

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

**IMPLEMENTAÇÃO DE ESTRUTURA DE ENTRADA E
SAÍDA PARA O AMBIENTE DE PROGRAMAÇÃO FURBOL**

MARCOS SILVA PIAZERA

BLUMENAU
2004

2004/2-36

MARCOS SILVA PIAZERA

**IMPLEMENTAÇÃO DE ESTRUTURA DE ENTRADA E
SAÍDA PARA O AMBIENTE DE PROGRAMAÇÃO FURBOL**

Trabalho de Conclusão de Curso submetido à
Universidade Regional de Blumenau para a
obtenção dos créditos na disciplina Trabalho
de Conclusão de Curso II do curso de Ciências
da Computação — Bacharelado.

Prof. Antônio Carlos Tavares - Orientador

**BLUMENAU
2004**

2004/2-36

IMPLEMENTAÇÃO DE ESTRUTURA DE ENTRADA E SAÍDA PARA O AMBIENTE DE PROGRAMAÇÃO FURBOL

Por

MARCOS SILVA PIAZERA

Trabalho aprovado para obtenção dos créditos
na disciplina de Trabalho de Conclusão de
Curso II, pela banca examinadora formada
por:

Presidente:

Prof. Antonio Carlos Tavares – Orientador, FURB

Membro:

Joyce Martins, FURB

Membro:

Prof. Mauro Mattos, FURB

Blumenau, 24 de novembro de 2004.

Dedico este trabalho a meus Pais e a minha
companheira Duda que me apoiaram durante
todo o período do curso na Universidade.

AGRADECIMENTOS

Especial agradecimento aos meus pais que em todo momento acreditaram nas minhas capacidades e sempre me ajudaram e ainda ajudam.

À minha noiva que esteve ao meu lado nos momentos mais difíceis.

Ao meu orientador que sempre demonstrou prontidão em auxiliar-me e esclarecer dúvidas.

E a todos que diretamente ou indiretamente contribuíram para que a conclusão deste projeto fosse concretizada.

RESUMO

Este trabalho descreve a implementação de um sistema de entrada e saída de dados baseado na arquitetura em camadas descrita por Tanenbaum (1992). O sistema implementado será utilizado em uma versão do ambiente de programação FURBOL originada por alguns Trabalhos de Conclusão de Curso na Universidade Regional de Blumenau. O FURBOL é brevemente apresentado e também são discutidas algumas áreas com as quais o trabalho tem relação como linguagens de programação e sistemas operacionais. Em especial é dada ênfase no gerenciamento de entrada e saída de dados. O trabalho apresenta a arquitetura do sistema de entrada e saída e descreve a interface entre ele e o FURBOL.

Palavras chaves: FURBOL, Sistema E/S, Sistemas Operacionais, Linguagens de Programação.

ABSTRACT

This work describes the implementation of an Input/Output (I/O) system based on the layered architecture described by Tanenbaum (1992). The system developed will be used in a version of the FURBOL programming environment originated by some course conclusion jobs at Universidade Regional de Blumenau. The FURBOL is briefly presented and some areas which are related to this work are discussed too, like programming languages and operating systems. It is emphasized the data input output management. The work presents the input/output system architecture and the interface with FURBOL.

Key-Words: FURBOL, Input/Output System, Operating System, Programming Languages.

LISTA DE ILUSTRAÇÕES

Figura 1 – Editor de programas no FURBOL	15
Figura 2 – Visualização código <i>assembly</i>	16
Figura 3 – Processamento de requisições de comandos de E/S.	18
Figura 4 – Camadas do Software de E/S. Adaptado de Tanenbaum (1992)	19
Figura 5 – Organização interna do processador 8088	22
Figura 6 – Registradores do 8088. Adaptado de Tischer (1990).....	23
Figura 7 – Interação entre usuário, FURBOL, programa do usuário e Sistema de E/S.	26
Figura 8 – Fluxograma de comportamento do sistema de E/S.....	27
Figura 9 – Fluxograma de comportamento da função ‘crie’.	28
Figura 10 – Fluxograma de comportamento da função ‘abra’.	29
Figura 11 – Fluxograma de comportamento da função ‘feche’.....	30
Figura 12 – Fluxograma de comportamento da função ‘grave’.	31
Figura 13 – Fluxograma de comportamento da função ‘leia’.	32
Quadro 1 – Exemplo de código gerado pelo FURBOL para um comando de E/S	33
Quadro 2 – Código gerado pela ação semântica da função ‘crie’.	35
Quadro 3 – Código gerado pela ação semântica da função ‘abra’.	36
Quadro 4 – Código gerado pela ação semântica da função ‘feche’.....	36
Quadro 5 – Código gerado pela ação semântica da função ‘grave’.....	37
Quadro 6 – Código gerado pela ação semântica da função ‘leia’.....	37
Quadro 7 – Definição dos Comandos da Linguagem.....	38
Quadro 8 – Definição dos Comandos de Atribuição.....	39
Quadro 9 – Definição dos Comandos de Entrada e Saída.....	40
Quadro 10 – Definição dos Comandos de Entrada e Saída (Continuação).....	41
Figura 14 - Camadas do Sistema de E/S.	42
Tabela 2 - Arquivos Abertos.	43
Figura 15 – Código para criar um arquivo no FURBOL.....	44
Figura 16 – Trecho de código <i>assembly</i> gerado pelo FURBOL para a função <i>crie</i>	45
Figura 17 - Inicialização de Sistema de E/S e execução de programa gerado no FURBOL....	45
Figura 18 - Resultado da execução do programa.	46
Figura 19 - Código para abrir um arquivo no FURBOL.	46
Figura 20 - trecho do código <i>assembly</i> gerado para a função ‘abra’.....	47
Figura 21 - Código para gravar dados em um arquivo no FURBOL.	48
Figura 22 - Trecho do código <i>assembly</i> gerado para a função ‘grave’.....	48

LISTA DE TABELAS

Tabela 1 – Grupo de comandos de E/S.....	34
Tabela 2 - Arquivos Abertos.	43

LISTA DE SIGLAS

BIOS – *Basic Input Output System*

CPU – *Central Processing Unit*

DOS – *Disk Operating System*

IOCS – *Input Output Control System*

LUT – *Logic Unit System*

PC – *Personal Computer*

PCB- *Process Control Block*

RF – *Requisito Funcional*

RNF – *Requisito não Funcional*

LISTA DE SÍMBOLOS

@ - arroba

SUMÁRIO

1 INTRODUÇÃO.....	11
1.1 OBJETIVOS DO TRABALHO	11
1.2 ESTRUTURA DO TRABALHO	12
2 FUNDAMENTAÇÃO TEÓRICA	13
2.1 AMBIENTES DE PROGRAMAÇÃO.....	13
2.2 COMPILADORES	13
2.3 AMBIENTE DE PROGRAMAÇÃO FURBOL	14
3 ENTRADA E SAÍDA DE DADOS	17
3.1 HARDWARE DE E/S.....	17
3.2 SOFTWARE DE E/S	18
3.2.1 SOFTWARE AO NÍVEL DE USUÁRIO	20
3.2.2 SOFTWARE INDEPENDENTE DE DISPOSITIVO	20
3.2.3 DRIVERS DE DISPOSITIVO.....	20
3.2.4 TRATADORAS DE INTERRUPÇÃO.....	21
3.3 ARQUITETURA IBM-PC 8088.....	21
3.3.1 ORGANIZAÇÃO INTERNA DO MICROPROCESSADOR 8088	22
3.3.2 REGISTRADORES	23
4 DESENVOLVIMENTO DO SISTEMA DE E/S	25
4.1 ESPECIFICAÇÃO DO SISTEMA	25
4.1.1 O GERENCIADOR DE E/S.....	25
4.1.2 COMANDOS DE E/S.....	32
4.2 IMPLEMENTAÇÃO	41
4.2.1 SISTEMA DE E/S	42
4.2.2 GERENCIADOR DE E/S.....	42
4.3 ALTERAÇÕES NO AMBIENTE FURBOL.....	43
4.4 RESULTADOS	44
5 CONCLUSÕES.....	49
5.1 EXTENSÕES	50
REFERÊNCIAS BIBLIOGRÁFICAS	51
APÊNDICE A –Código Fonte do Gerenciador de E/S	52
APÊNDICE B –Código Fonte da Função ‘crie’	53
APÊNDICE C –Código Fonte da Função ‘abra’	54

APÊNDICE D –Código Fonte da Função ‘feche’	55
APÊNDICE E –Código Fonte da Função ‘grave’	56
APÊNDICE F –Código Fonte da Função ‘leia’	57

1 INTRODUÇÃO

Há algum tempo vem sendo desenvolvido no Departamento de Sistemas e Computação da Universidade Regional de Blumenau (FURB), sob responsabilidade do professor José Roque Voltolini da Silva, o projeto FURBOL (ANDRE, 2000; ADRIANO, 2001; BIEGING, 2002; SILVA, 2002). Este projeto tem por objetivo construir um ambiente de programação com linguagem de programação própria totalmente em português. Ainda em fase de desenvolvimento, o FURBOL já implementa várias funcionalidades, entre estas, algumas de alto grau de complexidade como controle de processos concorrentes. Porém, ainda necessita de algumas funções básicas das quais um ambiente de programação não pode deixar de disponibilizar aos desenvolvedores, como as rotinas de entrada e saída.

O FURBOL necessita de rotinas de entrada e saída de dados para permitir a interação entre os programas e recursos externos.

Paralelamente, coordenado pelo professor Mauro Marcelo Mattos, vem sendo construído um simulador de uma CPU real chamado VXT (MATTOS; TAVARES; OLIVEIRA, 1997), que tem por objetivo construir uma ferramenta para utilização no ensino de arquitetura de computadores e sistemas operacionais, que permita a execução passo a passo de programas em linguagem de máquina, com detalhes de acesso a periféricos, interrupções entre outras características.

O domínio da geração de código através do FURBOL permitirá o desenvolvimento integrado de atividades utilizando o VXT para o estudo de conceitos de sistemas operacionais, arquitetura de computadores e linguagens de programação.

Este trabalho visa dar continuidade à implementação de recursos e funcionalidades para o ambiente de programação FURBOL e disponibilizar os códigos gerados neste ambiente para execução no VXT. Serão implementadas as rotinas de entrada e saída de dados.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é implementar a estrutura de entrada e saída para a linguagem FURBOL.

Os objetivos específicos do trabalho são:

- a) gerar as rotinas para tratar dados durante operações de leitura e escrita;
- b) gerar rotinas para acesso às estruturas de arquivos;
- c) gerar as rotinas para as chamadas das funções do núcleo residente.

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo contém uma introdução do que será apresentado neste trabalho bem com a organização em que será apresentado.

No capítulo dois é apresentada a fundamentação teórica, onde se discute algumas áreas e tecnologias relacionadas à implementação do mesmo. O terceiro capítulo aborda o tema de entrada e saída de dados. Na seqüência, o capítulo quatro contém a especificação demonstrando todo o sistema e sua organização. Em seguida, ainda no mesmo capítulo, são descritos os detalhes da implementação do sistema. Por fim, são apresentadas as conclusões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta brevemente as áreas correlatas ao trabalho, com o objetivo de fazer com que o leitor tenha uma base para prosseguir na leitura dos próximos capítulos que tratam do desenvolvimento do sistema.

2.1 AMBIENTES DE PROGRAMAÇÃO

Segundo Sebesta (1999), um ambiente de programação é uma coleção de ferramentas utilizadas no desenvolvimento de software. Esse ambiente pode compor-se de apenas um sistema de arquivos, editor de texto, *linker* e um compilador, ou pode compor-se de uma enorme coleção de ferramentas integradas voltadas para o desenvolvimento de software e acessadas por uma interface uniforme.

As várias categorias de ambientes de programação podem ter diversos objetivos e origens diferentes, mas o objetivo principal de seu surgimento é, segundo o mesmo autor, uniformizar o acesso às ferramentas e facilitar a edição e geração de programas fazendo com que o programador volte o foco de sua atenção para a codificação do programa em si, abstraindo-se de vários aspectos que não são de muita relevância para quem está preocupado em solucionar o problema.

Os ambientes de programação devem implementar algumas funções básicas chamadas de funções do núcleo, para a interação entre usuário (desenvolvedor) e o sistema operacional da máquina segundo Sebesta (1999). Alguns exemplos dessas funções são: criação e gravação de arquivos, alocação de memória e impressão (terminal de vídeo e impressora).

Um ambiente de programação deve disponibilizar em sua coleção de ferramentas uma que será responsável pela inclusão de rotinas externas ao programa no código de máquina. Esta ferramenta é chamada de ligador.

2.2 COMPILADORES

Para Aho, Sethi e Ullmann (1988, p. 01), um compilador é um programa que lê outro programa escrito em uma linguagem e traduz o mesmo para um programa equivalente em outra linguagem.

Na maioria dos casos, os compiladores têm como objetivo principal a geração de código de máquina a partir de uma linguagem de nível mais alto, como o FURBOL, por exemplo, para facilitar ao programador a representação dos problemas do mundo real.

Para fazer esta geração de código de máquina o compilador normalmente passa por três etapas que são: análise léxica, análise sintática e análise semântica.

Na análise léxica, o compilador faz uma varredura no código fonte do programa escrito na linguagem de mais alto nível validando todos os *tokens* do programa. O resultado desta etapa é uma tabela de símbolos.

Esta tabela de símbolos que resulta da análise léxica é a entrada para a próxima etapa que é a análise sintática. Nesta etapa é avaliada a utilização correta dos comandos da linguagem, o formato com que o programa foi construído. O compilador agora faz novamente uma varredura nos *tokens* do programa e, baseando-se nas regras de especificação da linguagem, verifica se a seqüência em que os símbolos foram utilizados para escrever o programa está correta.

Finalmente na análise semântica, o compilador irá fazer validações de tipos de dados, por exemplo. Nesta etapa é verificado se as operações feitas sobre identificadores são válidas para os tipos os quais foram definidos para estes (inteiro, *string*, *array*, etc.). Durante a análise semântica, ações semânticas podem gerar o código objeto.

É importante salientar que a análise semântica poderá ser feita ao mesmo tempo em que a análise sintática. Não é preciso separar as duas.

2.3 AMBIENTE DE PROGRAMAÇÃO FURBOL

O FURBOL foi desenvolvido utilizando-se a linguagem DELPHI. O núcleo do FURBOL é composto por uma série de rotinas em linguagem *Assembly*.

