term == 0){
               executa(cel_vca_term.getAcao());
               vet_term = cel_vca_term.getVertice();
               num_term = 0;
          }
          if (resp_term == 1)
               num_term++;

     }
} // fim execterm

void anasin::execatrib (int novo_vertice_atrib, int n_num_atrib){
     int vet_atrib = novo_vertice_atrib;
     int num_atrib = n_num_atrib;
     int resp_atrib;
     vector <celula_vca> vca_atrib;
     celula_vca cel_vca_atrib;
     parada_atrib = 0;


     while(parada_atrib == 0){
          vca_atrib = grafo.retornacelulas( vet_atrib );
          cel_vca_atrib = vca_atrib[num_atrib];
          resp_atrib = verificacondicao(cel_vca_atrib.getCondicao());
          if (resp_atrib == 0){
               if (cel_vca_atrib.getCondicao()==19){
                    if( verificaexistencia (atomo)==1)
                         executa(31);
                    else{
```

```
                    var_exp_atrib.push_back(retornotipodoid(atomo));
                    }
                }
                // todos os dados de uma atribuição são armazenados num vetor para verificação dos tipos
                if (cel_vca_atrib.getCondicao()== 45|cel_vca_atrib.getCondicao()==53)//CTE ou CARACTER
                    var_exp_atrib.push_back(tipo);
                // antes da atribuição terminar é verificado se os dados são compatíveis
                if (cel_vca_atrib.getAcao()==409){
                    if (verificacaodetipos(var_exp_atrib)==1)
                        executa(32);
                }

                executa(cel_vca_atrib.getAcao());
                vet_atrib = cel_vca_atrib.getVertice ();
                num_atrib = 0;
            }
            if (resp_atrib == 1)
                num_atrib++;

        }
        var_exp_atrib.clear (); // o vetor é esvaziado para a próxima atribuição
        var_exp_comp.clear();
    } // fim execatrib


    void anasin::execfator (int novo_vertice_fator, int n_num_fator){
        int vet_fator = novo_vertice_fator;
        int num_fator = n_num_fator;
        int resp_fator;
        vector <celula_vca> vca_fator;
        celula_vca cel_vca_fator;
        parada_fator = 0;
        while(parada_fator == 0){
            vca_fator = grafo.retornacelulas( vet_fator );
            cel_vca_fator = vca_fator[num_fator];
            resp_fator = verificacondicao(cel_vca_fator.getCondicao());
            if (resp_fator == 0){
                if (cel_vca_fator.getCondicao()==19){
                    if ( verificaexistencia (atomo)==1)
                        executa(31);
                    else {
                            var_exp_atrib.push_back(retornotipodoid(atomo));
```

```
                        var_exp_comp.push_back(retornotipodoid(atomo));
                    }
                }

                if (cel_vca_fator.getCondicao()== 45|cel_vca_fator.getCondicao()==53){
                    var_exp_atrib.push_back(tipo);
                    var_exp_comp.push_back(tipo);
                }

                executa(cel_vca_fator.getAcao());
                vet_fator = cel_vca_fator.getVertice ();
                num_fator = 0;
            }
            if (resp_fator == 1)
                num_fator++;

        }
    } // fim execfator


    // verifica se o atomo ID no caso foi declarado na tabela de símbolos
    int anasin:: verificaexistencia (string str ){
        int i = 0;
        string ver_str = str ;
        while (i < tabelasimb.size ()){
            if (ver_str == tabelasimb[i ]. getId ()){
                return 0;
            }
            else i++;
        }
        if (i >= tabelasimb.size ()) return 1;
    }
    // verifica os dados de uma atribuição ou de uma expressão
    // a função recebe todos os tipos do dados de uma expressão
    // e verifica se todos são do mesmo tipo,
    // ou seja se uma variável é inteira os valores das variáveis que
    // são atribuidas à ela de ter o mesmo tipo
    // o mesmo acontece com os caracteres.
    int anasin:: verificacaodetipos (vector < string > variaveis ){
        vector < string > t_variaveis = variaveis ;
        vector < int > tipos_var ;
        string tipo ;
```