A Figura 1 apresenta a tela principal do FURBOL. O FURBOL incorpora as funcionalidades de abrir, criar e compilar um programa.

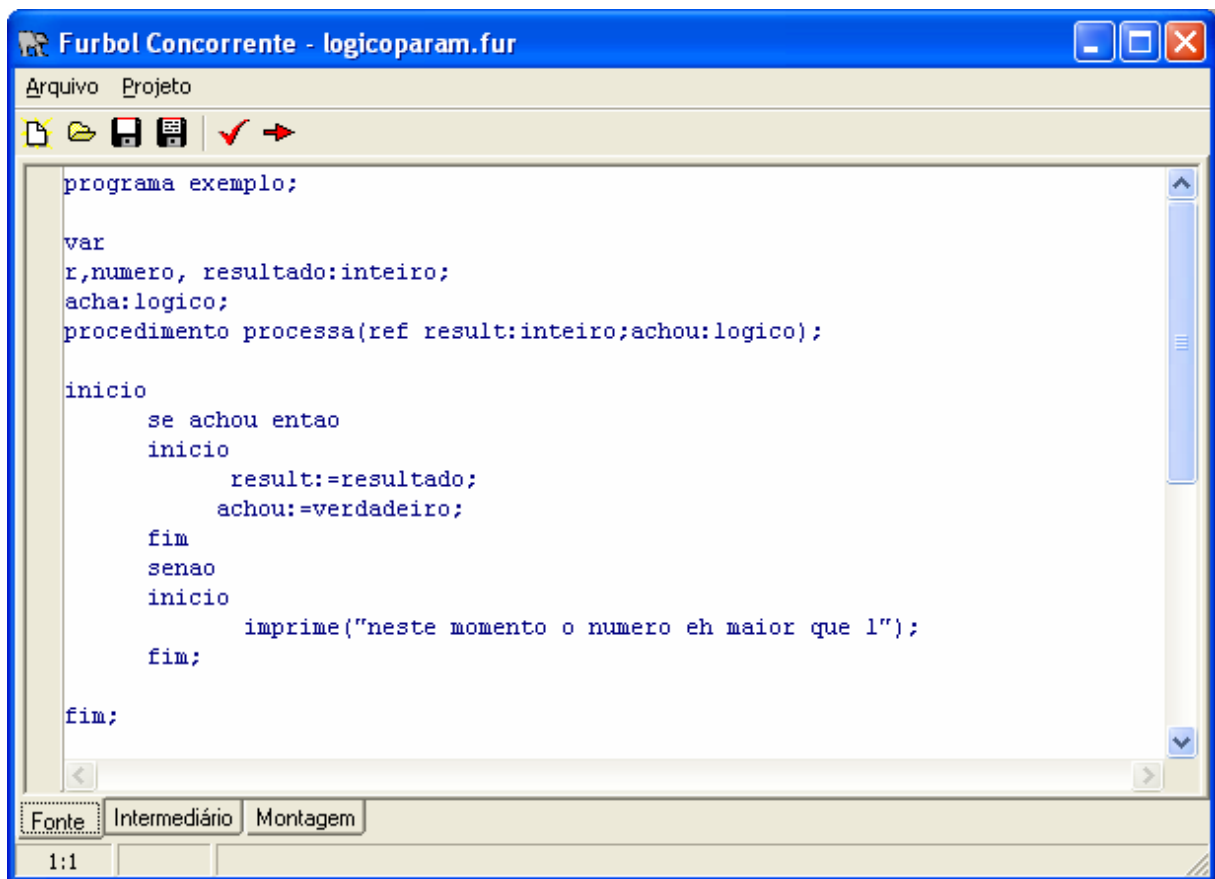
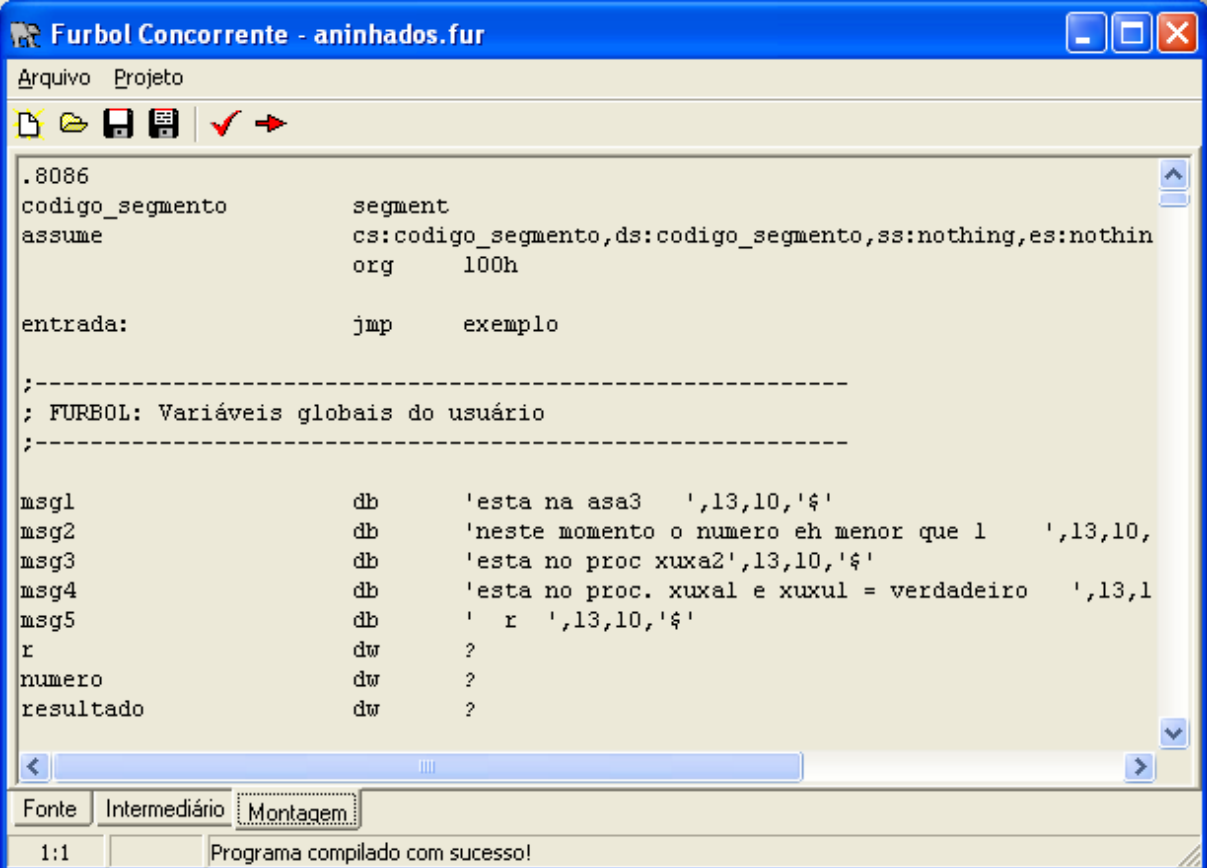


Figura 1 – Editor de programas no FURBOL

Para converter o programa escrito na linguagem FURBOL é necessário que este passe pelo compilador. O compilador analisa todo o código fonte do programa e converte os comandos do programa da linguagem FURBOL para a linguagem *Assembly*. O FURBOL adiciona ainda ao programa, todas as rotinas escritas em linguagem *Assembly* que compõem o seu núcleo. Até então, o programa não pode ser executado. Para isso o FURBOL utiliza um montador que converte o programa da linguagem *Assembly* para um programa objeto.

O Ambiente mostra ainda o código *Assembly* (Figura 2) gerado na compilação do programa.



```

Furbol Concorrente - aninhados.fur
Arquivo Projeto
[Icons: New, Open, Save, Print, Run, Stop]

.8086
codigo_segmento      segment
assume               cs:codigo_segmento,ds:codigo_segmento,ss:nothing,es:nothin
org                  100h

entrada:             jmp      exemplo

;-----
; FURBOL: Variáveis globais do usuário
;-----

msg1                 db      'esta na asa3  ',13,10,'$'
msg2                 db      'neste momento o numero eh menor que 1  ',13,10,
msg3                 db      'esta no proc xuxa2',13,10,'$'
msg4                 db      'esta no proc. xuxal e xuxul = verdadeiro  ',13,1
msg5                 db      ' r ',13,10,'$'
r                    dw      ?
numero               dw      ?
resultado            dw      ?

Fonte  Intermediário  Montagem
1:1   Programa compilado com sucesso!

```

Figura 2 – Visualização código *assembly*

Maiores esclarecimentos sobre o funcionamento do ambiente FURBOL e da especificação e estruturas de sua linguagem de programação podem ser encontrados em Biegging (2002) e Silva (2002) que são trabalhos desenvolvidos na Universidade e dão continuidade ao FURBOL. A versão do FURBOL gerada por SILVA (2002) implementa diversas funcionalidades, entre elas o controle de processos concorrentes. É possível também implementar sub-rotinas com a linguagem FURBOL. No entanto, as funcionalidades já existentes, ainda não implementam as rotinas de entrada e saída de dados.

3 ENTRADA E SAÍDA DE DADOS

Para Silberschatz, Peterson e Galvin (1991), dificilmente encontra-se um programa de computador que não execute instruções de entrada e saída de dados, isso inclui criação e manipulação de arquivos, navegação entre diretórios, leitura de dados do teclado, etc. Estas instruções de entrada e saída de dados pelo programa são feitas através de chamadas de funções residentes no sistema operacional da máquina. Existem várias funções residentes no sistema operacional, entre elas existe um grupo que é o de rotinas de entrada e saída de dados. Quando um programa necessita abrir um arquivo, por exemplo, é executada uma instrução que chama uma função residente do sistema para executar a abertura do arquivo. Neste momento o sistema operacional assume o controle da máquina, salva o contexto do programa, que é o conteúdo dos registradores e o contador de instruções, e então executa a função. Após o término da execução da função, o sistema operacional recupera o conteúdo dos registradores anterior à chamada da função, ou seja, o contexto do programa, e retorna o controle da máquina para o mesmo, que então continua de onde havia parado.

O sistema de entrada e saída de dados de um computador é, se não a principal, mas uma de suas principais funcionalidades. É através dele que o usuário pode interagir, enviando comandos e recebendo respostas do computador. Para Silberschatz, Peterson e Galvin (1991), um dos propósitos do sistema de entrada e saída de dados é esconder as peculiaridades dos dispositivos de hardware específicos, do usuário, facilitando, desta forma, a interação do usuário com o computador. Tanenbaum (1992) apresenta uma arquitetura de entrada e saída dividida em camadas, que será discutida melhor na seção que trata de software de entrada e saída.

Este capítulo descreve brevemente o sistema de entrada e saída de dados, referenciado deste ponto em diante por sistema de E/S, bem como as partes que o compõem. Muito provavelmente alguns aspectos do sistema de E/S descritos neste trabalho aplicam-se somente à arquitetura IBM PC 8088, pois, é baseado nesta arquitetura que o trabalho foi desenvolvido.

3.1 HARDWARE DE E/S

Na arquitetura IBM PC 8088, o hardware de E/S é dividido em duas partes: os dispositivos e suas controladoras. O dispositivo é quem efetivamente executa os comandos de E/S. O disco rígido, a impressora, o terminal de vídeo, são exemplos de dispositivos de E/S.

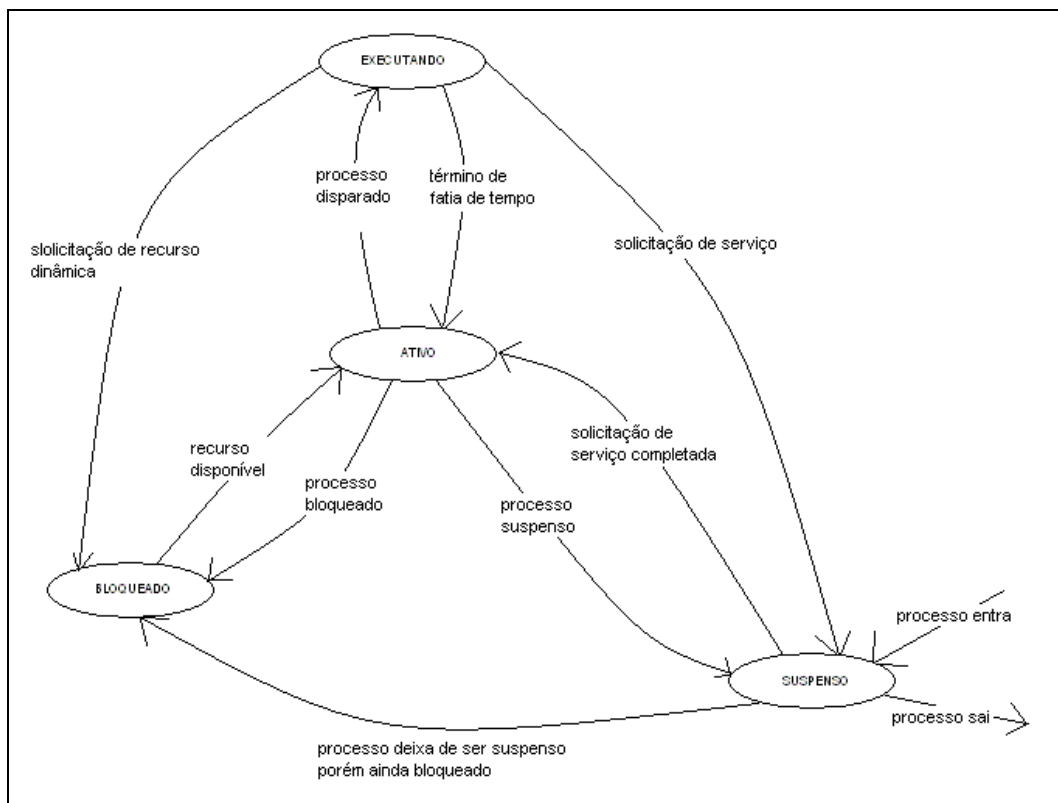
Cada dispositivo trabalha de uma maneira diferente, é acessado através de um endereço diferente, utiliza registradores diferentes, por isso existe uma controladora para cada dispositivo, ou tipo de dispositivo. A controladora do dispositivo fica responsável por receber os comandos de E/S e devolver o retorno de sua execução.

Existem várias técnicas que tratam os problemas existentes na execução dos comandos de E/S ao nível de hardware, porém, não serão abordados neste trabalho, pois não é o foco principal.

3.2 SOFTWARE DE E/S

O software de E/S é quem gerencia a utilização dos dispositivos de hardware do computador. Cada dispositivo contém particularidades que devem ser tratadas nesta parte do sistema operacional. Existem algumas propostas para se implementar o software de E/S. Para isso é preciso se conhecer todo o processo necessário para a execução de comandos de E/S.

Pinkert e Wear (1989) apresentam um diagrama dos passos do processamento de requisições de comandos de E/S (Figura 3).



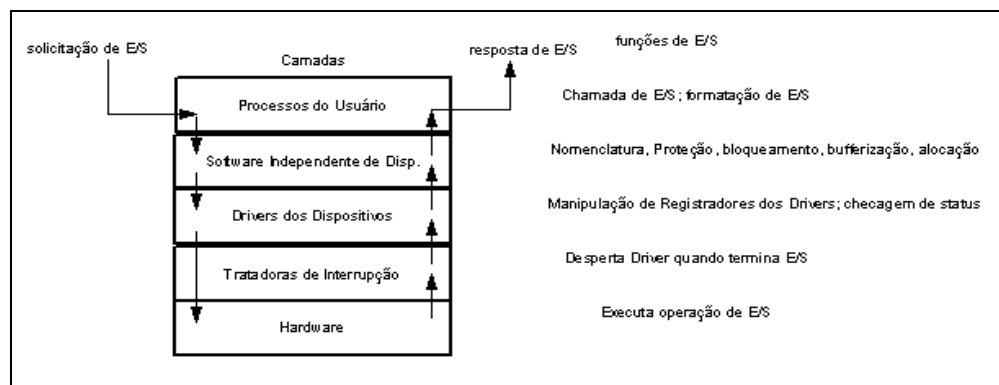
Fonte: Adaptado de Pinkert e Wear (1989).

Figura 3 – Processamento de requisições de comandos de E/S.

Pinkert e Wear (1989), na mesma obra, definem o que eles chamam de *Input Output Control System* (IOCS), ou seja, Sistema de Controle de Entrada e Saída. Os autores afirmam que o IOCS é quem deve verificar a disponibilidade dos dispositivos antes de iniciar as operações e verificar o *status* dos dispositivos após o término das mesmas. Os autores sugerem também a utilização de uma tabela lógica dos dispositivos a qual chama de *Logic Unit Table* (LUT). Outro aspecto interessante abordado na obra é o de que nem sempre os processos que solicitam operações de E/S precisam ser bloqueados pelo sistema operacional para aguardar o término da execução da operação, sugerindo a implementação de um *loop* dentro do próprio processo, que teria, então, a responsabilidade de saber quando a operação foi completada.

Dentre outras propostas de estruturação para o software de E/S existentes, a de Tanenbaum (1992) foi escolhida como base para a implementação deste trabalho.

Com seus objetivos pré-definidos, Tanenbaum (1992) divide a estruturação do software de E/S em quatro camadas (Figura 4), quais sejam: software ao nível de usuário, software independente de dispositivo, *drivers* de dispositivo, tratadoras de interrupção.



Fonte: Adaptado de Tanenbaum (1992).

Figura 4 – Camadas do Software de E/S. Adaptado de Tanenbaum (1992)

Estas quatro camadas dividem o sistema de E/S por grau de abstração e funcionalidades, visando facilitar o entendimento, implementação e manutenção do sistema de E/S. A Figura 6 apresenta como cada camada está organizada e como é a interface entre elas.

3.2.1 SOFTWARE AO NÍVEL DE USUÁRIO

Esta é a camada de mais alto nível de todas, onde ocorre a interação com o usuário. É através dela que o usuário envia comandos de E/S e recebe o retorno da execução do comando. Na verdade, esta camada é o próprio programa do usuário, ela não faz parte do sistema de E/S.

A função desta camada é disponibilizar comandos para que o compilador reconheça-os como sendo comandos de E/S e os converta para chamadas diretas à próxima camada que é a primeira efetivamente do sistema de E/S e onde tudo é centralizado.

3.2.2 SOFTWARE INDEPENDENTE DE DISPOSITIVO

Para Tanenbaum (1992), a fronteira entre os *drivers*, que é a camada inferior, e o software independente de dispositivo irá depender do sistema, pois algumas funcionalidades que poderiam ser feitas independentes do dispositivo podem ser feitas no *driver* por eficiência ou outras razões.

O software independente de dispositivo normalmente deve preocupar-se em executar funções comuns para todos os dispositivos, prover uma interface uniforme para a camada superior (programa do usuário), mapear nomes de dispositivos para os respectivos *drivers*, armazenar dados, controlar acesso aos dispositivos e tratar eventuais erros que a camada inferior não foi capaz de fazê-lo.

O controle do acesso aos dispositivos diz respeito ao gerenciamento do mesmo quanto a alocar ou liberar um recurso. Esta camada deve, antes de enviar comandos ao *driver*, verificar se o dispositivo solicitado está sendo utilizado ou não e também deve liberar o dispositivo após o seu uso para que outros processos possam utilizá-lo.

3.2.3 DRIVERS DE DISPOSITIVO

O *driver* do dispositivo é quem irá solicitar a execução de comandos de E/S diretamente ao hardware de E/S. Ele irá receber comandos da camada de software independente de dispositivo, que já verificou a consistência dos mesmos, e deve convertê-los em comandos que representem ações para o dispositivo.

O *driver* envia comandos e parâmetros para a controladora do dispositivo através de seus registradores. Somente nesta camada é que se deve saber realmente como se comunicar com os dispositivos de entrada e saída, até então isto era transparente. Ao solicitar E/S, o *driver* bloqueia-se esperando o término da execução do comando.

O término da E/S é indicado por uma mensagem vinda da tratadora de interrupção (seção seguinte), que “desperta” o *driver* de seu estado de bloqueio. Este por sua vez irá fazer verificações para ver se o comando foi realmente executado, ou não, fazer algum eventual tratamento e passar o resultado da operação para a camada superior.

O gerenciamento do dispositivo de E/S pode ser feito no *driver* ou mesmo na camada de software independente de dispositivo. A escolha pode variar dependendo da prioridade de fatores como performance, por exemplo.

Pinkert e Wear (1989) dividem o *driver* em duas partes principais: *iniciator* e *continuator*. O *iniciator* é responsável por iniciar a operação de E/S e o *continuator* é quem deve processar as interrupções que sinalizam o término da E/S e efetuar a transferência dos dados.

3.2.4 TRATADORAS DE INTERRUPÇÃO

As tratadoras de interrupção são rotinas residentes na memória principal do computador. Estas rotinas são responsáveis por informar ao *driver* que solicitou o comando de E/S que o mesmo já foi executado. O fim da execução do comando de E/S é sinalizado pelo próprio dispositivo através de uma interrupção de hardware.

Cada dispositivo deve ter uma rotina tratadora de interrupção. No vetor de interrupções são colocados os endereços destas rotinas, de forma que quando um dispositivo deseja informar que terminou a execução do comando de E/S, ele gera uma interrupção de hardware, informando a posição no vetor de interrupções que contém o endereço para a rotina que deve tratar esta interrupção. Assim, o controle é passado para esta rotina que irá tratar de mandar uma mensagem para o *driver* que solicitou E/S para o dispositivo e terminar.

3.3 ARQUITETURA IBM-PC 8088

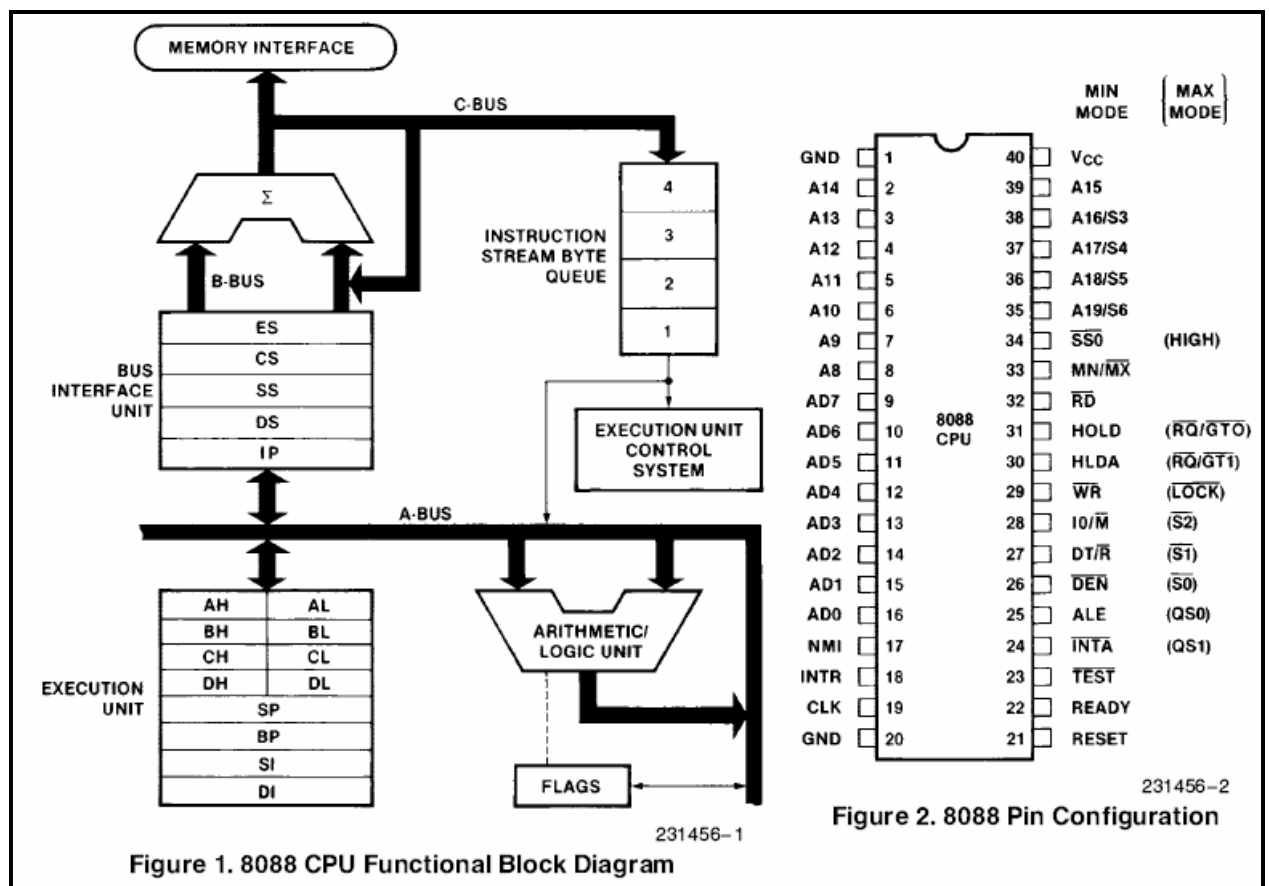
Nesta seção é descrita brevemente a arquitetura IBM-PC 8088, que é a arquitetura para qual o trabalho foi desenvolvido. O objetivo desta seção não é de esgotar as possibilidades em

torno do assunto e sim apresentar esta arquitetura ao leitor. Para obter informações detalhadas acerca do assunto o leitor pode recorrer a uma enorme coleção de obras já publicadas. A obra utilizada pelo autor como referência foi a de Tischer (1990).

O microcomputador IBM PC 8088, segundo Tischer (1990), é o cérebro do computador pessoal (PC). Ele pode entender uma série de instruções em linguagem de máquina e executá-las. O grau de abstração de um programa escrito em linguagem de máquina é muito menor do que um escrito em linguagem de alto nível como, por exemplo, Pascal.

3.3.1 ORGANIZAÇÃO INTERNA DO MICROPROCESSADOR 8088

A Figura 5 ilustra a organização interna do processador 8088. Maiores informações podem ser encontradas em Tischer (1990).

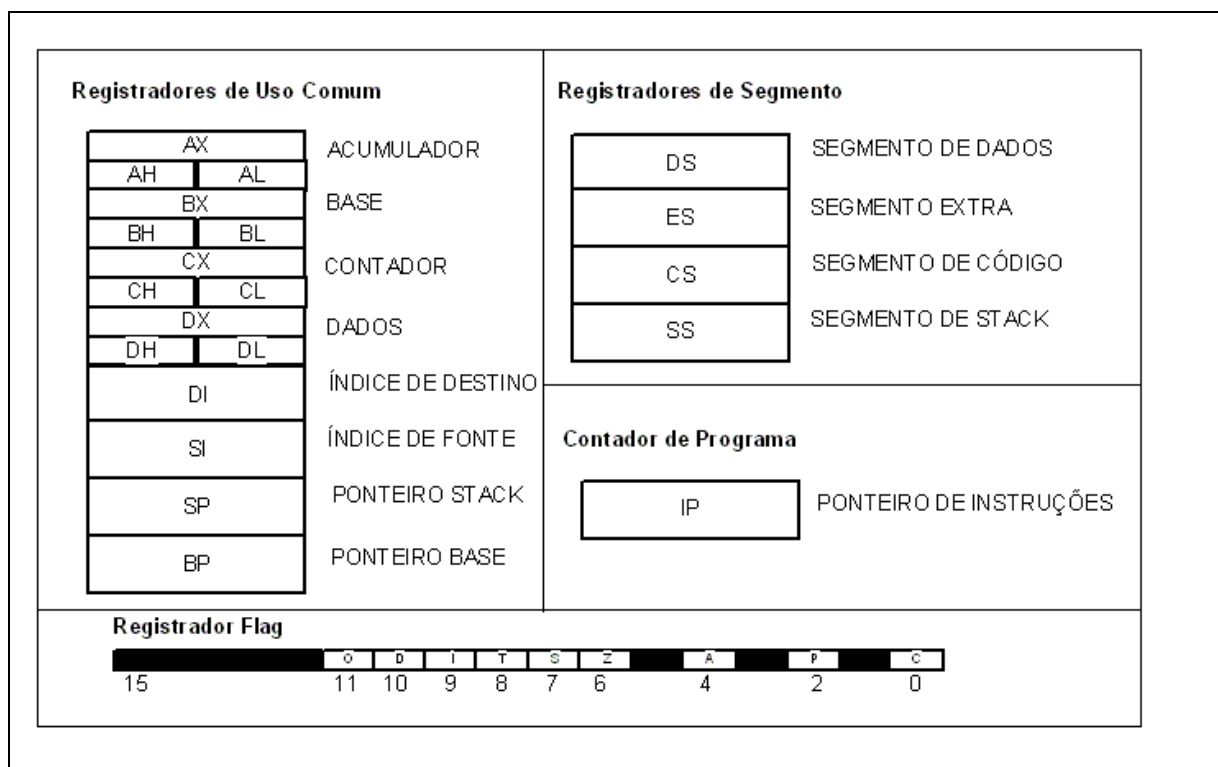


Fonte: IBM, United States (2004)

Figura 5 – Organização interna do processador 8088

3.3.2 REGISTRADORES

O microprocessador 8088 contém vários registradores (Figura 6). Os registradores são posições de memória dentro do próprio microprocessador que são utilizados para armazenar dados e endereços durante a execução das instruções. Eles podem ser de uso geral ou específico.