```
    int i=0;
    while (i < t_variaveis.size()){
        if (t_variaveis[i]=="CHAR"|t_variaveis[i]=="CARACTER")
            tipos_var.push_back(1);
        if (t_variaveis[i]=="INT"|t_variaveis[i]=="CTE")
            tipos_var.push_back(0);
        i++;
    }

    i = 1;
    while( i < tipos_var.size()){
        if (tipos_var[0] != tipos_var[i]){
            return 1;
        }
        else
            i++;
    }
    if (i >= tipos_var.size()) return 0;

}

// na tabela de simbolos é armazenado o ID e o seu tipo
// quando é feita a verificaçãodo tipo é necessário verificar otipo do ID
string anasin::retornotipodoid(string atomo){
    int i=0;
    while (i < tabelasimb.size()){
        if(tabelasimb[i].getId() == atomo){
            return tabelasimb[i].getTipo();
        }
        else i++;
    }
    if (i >= tabelasimb.size()) return 0;
}

// a tabela de simbolos é impressa quando termina a execução de <decl_list>
void anasin::imprimitabelasimbolos (vector <simbolo> tabela){
    string caminhotabsimb = caminho + "tabelasimbolos.txt";
    tabelasimbolos.open(caminhotabsimb.c_str());

    simbolo simb_temp;
    for (int i = 0; i < tabela.size(); i++){
        simb_temp = tabela[i];
```

```
            tabelasimbolos « simb_temp.getId() << "\t" « simb_temp.getTipo() « endl;
        }
    }


    #endif
```

G.1.1 Analisador sintático - cel_vertice.h

```
    // Projeto : Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
    // Fase 3: Desenvolvimento do Compilador – Analisador Sintático
    // Desenvolvedor: Edna Mie Kanazawa
    // Orientador: Wilian Soares Lacerda
    // CELULA DE VERTICE, CONDICAO, E ACAO
    #ifndef CEL_VERTICE_H
    #define CEL_VERTICE_H

    #include <iostream>
    #include <fstream>
    #include <vector>
    #include <string>
    #include "celula_vca.h"

    class cel_vertice{

    public:

     cel_vertice(vector<celula_vca> cel, int c1, int c2);
     vector <celula_vca> v_celulavca;
     int v_1;
     int v_2;

    private:

    };


    cel_vca::cel_vertice(vector<celula_vca> cel, int c1, int c2);
     v_celulavca = cel;
     v_1 c1;
     v_2 c2;

    }
```

```
#endif
```

G.1.2 Analisador sintático - celula_vca.h

```
// Projeto : Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
// Fase 3: Desenvolvimento do Compilador – Analisador Sintático
// Desenvolvedor: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda
// CELULA DE VERTICE, CONDICAO, E ACAO
#ifndef CELULA_VCA_H
#define CELULA_VCA_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>



class celula_vca{

public:

 celula_vca();
 void setcelula_vca (int vertice , int condicao, int acao, int prox = 0);
 int getVertice ();
 int getCondicao();
 int getAcao();
 int getProx();
 int c_vertice , c_condicao, c_acao, c_prox;


private:

};


celula_vca::celula_vca(){


}
void celula_vca::setcelula_vca (int vertice , int condicao, int acao, int prox){
```

```
  c_vertice = vertice;
  c_condicao = condicao;
  c_acao = acao;
  c_prox = prox;

}
int celula_vca::getVertice (){
 return c_vertice;

}
int celula_vca::getCondicao(){
 return c_condicao;

}
int celula_vca::getAcao(){
 return c_acao;

}
int celula_vca::getProx(){
 return c_prox;

}

#endif
```

G.1.3 Analisador sintático - simbolo.h

```
// Projeto: Desenvovlvimento de um Sistema Integrado de Montagem, compilação e simulação
// Fase 3: Desenvolvimento do Compilador – Analisador Sintático
// Desenvolvedor: Edna Mie Kanazawa
// Orientador: Wilian Soares Lacerda

#ifndef SIMBOLO_H
#define SIMBOLO_H

#include <string>

class simbolo{

public:

    simbolo();
```

```
    void setSimbolo(string id , string tipo );
    string getId ();
    string getTipo();

private:

    string s_id;
    string s_tipo;
};

simbolo::simbolo(){
}

void simbolo::setSimbolo(string id , string tipo ){
    s_id = id;
    s_tipo = tipo;
}

string simbolo::getId (){
    return s_id;
}

string simbolo::getTipo(){
    return s_tipo;
}


#endif
```