Fonte: Adaptado de Tischer (1990).

Figura 6 – Registradores do 8088.

Os registradores, conforme mostrado na Figura 08, estão divididos em quatro grupos: registradores de uso comum, registradores de segmento, contador de programa e *flags*.

Os registradores de uso comum são os seguintes: AX (acumulador), BX (base), CX (contador), DX (dados), de 16 bits, que podem ser divididos em 2 registradores de 8 bytes sendo AH e AL por exemplo, onde AH são os 8 bits mais significativos de AX e AL os 8 menos significativos. Também compõem o grupo dos registradores de uso comum o DI (índice de destino), o SI (índice de origem), o SP (ponteiro da memória *stack*) e o BP (base da memória *stack*). Os registradores de uso comum são utilizados geralmente em operações

aritméticas, manipulação de caracteres, chamadas de interrupções (parâmetros de entrada e/ou saída) e gerenciamento da memória *stack*. O DS (segmento de dados), o ES (segmento extra), o CS (segmento de código) e o SS (segmento de *stack*) formam o grupo dos registradores de segmento. O microprocessador 8088 utiliza estes registradores como base no endereçamento de memória. O endereço de memória utilizado pelo microprocessador quando executa uma instrução é formado por um dos registradores de segmento e mais um endereço que é informado na própria instrução. O endereço informado na instrução indica o deslocamento em relação ao início do segmento especificado pelo registrador DS até a posição de memória desejada para a operação.

O IP (contador de instruções) forma sozinho o grupo do contador de programa. Ele contém o endereço da próxima instrução a ser executada pelo microprocessador 8088. A posição de memória para a qual o IP aponta deve ser formada por uma instrução válida para o microprocessador, ou seja, algo que ele consiga interpretar como uma ação a ser tomada.

Finalmente o registrador *flags* faz parte do último grupo. O registrador *flags* é alterado quando da execução de uma instrução. Para saber se alguma anormalidade aconteceu durante a execução da instrução, ou se alguma situação especial ocorreu, existem instruções especiais para testar os bits deste registrador. Estas instruções podem ser encontradas em Tischer (1990).

4 DESENVOLVIMENTO DO SISTEMA DE E/S

O sistema de E/S desenvolvido neste trabalho tem como objetivo disponibilizar rotinas de entrada e saída de dados, em especial de manipulação de arquivos, para os usuários do ambiente de programação FURBOL.

4.1 ESPECIFICAÇÃO DO SISTEMA

Para poder contemplar os objetivos deste trabalho, foi desenvolvido um sistema, atendendo aos requisitos levantados na proposta quais sejam:

- a) as rotinas devem atender às funções básica de entrada e saída de dados na tela do computador (requisito funcional);
- b) as rotinas devem atender às funções básicas de armazenamento em disco (requisito funcional);
- c) seguir o padrão de nomenclatura de comandos, tipos de dados etc, utilizados no FURBOL, ou seja, o português (requisito não funcional);
- d) as rotinas devem ser implementadas de maneira que sejam facilmente interpretadas, visando tornar o seu entendimento de forma intuitiva pelos usuários do ambiente (requisito não funcional).

Um sistema de E/S precisa de comandos que permita manipular arquivos, enviar dados à impressora, capturar dados do teclado e enviar dados para o terminal de vídeo. Os comandos de manipulação de arquivos devem permitir a criação, abertura, gravação, leitura e fechamento dos mesmos.

Nas próximas seções é apresentada a especificação de todo o sistema de E/S bem como das partes que o compõem e também as alterações na especificação da linguagem FURBOL.

4.1.1 O GERENCIADOR DE E/S

O usuário do FURBOL escreve um programa que contém comandos de E/S e em seguida compila-o. O FURBOL então, converte os comandos de E/S para instruções na linguagem *assembly* que representam chamadas ao sistema de E/S. O sistema de E/S é quem executa as funções e retorna um código para o programa escrito no FURBOL, o código pode ser diferente dependendo da função desejada. A ilustração desta interação pode ser visualizada na Figura 7. O diagrama da figura foi criado utilizando-se o software *Enterprise*

Arquitetura de gerenciamento de projetos da empresa Sparx Systems. Pode-se ver, na figura, que a interação, na verdade, é entre o programa gerado a partir do ambiente de programação FURBOL e o sistema de E/S.

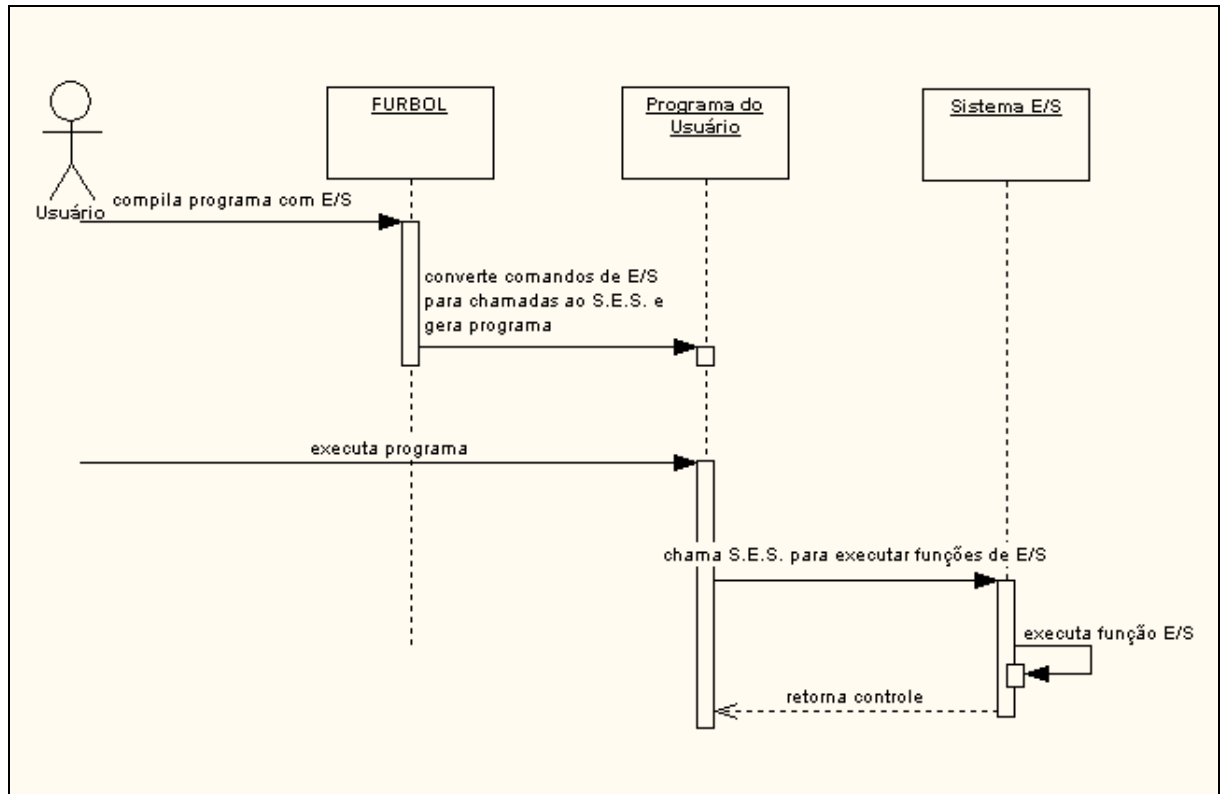


Figura 7 – Interação entre usuário, FURBOL, programa do usuário e Sistema de E/S.

A execução da função de E/S descrita no fluxograma da Figura 8 envolve uma série de passos tomados até que o comando de E/S seja realmente executado. Neste fluxograma pode-se observar o funcionamento de maneira genérica do sistema de E/S após a solicitação feita pelo programa gerado pelo FURBOL.

É importante observar que o sistema por si próprio não executa qualquer comando de E/S sem que seja feita alguma solicitação externa.

Para todos os comandos de E/S que o sistema dispõe, o fluxo de execução do sistema é similar ao da Figura 8. O código de retorno e o número de parâmetros são o que pode ser diferente entre as funções e deve ser interpretado de acordo com a definição de cada função. Quando o sistema de E/S assume o controle, ele verifica o código da função, desempilha outros eventuais parâmetros e então, chama uma função do DOS para executar o comando de

E/S. Após a execução do comando de E/S, é feita uma checagem da execução do comando e então o repassa o controle para o processo/programa que o chamou.

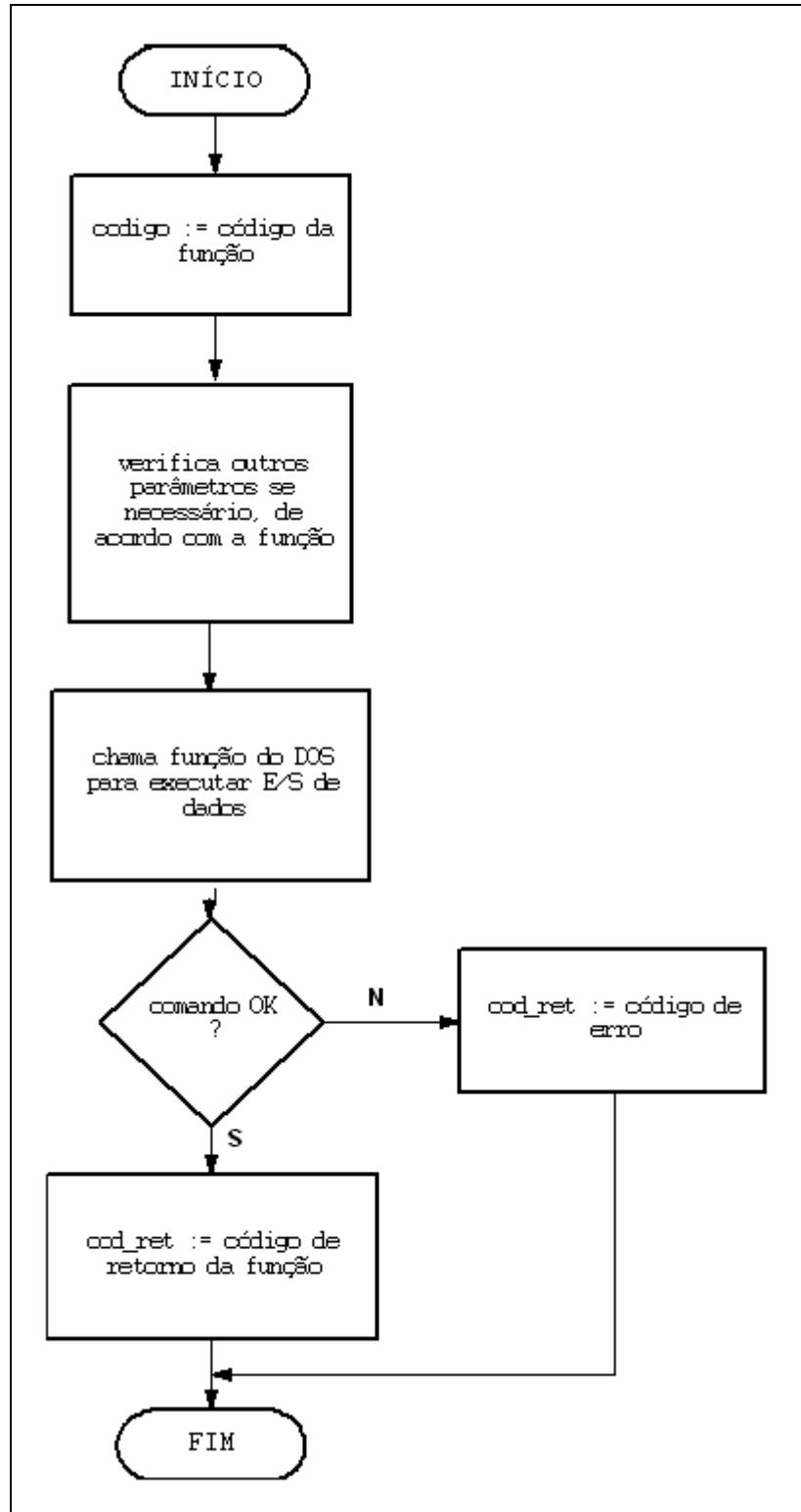


Figura 8 – Fluxograma de comportamento do sistema de E/S.

A Figura 9 demonstra o comportamento da função 'crie', que é a função de E/S para criar arquivos. A função do DOS é chamada para criar o arquivo e é validado o comando.

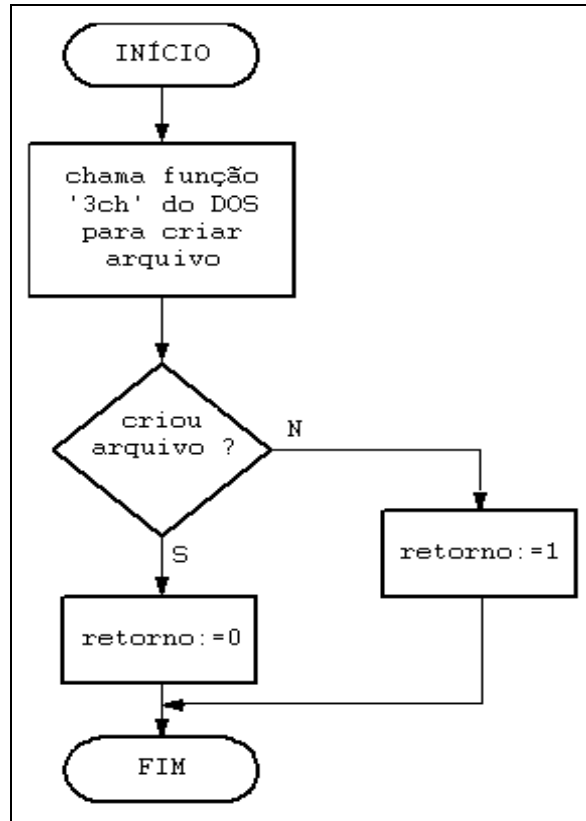


Figura 9 – Fluxograma de comportamento da função 'crie'.

O comportamento da função 'abra' é ilustrado na Figura 10. O retorno da função pode ser zero (0) ou um número maior do que zero (0) que é o *handle* do arquivo.

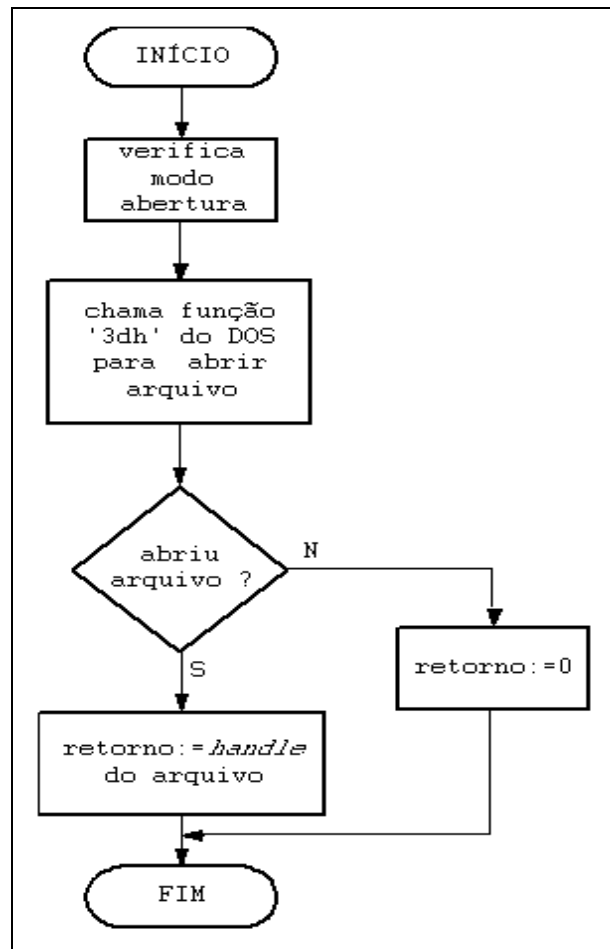


Figura 10 – Fluxograma de comportamento da função 'abra'.

A próxima figura (Figura 11) ilustra o comportamento de outra função de E/S, a função 'feche'. Esta função similar às outras já apresentadas, também utiliza uma função do DOS para fechar o arquivo. O retorno dependerá do sucesso da execução do comando.

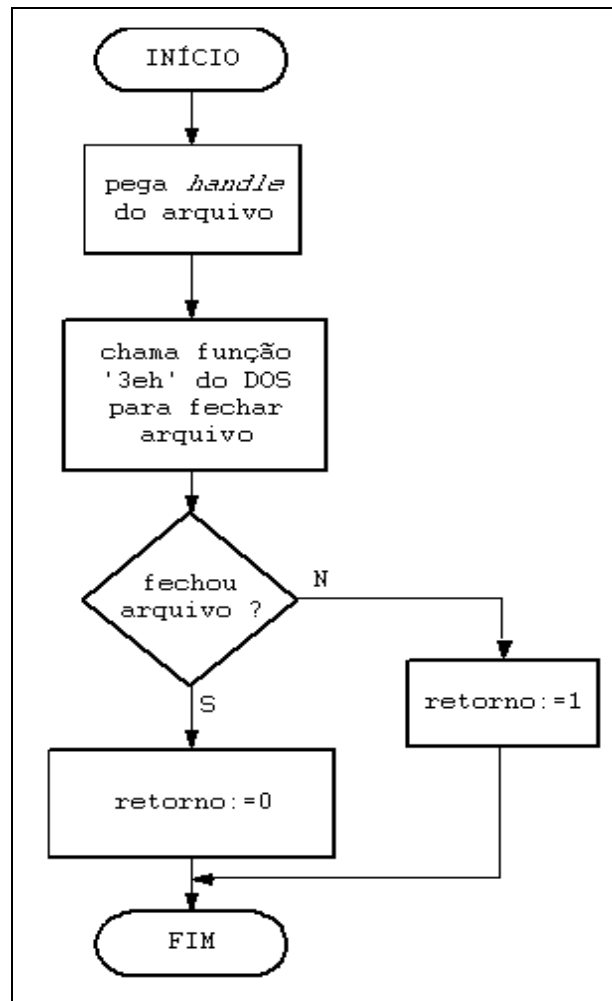


Figura 11 – Fluxograma de comportamento da função 'feche'.

A gravação do arquivo é representada na Figura 12. Esta função grava somente dados do tipo inteiro devido a limitações do FURBOL.

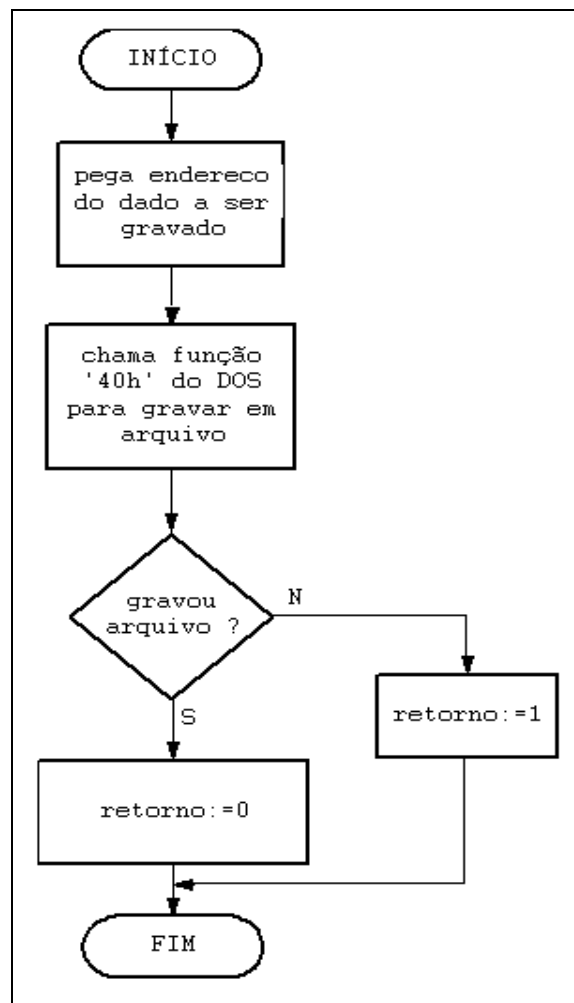


Figura 12 – Fluxograma de comportamento da função 'grave'.

A leitura do arquivo (Figura 13), após chamar a função do DOS que lê dados de um arquivo, e verifica o *status* da execução da função. Caso a operação desejada ocorreu o dado lido é movido para a área de memória do processo/programa do usuário. A função 'grave' (Figura 12) também acessa a área de memória do programa do usuário, porém, só para leitura.

Todas as funções de E/S apresentadas nos fluxogramas fazem a verificação do *status* da execução da função do DOS e retornam um valor.

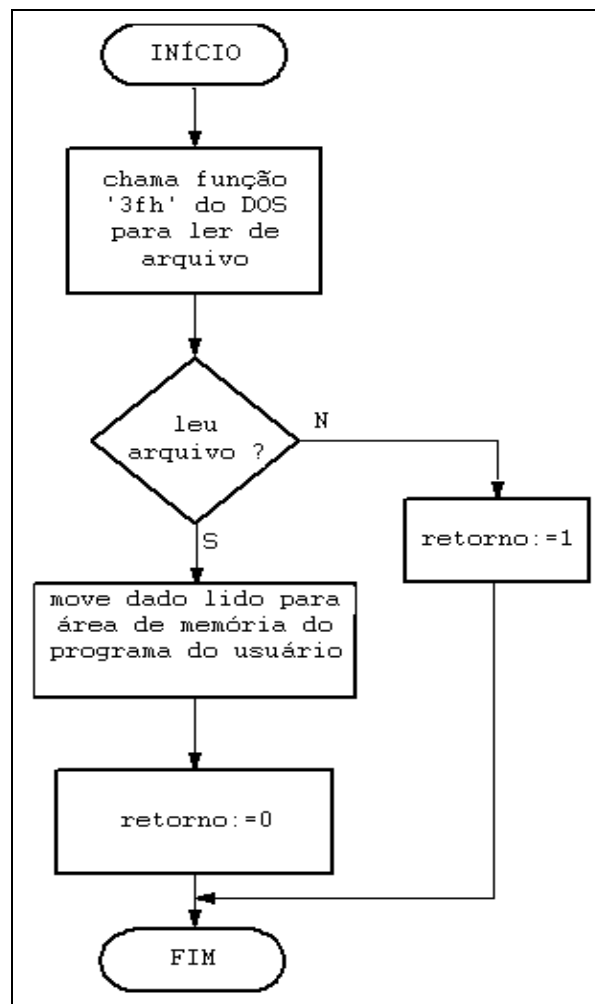


Figura 13 – Fluxograma de comportamento da função ‘leia’.

O usuário do FURBOL pode validar a execução das funções de E/S através de seu retorno.

4.1.2 COMANDOS DE E/S

O FURBOL já reconhecia dois comandos de E/S que são os comandos de leitura do teclado e escrita na tela. Para o comando de leitura o FURBOL não gera código, somente o reconhece. Já para o comando de escrita o FURBOL gera código, mas há uma limitação, o tipo de dados que o comando escreve é somente o tipo constante literal. Novos comandos de E/S foram inseridos no FURBOL. Cada comando incluído possui uma respectiva ação semântica e que deve gerar um código *Assembly* similar ao código apresentado no Quadro 1.

```

inic. variável controle
empilha parâmetros
empilha id do processo
empilha cód. função
int 60h
desempilha retorno
desempilha id do processo
desempilha parâmetros
final. variável controle

```

Quadro 1 – Exemplo de código gerado pelo FURBOL para um comando de E/S

Todos os comandos têm em comum a chamada da interrupção 60h, que é a interrupção que chama o sistema de E/S desenvolvido neste trabalho, a passagem do “PID” que é a identificação do processo que está solicitando E/S e todos movem o conteúdo do registrador de memória AX para uma variável de memória, que é um parâmetro de saída, a variável para qual o usuário do FURBOL deseja que o retorno da função seja atribuído (Ex: retorno := crie(‘arquivo’);).

Os comandos representam as funções de E/S que o sistema desenvolvido contém. Estas funções que também ficam residentes em memória são referenciadas pelo Gerenciador de E/S. As funções de E/S fazem parte da camada que contém os *drivers* dos dispositivos.

Estas funções são quem efetivamente irão executar os comandos de E/S. Elas utilizam funções do *Disk Operating System (DOS)* para executá-los. As funções de E/S representam a camada dos *drivers* dos dispositivos.

Cada função contém chamadas para funções específicas do DOS. Estas chamadas são feitas através da interrupção 21h. A rotina associada a esta interrupção contém uma série de funções que devem ser indicadas utilizando os registradores do processador. Cada função do DOS é representada por um código único. Estas funções normalmente são referenciadas utilizando-se número hexadecimal. A função de criação de arquivos, por exemplo, é a “3CH”.

Ao término de cada função do sistema de E/S do FURBOL, o gerenciador de E/S assume o controle então para repassar o resultado da operação para o processo que a solicitou.

O número de parâmetros que serão empilhados varia de acordo com o comando a ser executado. O código da função é obrigatório e cada função tem um código único. Após o empilhamento do código da função desejada, é, então, chamada a função do gerenciador através da interrupção 60h. Esta posição no vetor de interrupções contém um ponteiro para a função principal do sistema de E/S do FURBOL, o gerenciador de E/S. Todas as funções retornam um código que indica se o comando foi executado com êxito ou não.

Os comandos inseridos na linguagem FURBOL são divididos e agrupados conforme a Tabela 1.

Tabela 1 – Grupo de comandos de E/S

Dispositivos de E/S	Operações realizadas
Arquivo	Criação Abertura Fechamento Gravação Leitura
Impressora	Escrita
Teclado	Leitura
Vídeo	Escrita

Foram incluídos os novos comandos na linguagem FURBOL e ações semânticas para os mesmos.

A seguir são apresentados os comandos de E/S. Cada comando quando reconhecido pelo FURBOL gera código *Assembly*. O código gerado é na verdade uma chamada ao sistema de E/S para que este execute um comando de E/S.

Os novos comandos da linguagem FURBOL têm a seguinte sintaxe:

- a) `crie(nome_do_arquivo:constante literal):inteiro` → o retorno desta função é um número inteiro maior que zero, se o arquivo foi criado, ou zero caso contrário. A criação de um arquivo implica automaticamente em sua abertura para escrita. Desta forma é possível gravar dados no mesmo. O Quadro 2 apresenta o código gerado pela ação semântica do comando `crie`.

```

mov flag_io,1
push parametro1
push PID
push 1
int 60h
pop AX
mov variável,AX
pop PID
pop parametro1
mov flag_io,0

```

Quadro 2 – Código gerado pela ação semântica da função ‘crie’.

A variável “flag_io” é uma variável de controle do escalonador de processos do FURBOL. Se “flag_io” for 1, não mudar o contexto do processo atualmente em execução, se for 0 pode mudar de contexto. O “parâmetro1” é o endereço da área de memória que contém o nome do arquivo, composto por segmento e deslocamento. “PID” é o identificador do processo; e o número inteiro ‘1’ é o código respectivo da função. Após a chamada da interrupção que desperta o sistema de E/S, o primeiro valor desempilhado é o retorno do sistema. Para todas as funções esse retorno é movido para uma variável;

- b) abra(nome_do_arquivo: constante literal, modo_de_abertura: caractere):inteiro → esta função retorna zero caso o arquivo não possa ser aberto, ou um número inteiro maior do que zero caso o arquivo tenha sido aberto e este número é o *handle* para o arquivo. Como parâmetros esta função recebe o nome do arquivo a ser aberto e o modo em que deve ser aberto. O Quadro 3 apresenta o código gerado pela ação semântica deste comando.

```

mov flag_io,1
push parametro1
push parametro2
push PID
push 2
int 60h
pop AX
mov variável,AX
pop PID
pop parametro2
pop parametro1
mov flag_io,0

```

Quadro 3 – Código gerado pela ação semântica da função ‘abra’.

Tem-se que: ‘parâmetro1’ é o endereço na memória, composto por segmento e deslocamento, do nome do arquivo a ser criado; ‘parametro2’ é o modo de abertura do arquivo que pode ser ‘L’ (Leitura) ou ‘E’ (Escrita); o número inteiro ‘2’ é o código respectivo da função;

- c) `feche(handle_do_arquivo: inteiro):inteiro` → esta função retorna zero caso o arquivo tenha sido fechado ou um caso contrário. O código gerado pelo FURBOL para esta função é apresentado no Quadro 4.

```

mov flag_io,1
push parametro1
push PID
push 3
int 60h
pop AX
mov variavel,AX
pop PID
pop parametro1
mov flag_io,0

```

Quadro 4 – Código gerado pela ação semântica da função ‘feche’.

Tem-se que: ‘parametro1’ é um inteiro, o *handle* do arquivo; o número inteiro ‘3’ é o código da função;

- d) `grave(handle_do_arquivo: inteiro, número: inteiro):inteiro` → esta função retorna um número inteiro dois caso o valor tenha sido gravado no arquivo ou zero caso

contrário. O número inteiro dois representa a quantidade de bytes gravados no arquivo. No Quadro 5 encontra-se o código gerado para esta função.

```

mov flag_io,1
push parametro1
push parametro2
push PID
push 4
int 60h
pop AX
mov variável, AX
pop PID
pop parametro2
pop parametro1
mov flag_io,0

```

Quadro 5 – Código gerado pela ação semântica da função 'grave'

Tem-se que: 'parametro1' é o descritor do arquivo; 'parametro2' é um valor inteiro de 2 bytes; e 4 é o código da função grave;

- e) leia(handle_do_arquivo: inteiro, variável: inteiro):inteiro → esta função retorna zero se a leitura ocorreu com sucesso ou um caso contrário. O segundo parâmetro é onde deve estar o conteúdo lido do arquivo. O Quadro 7 apresenta o código para esta função.

```

mov flag_io,1
push parametro1
push parametro2
push PID
push 5
int 60h
pop AX
mov variável,AX
pop PID
pop parametro2
pop parametro1
mov flag_io,0

```

Quadro 6 – Código gerado pela ação semântica da função 'leia'

Tem-se que: ‘parametro1’ é o *handle* do arquivo, ‘parametro2’ é o endereço composto por segmento e deslocamento da variável que irá receber o conteúdo lido do arquivo e 4 é o código da função grave.

Foi preciso alterar a especificação da linguagem FURBOL para contemplar os novos comandos. Os Quadros 7, 8, 9 e 10 apresentam os trechos da BNF do FURBOL que foram alterados. A BNF completa da linguagem FURBOL pode ser encontrada em SILVA (2002).

Comando	→	id
ChamProc / L, Atribuicao		<p><i>Simb</i> := <i>Simbolos.ProcurarSimbolo</i>(<i>id.nome</i>); Se <i>Simb.tipo</i> = <i>tsProcedimento</i> então <i>Comando.codigo</i> := <i>ChamProc.codigo</i> <i>GeraCodigoEloEstatico</i> 'chamada' <i>id.nome</i>; <i>Comando.codasm</i> := <i>ChamProc.codasm</i> <i>GeraCodigoEloEstatico</i> 'call' <i>id.nome</i>; Senão Se <i>Simb.tipo</i> = <i>tsTarefa</i> então <i>Comando.codigo</i> := 'disparar(' <i>id.nome</i> ')'; <i>Comando.codasm</i> := 'push ax' <i>CRLF</i> 'mov ax,offset ' <i>id.nome</i> <i>CRLF</i> 'push ax' <i>CRLF</i> 'call disparar_proc' <i>CRLF</i> 'pop ax' <i>CRLF</i>; Senão <i>Comando.codasm</i> := 'push offset' <i>id.nome</i> <i>CRLF</i> <i>Comando.codigo</i> := <i>Atribuicao.codigo</i>; <i>Comando.codasm</i> := <i>Atribuicao.codasm</i>;</p>
CCrie		
Virgula		<p><i>Comando.codigo</i> := <i>CCrie.codigo</i> <i>Virgula.codigo</i>; <i>Comando.codasm</i> := <i>CCrie.codasm</i> <i>Virgula.codasm</i>;</p>
CAbra		
Virgula		<p><i>Comando.codigo</i> := <i>CAbra.codigo</i> <i>Virgula.codigo</i>; <i>Comando.codasm</i> := <i>CAbra.codasm</i> <i>Virgula.codasm</i>;</p>
CFech		
Virgula		<p><i>Comando.codigo</i> := <i>CFech.codigo</i> <i>Virgula.codigo</i>; <i>Comando.codasm</i> := <i>CFech.codasm</i> <i>Virgula.codasm</i>;</p>
CGrave		
Virgula		<p><i>Comando.codigo</i> := <i>CGrave.codigo</i> <i>Virgula.codigo</i>; <i>Comando.codasm</i> := <i>CGrave.codasm</i> <i>Virgula.codasm</i>;</p>
CLeia		
Virgula		<p><i>Comando.codigo</i> := <i>CLeia.codigo</i> <i>Virgula.codigo</i>; <i>Comando.codasm</i> := <i>CLeia.codasm</i> <i>Virgula.codasm</i>;</p>

Quadro 7 – Definição dos Comandos da Linguagem

<i>Virgula</i>	→	','	
		<i>Comando</i>	<i>Virgula.codigo := Comando.codigo</i> <i>Virgula.codasm := Comando.codasm</i>
		^	
<i>Atribuicao</i>	→	':='	Se <i>Atribuicao.deslocamento</i> <> '' então <i>RI := Novo_T</i> ; <i>IndCodAsm := 'mov ' RI ', ' Ldesloca CRLF 'push ' RI</i> ; <i>PreIdAt := 'pop di'</i> ;
		<i>Atributo</i>	<i>Atribuicao.codigo := Atributo.codigo</i> <i>Atribuicao.codasm := Atributo.codasm</i>
<i>Atributo</i>	→	<i>Expressao</i>	<i>E.local := Expressao.local</i> ; Se <i>Expressao.tipo</i> <> <i>Atributo.tipo</i> então erro ; Se <i>Expressao.deslocamento</i> <> '' então <i>Atributo.codigo := gerar(idAT['L.deslocamento']:=E.local)</i> ; Senão <i>Atributo.codigo := gerar(IdAT ':=' E.Local)</i> ;
		<i>Comando_ES</i>	<i>RX := Registrador</i> ; <i>Atributo.codasm := IndCodAsm Expressao.codasm AtPrelocal CRLF </i> <i>'mov ' RX ', ' aTlocal CRLF </i> <i>PreIdAT CRLF </i> <i>'mov ' IdAT ', ' RX</i> ;
			<i>Atributo.codasm := Comando_ES.codasm</i> ; <i>Atributo.codasm := Atributo.codasm + 'pop BP' CRLF</i> ; <i>Atributo.codasm := Atributo.codasm + 'mov [BP],AX' CRLF</i> ;
<i>Comando_ES</i>	→	<i>CCrie</i>	<i>Comando_ES.codasm := CCrie.codasm</i>
		<i>CAbra</i>	<i>Comando_ES.codasm := CAbra.codasm</i>
		<i>CFeche</i>	<i>Comando_ES.codasm := CFeche.codasm</i>
		<i>CGrave</i>	<i>Comando_ES.codasm := CGrave.codasm</i>
		<i>CLeia</i>	<i>Comando_ES.codasm := CLeia.codasm</i>
<i>CCrie</i>	→	'crie'	
		('	
		<i>ConstanteLiteral</i>	<i>Nome := EmiteDados(ConstanteLiteral)</i> ;
)'	<i>CCrie.codasm := 'mov AX' 'offset' Nome CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'push AX' CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'mov AX,1' CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'push AX' CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'int 60' CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'pop AX' CRLF</i> ; <i>CCrie.codasm := CCrie.codasm + 'pop AX' CRLF</i> ;

Quadro 8 – Definição dos Comandos de Atribuição

<i>CAbra</i>	→	'abra'	
		'('	
		ConstanteLit Nome := EmiteDados(ConstanteLiteral);	
		eral	
		','	
		<i>Modo</i>	
)'	<i>CAbra.codasm</i> := 'mov AX' 'offset' Nome CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'push AX' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'mov AX' ',' <i>Modo</i> CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'push AX' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'mov AX,2' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'push AX' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'int 60' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'pop AX' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'pop AX' CRLF; <i>CAbra.codasm</i> := <i>CAbra.codasm</i> + 'pop AX' CRLF;
<i>Modo</i>	→	'E'	
		'L'	
<i>CFeche</i>	→	'feche'	
		'('	
		id	Se <i>Simbolos.ProcurarSimbolo</i> (id.nome) = nil então erro('Identificador não declarado');
)'	<i>CFeche.codasm</i> := 'mov AX' ',' id.nome CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'push AX' CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'mov AX,3' CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'push AX' CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'int 60' CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'pop AX' CRLF; <i>CFeche.codasm</i> := <i>CFeche.codasm</i> + 'pop AX' CRLF;
<i>CGrave</i>	→	'grave'	
		'('	
		id	Se <i>Simbolos.ProcurarSimbolo</i> (id.nome) = nil então erro('Identificador não declarado');
		Senão	<i>CGrave.codasm</i> := 'mov AX' ',' id.nome CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'push AX' CRLF;
		','	
		id	Se <i>Simbolos.ProcurarSimbolo</i> (id.nome) = nil então erro('Identificador não declarado');
		Senão	<i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'mov AX' ',' id.nome CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'push AX' CRLF;
)'	<i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'mov AX,1' CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'push AX' CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'int 60' CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'pop AX' CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'pop AX' CRLF; <i>CGrave.codasm</i> := <i>CGrave.codasm</i> + 'pop AX' CRLF;

Quadro 9 – Definição dos Comandos de Entrada e Saída

<i>CLeia</i>	→	'leia'
		('
id		Se <i>Simbolos.ProcurarSimbolo(id.nome) = nil</i> então <i>erro('Identificador não declarado');</i> Senão <i>CLeia.codasm := 'mov AX' ';' id.nome CRLF;</i> <i>CLeia.codasm := CLeia.codasm + 'push AX' CRLF;</i>
		;
id		Se <i>Simbolos.ProcurarSimbolo(id.nome) = nil</i> então <i>erro('Identificador não declarado');</i> Senão <i>CLeia.codasm := CLeia.codasm + 'mov AX' ';' 'offset' id.nome CRLF;</i> <i>CLeia.codasm := CLeia.codasm + 'push AX' CRLF;</i>
)'
		<i>CAbra.codasm := CAbra.codasm + 'mov AX,1' CRLF;</i> <i>CAbra.codasm := CAbra.codasm + 'push AX' CRLF;</i> <i>CAbra.codasm := CAbra.codasm + 'int 60' CRLF;</i> <i>CAbra.codasm := CAbra.codasm + 'pop AX' CRLF;</i> <i>CAbra.codasm := CAbra.codasm + 'pop AX' CRLF;</i> <i>CAbra.codasm := CAbra.codasm + 'pop AX' CRLF;</i>

Quadro 10 – Definição dos Comandos de Entrada e Saída (Continuação)

4.2 IMPLEMENTAÇÃO

O principal objetivo do trabalho foi disponibilizar comandos de E/S de dados para os usuários do FURBOL. Após um estudo detalhado, decidiu-se desenvolver o sistema de E/S procurando fazer o mínimo possível de alterações na definição da linguagem de programação FURBOL. O sistema foi desenvolvido baseado na arquitetura de camadas apresentada por Tanenbaum (1992), que é composto por quatro camadas conforme já descrito neste trabalho. As camadas do sistema são compostas por um conjunto de rotinas desenvolvidas nas linguagens de programação *Pascal* e *Assembly* que são iniciadas e ficam residentes em memória quando o núcleo do FURBOL é iniciado. Um gerenciador de E/S e funções de E/S compõem o sistema. As próximas seções irão detalhar o funcionamento do sistema.

O sistema desenvolvido funciona independentemente do ambiente de programação FURBOL e pode vir a ser utilizado por outros sistemas futuramente. Também foram necessárias algumas alterações na especificação e implementação da linguagem FURBOL para que pudesse interagir com o sistema de E/S.

4.2.1 SISTEMA DE E/S

O sistema de entrada e saída do FURBOL é composto por um gerenciador de E/S que representa a camada de software independente de dispositivo e mais as funções de E/S. O sistema contém ainda algumas estruturas de controle para gerenciar o acesso aos dispositivos que fazem parte do gerenciador de E/S.

A figura 14 ilustra as camadas do sistema de E/S.

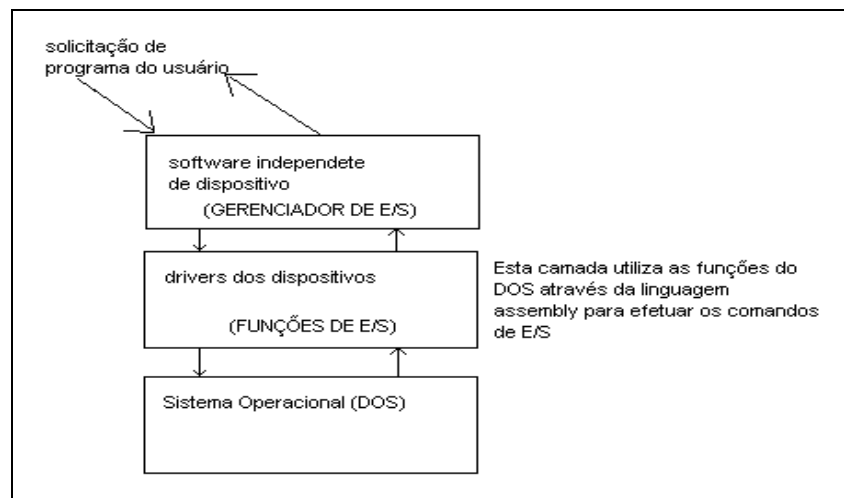


Figura 14 - Camadas do Sistema de E/S.

4.2.2 GERENCIADOR DE E/S

O Gerenciador de E/S é uma rotina que fica residente em memória e é a rotina principal do sistema de E/S. Os programas escritos na linguagem FURBOL irão interagir somente com ela, que funciona como uma fachada para o sistema de E/S. A existência do gerenciador se faz necessário, pois é preciso que sejam tomadas algumas medidas de precaução para cada comando de E/S específico disparado pelo FURBOL para que os dispositivos sejam utilizados de maneira adequada.

A principal função do gerenciador é receber solicitações de comandos de E/S dos processos e responder aos mesmos assim que o comando estiver terminado, podendo passar alguns parâmetros, como resultado da operação e endereço de dados, por exemplo.

Para que o gerenciador seja chamado é preciso invocar a interrupção 60h. Este número foi escolhido, pois é uma posição no vetor de interrupções que não é utilizada pelo sistema operacional. A posição 60h irá conter o endereço da rotina principal do gerenciador de E/S do FURBOL.

Existem dispositivos que podem ser compartilhados por diferentes processos, para estes o gerenciador mantém algumas tabelas de acesso para que os mesmos sejam utilizados da maneira correta. Por exemplo, se o processo1 deseja ler o conteúdo do arquivo1 e simultaneamente o processo2 deseja ler o conteúdo do arquivo2, o gerenciador mantém uma tabela com os arquivos abertos, o modo de abertura e os processos que estão utilizando esses arquivos conforme Tabela 2. A tabela contém o *handle* do arquivo, o nome do arquivo, o seu modo de abertura e o identificador do processo que o abriu. Desta forma é possível que o sistema monitore as atividades nos arquivos através do identificador do processo. Nenhum processo pode ler o conteúdo de um arquivo se este não estiver na tabela de arquivos relacionado com o identificador do processo.

Tabela 2 - Arquivos Abertos.

<i>handle</i>	Nome	Modo	PID
5	C:\tmp\arquivo.txt	'L'	1
7	D:\testes\fl.l	'E'	2
9	A:\bd.dat	'A'	3
8	C:\temp\itens\it_01.dat	'L'	4
10	Impressora	'E'	6
...
...

Por outro lado, existem dispositivos que não podem ser utilizados por diferentes processos simultaneamente. Um exemplo claro disso é a impressora, que precisa iniciar e terminar um trabalho para que outro possa ser iniciado. Para resolver este problema, foi definido que a impressora será tratada de forma similar a um arquivo. Somente um processo poderá utilizá-la em um determinado momento e sendo assim, o gerenciador irá utilizar uma linha da tabela de arquivos que representará a impressora e só poderá estar aberta por um processo no modo de escrita ('E').

As estruturas de controle de uso dos dispositivos de E/S mantidas pelo gerenciador fazem parte do software independente de dispositivo. O Gerenciador de E/S faz parte da camada de software independente de dispositivo.

4.3 ALTERAÇÕES NO AMBIENTE FURBOL

Apesar de o ambiente de programação FURBOL ser um ambiente que possibilita a criação de programas com gerenciamento de processos concorrentes, os comandos de E/S serão realizados de forma atômica, ou seja, pode existir um ou mais processos concorrentes

em um programa FURBOL, porém, somente um irá realizar um comando de E/S em um determinado instante e realizará o comando até o término da E/S.

Para isto foi necessário criar um *flag* de execução de comandos de E/S que será utilizado pelo escalonador de processos do FURBOL e irá indicar se o programa está executando um comando de E/S ou não.

O escalonador do FURBOL continuará sendo chamado a cada interrupção gerada pelo *timer*, porém ele só irá realizar efetivamente o escalonamento de outro processo se o *flag* de comando de E/S estiver 'apagado'.

Desta forma o programa deixará de ser concorrente quando da realização de um comando de E/S. Esta solução foi adotada levando-se em consideração o grau de complexidade das medidas que teriam de ser tomadas para gerenciar a E/S e o tempo para a conclusão do trabalho.

4.4 RESULTADOS

Os resultados obtidos com a integração entre FURBOL e o Sistema de E/S serão demonstrados através de um exemplo utilizando-se o comando 'crie'.

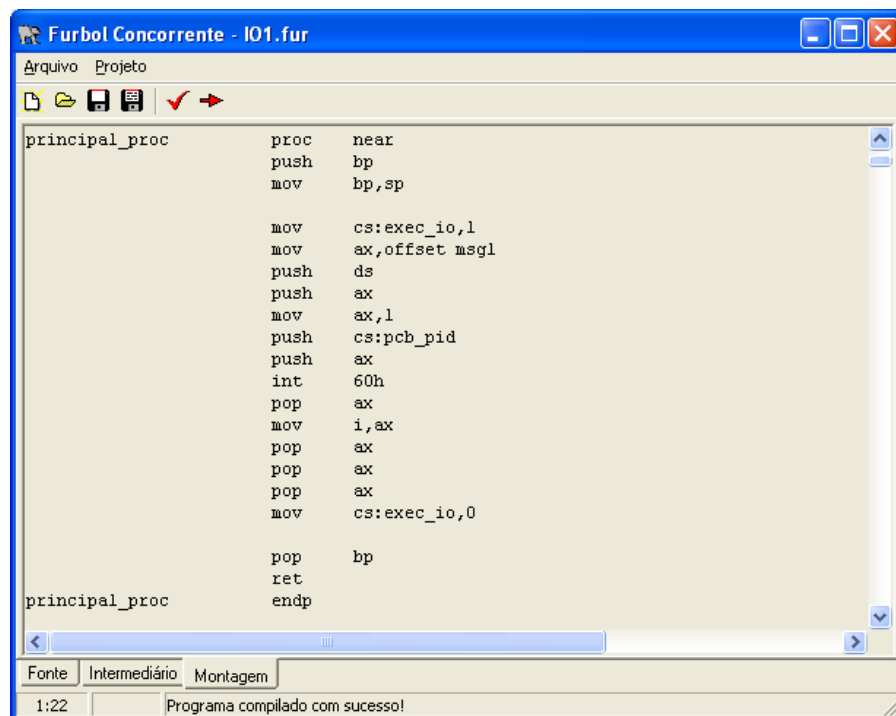
A figura 15 apresenta o código fonte de um programa na linguagem FURBOL. Este é o código para se criar um arquivo.



```
programa teste;
var i:inteiro;
inicio
  i:=crie("c:\tmp\templ.txt");
fim.
```

Figura 15 – Código para criar um arquivo no FURBOL.

Na figura 16 é possível ver o trecho de código *Assembly* gerado pelo FURBOL para este programa.



```

Furbo1 Concorrente - IO1.fur
Arquivo Projeto
principal_proc      proc      near
                    push     bp
                    mov      bp,sp

                    mov      cs:exec_io,1
                    mov      ax,offset msg1
                    push     ds
                    push     ax
                    mov      ax,1
                    push     cs:pcb_pid
                    push     ax
                    int      60h
                    pop      ax
                    mov      i,ax
                    pop      ax
                    pop      ax
                    pop      ax
                    mov      cs:exec_io,0

                    pop      bp
                    ret
principal_proc      endp

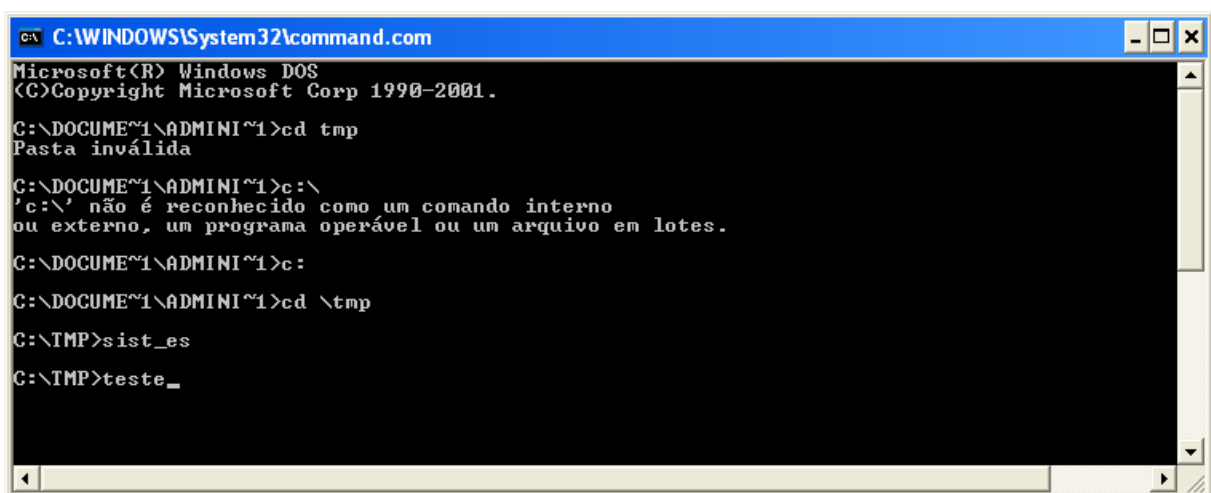
Fonte  Intermediário  Montagem
1:22   Programa compilado com sucesso!

```

Figura 16 – Trecho de código assembly gerado pelo FURBOL para a função `crie`.

No código gerado pelo FURBOL são empilhados os parâmetros necessários para a execução do comando de E/S e em seguida é feita uma chamada ao sistema de E/S através da instrução ‘`int 60h`’. Esta instrução gera uma interrupção de software que irá ativar o sistema de E/S. Para isso, é preciso que o sistema de E/S já tenha sido inicializado e esteja residente em memória no momento da execução do programa.

A figura 17 apresenta a inicialização do sistema de E/S e em seguida a execução do programa gerado a partir do código da figura 15.



```

C:\WINDOWS\System32\command.com
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>cd tmp
Pasta inválida

C:\DOCUME~1\ADMINI~1>c:\
'c:\' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\DOCUME~1\ADMINI~1>c:
C:\DOCUME~1\ADMINI~1>cd \tmp
C:\TMP>sist_es
C:\TMP>teste_

```

Figura 17 - Inicialização de Sistema de E/S e execução de programa gerado no FURBOL.

O resultado obtido é apresentado na figura 18 com o arquivo criado conforme especificado no programa.

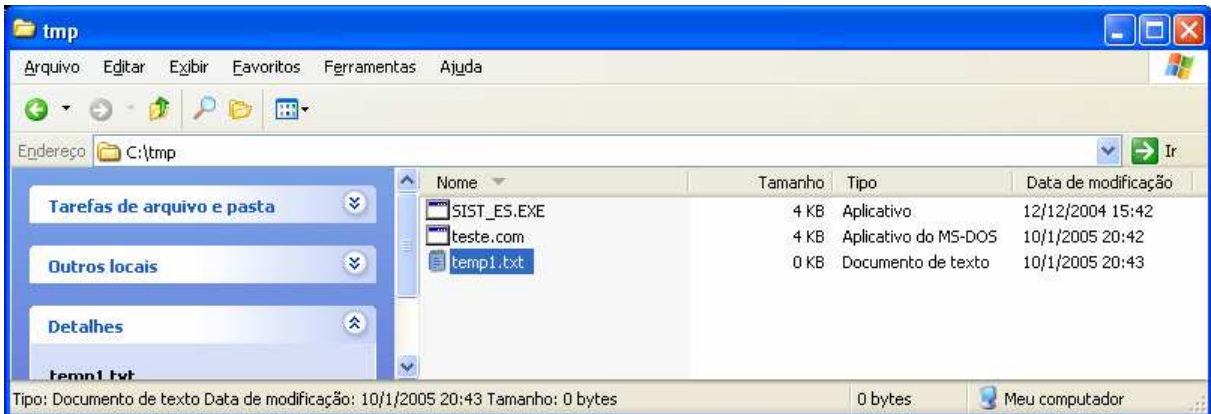


Figura 18 - Resultado da execução do programa.

As próximas ilustrações são exemplos de outros comandos de E/S que o compilador do FURBOL reconhece. O código *Assembly* gerado pelas ações semânticas também pode ser visualizado.

A figura 19 apresenta o código para a abertura de um arquivo no FURBOL, em seguida a figura 20 apresenta o código *assembly* gerado pelo FURBOL.



Figura 19 - Código para abrir um arquivo no FURBOL.

```

principal_proc      proc      near
                   push     bp
                   mov      bp,sp

                   mov      cs:exec_io,1
                   mov      ax,'E'
                   push     ax
                   mov      ax,offset msg1
                   push     ds
                   push     ax
                   mov      ax,2
                   push     cs:pch_pid
                   push     ax
                   int      60h
                   pop      ax
                   mov      i,ax
                   pop      ax
                   pop      ax
                   pop      ax
                   pop      ax
                   mov      cs:exec_io,0

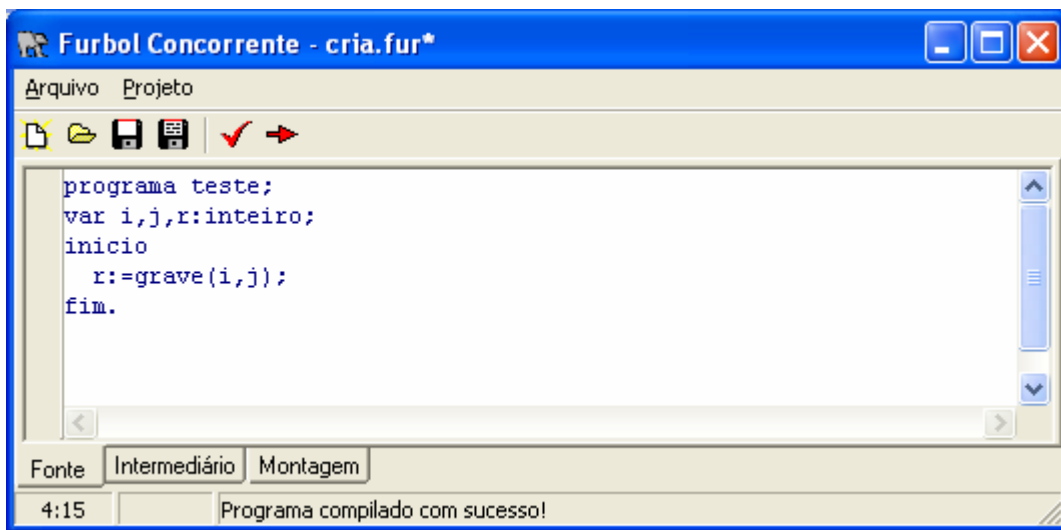
                   pop      bp
                   ret
principal_proc      endp

```

4:33 Programa compilado com sucesso!

Figura 20 - trecho do código assembly gerado para a função 'abra'.

O código para gravação de arquivo no FURBOL deve ser conforme exemplo da figura 21.



The screenshot shows a window titled 'Furbol Concorrente - cria.fur*'. The menu bar includes 'Arquivo' and 'Projeto'. Below the menu bar are icons for file operations. The main text area contains the following code:

```

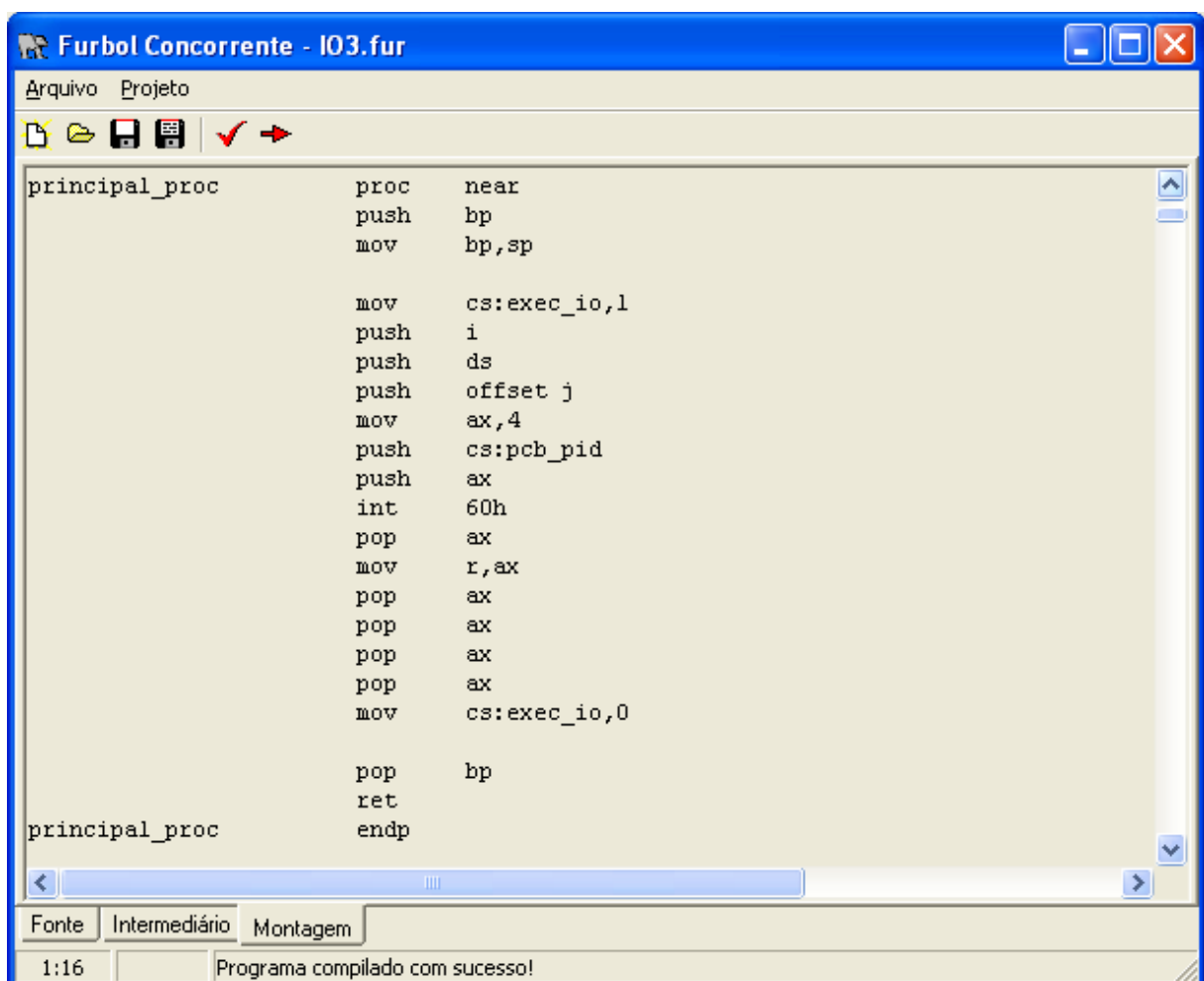
programa teste;
var i,j,r:inteiro;
inicio
  r:=grave(i,j);
fim.

```

At the bottom, there are tabs for 'Fonte', 'Intermediário', and 'Montagem'. The status bar shows the time '4:15' and the message 'Programa compilado com sucesso!'.

Figura 21 - Código para gravar dados em um arquivo no FURBOL.

O código gerado pelo compilador do FURBOL para a função 'grave' pode ser visualizado na figura 22.



The screenshot shows a window titled 'Furbol Concorrente - IO3.fur'. The menu bar includes 'Arquivo' and 'Projeto'. Below the menu bar are icons for file operations. The main text area contains the following assembly code:

```

principal_proc      proc      near
                   push     bp
                   mov      bp,sp

                   mov      cs:exec_io,1
                   push     i
                   push     ds
                   push     offset j
                   mov      ax,4
                   push     cs:pcb_pid
                   push     ax
                   int      60h
                   pop      ax
                   mov      r,ax
                   pop      ax
                   pop      ax
                   pop      ax
                   pop      ax
                   mov      cs:exec_io,0

                   pop      bp
                   ret
principal_proc      endp

```

At the bottom, there are tabs for 'Fonte', 'Intermediário', and 'Montagem'. The status bar shows the time '1:16' and the message 'Programa compilado com sucesso!'.

Figura 22 - Trecho do código assembly gerado para a função 'grave'.

5 CONCLUSÕES

A interação entre o FURBOL e o sistema de E/S desenvolvido neste trabalho exigiu especial atenção para o tratamento de dados. O sistema de E/S acessa a área de dados dos programas que interagem com ele. Foi preciso rever os conceitos e teorias estudados no curso nas disciplinas de arquitetura de computadores e sistemas operacionais para que não houvesse grandes complicações no desenvolvimento do protótipo.

A proposta deste trabalho foi atingida, que era a de implementar uma estrutura de E/S para o ambiente FURBOL. A implementação de saída para terminal e saída para impressora não foi concluída, porém, com os comandos criados para manipulação de arquivos é possível realizar E/S nestes dispositivos. O trabalho servirá como um bom ponto de partida para futuros estudos relacionados a aspectos de Entrada e Saída de dados. O sistema dá uma visão prática da teoria de E/S discutida na fundamentação teórica e também na disciplina de Sistemas Operacionais durante o curso na Universidade. Grande parte da teoria discutida na Universidade é fundamentada nas obras de Tanenbaum assim também o faz o sistema implementado neste trabalho.

Procurou-se fazer com que a interação entre os programas escritos no FURBOL e o sistema de E/S fosse da forma mais fácil possível. Desta forma decidiu-se por desenvolver um sistema de E/S que funciona independentemente do FURBOL e a interação entre eles é feita através de chamadas para o sistema de E/S pelos programas gerados no FURBOL. Como o sistema é independente, ele pode vir a ser utilizado por outras linguagens de programação. Uma possibilidade interessante é a de adaptar ambientes de programação já existentes e de código fonte aberto para a interação com este sistema, já que o mesmo foi desenvolvido na Universidade e o seu funcionamento é de total conhecimento. É importante salientar que a adaptação de um ambiente de programação desconhecido para o sistema de E/S exigirá um esforço considerável, porém é possível.

As principais limitações do sistema são que ele ainda contém um número pequeno de funções de E/S e também é baseado no sistema operacional DOS. O tipo de dado aceito nas operações de E/S é o tipo inteiro. Esta é uma limitação que já existe na versão do FURBOL escolhida para interagir com o sistema.

Espera-se que o trabalho seja de grande valia para o curso de Ciências da Computação da Universidade Regional de Blumenau e que problemas encontrados neste possam ser identificados e resolvidos em futuros trabalhos gerados na Universidade.

5.1 EXTENSÕES

A incorporação de novas funções de E/S no sistema originado a partir deste trabalho é uma sugestão de trabalho futuro. Como segunda sugestão, seria a conversão deste sistema que é totalmente baseado no sistema operacional DOS para outro sistema operacional, ou ainda, a desvinculação do sistema de E/S com qualquer sistema operacional, ou seja, fazer com que o sistema passe a utilizar funções do BIOS para efetuar os comandos de E/S. Sugere-se também que sejam implementadas melhorias nas funções de E/S como, por exemplo, fazer com que as mesmas trabalhem com o tipo de dados *string*. Uma proposta levantada na conclusão deste trabalho e que parece ser de bastante valor para a Universidade seria a de fazer com que algum ambiente de programação de código fonte aberto utilize este sistema para efetuar comandos de E/S.

REFERÊNCIAS BIBLIOGRÁFICAS

ADRIANO, Anderson. **Implementação de mapeamento finito (array's) no ambiente FURBOL**. 2001. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Universidade Regional de Blumenau, Blumenau.

AHO, Alfred V; SETHI, Ravi; ULLMAN, Jeffery D. **Compilers: principles, techniques and tools**. Massachusetts: Addison-Wesley, 1988.

ANDRE, Geovanio Batista. **Protótipo de gerador de código executável a partir do ambiente FURBOL**. 2000. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Universidade Regional de Blumenau, Blumenau.

BIEGING, Stephan D. **Implementação de mapeamento finito (arrays) dinâmico no ambiente FURBOL**. 2002. 116 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Universidade Regional de Blumenau, Blumenau.

IBM United States. New York, 2004. Disponível em: <http://www.ibm.com/us/>. Acesso em: 20 dez. 2004.

PINKERT, James R; WEAR, Larry L. **Operating systems: concepts, policies, and mechanisms**. Englewood Cliffs: Prentice-Hall, 1989.

SEBESTA, Robert W. **Concepts of programming languages**. 4th ed. Massachusetts: Addison-Wesley, 1999.

SILBERSCHATZ, Abraham; PETERSON, James L; GALVIN, Peter B. **Operating systems concepts**. 3rd ed. Massachusetts: Addison-Wesley, 1991.

SILVA, Paulo. **Implementação de unidades para processos concorrentes no ambiente FURBOL**. 2002. 169 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Universidade Regional de Blumenau, Blumenau.

TANENBAUM, Adrew S. **Modern operating systems**. Englewood Cliffs: Prentice-Hall, 1992.

TISCHER, Michael. **PC system programming: an in-depth reference for the DOS programmer**. Grand Rapids: Abacus, 1990.

APÊNDICE A –Código Fonte do Gerenciador de E/S

<pre> procedure geren_es; interrupt; begin {desempilha codigo da função a posição de memória "es:244" é onde está guardado o endereço do segmento de dados do sistema de E/S. A cada vez que uma interrupção desperta o sistema ele restaura o registros de dados para poder acessar suas variáveis internas. A variável "stack_pos" é uma constante (24) que indica o deslocamento do início dos parâmetros passados ao sistema em relação ao endereço atual da pilha. } asm mov ax,0 mov es,ax mov ax,es:[244] mov ds,ax mov ax,[bp+stack_pos] mov codigo,ax mov ax,[bp+stack_pos+2] mov PID,al end; case codigo of 1:begin ret:=-1; asm mov ax,[bp+stack_pos+4] mov arquivo,ax mov ax,[bp+stack_pos+6] mov newsgmnt,ax end; ret:=crie(newsgmnt,arquivo); end; 2:begin ret:=0; asm mov ax,[bp+stack_pos+4] mov arquivo,ax mov ax,[bp+stack_pos+6] mov newsgmnt,ax mov ax,[bp+stack_pos+8] mov modo,al end; nome_arq(newsgmnt,arquivo); if not(arq_aberto(modo2,PID2)) then ret:=abra(newsgmnt,arquivo,modo); end; 3:begin ret:=0; asm mov ax,[bp+stack_pos+4] mov hndl,ax end; </pre>	<pre> 4:begin ret:=0; asm mov ax,[bp+stack_pos+4] mov arquivo,ax mov ax,[bp+stack_pos+6] mov newsgmnt,ax mov ax,[bp+stack_pos+8] mov hndl,ax end; if (hndl_valido(hndl,modo2,PID2)) then if (modo='E') and (PID2=PID) then ret:=grave(hndl,newsgmnt,arquivo); end; 5:begin ret:=0; asm mov ax,[bp+stack_pos+4] mov arquivo,ax mov ax,[bp+stack_pos+6] mov newsgmnt,ax mov ax,[bp+stack_pos+8] mov hndl,ax end; if (hndl_valido(hndl,modo2,PID2)) then if (modo='L') and (PID2=PID) then ret:=leitura(hndl,newsgmnt,arquivo); end; else begin ret:=-2; {Opcao invalida} end; end; {returno da execucao da funcao} asm mov ax,ret mov [bp+stack_pos],ax end; end; { Programa Principal } begin { Inicializacao do Sistema } asm mov ax,0 mov es,ax mov ax,ds mov es:[244],ax end; num_arq:=0; setintvec(\$60,@geren_es); keep(0); end. </pre>
---	--

APÊNDICE B –Código Fonte da Função ‘crie’

```
function crie(segmento,deslocamento:word):integer;
begin
  cod:=0;
  asm
    xor cx,cx
    push ds
    mov ds,segmento
    mov dx,deslocamento
    mov ah,3ch
    int 21h
    pop ds
    jnc @fim1
    mov cod,1
  @fim1:
  end;
  crie:=cod;
end;
```

APÊNDICE C –Código Fonte da Função ‘abra’

```
function abra(segmento,deslocamento:word; md:char):integer;
begin
  case md of
    'L':
      num_bytes:=0;
    'E':
      num_bytes:=1;
  end;
  asm
    push ds
    mov ds,segmento
    mov dx,deslocamento
    mov al,num_bytes
    mov ah,3dh
    int 21h
    pop ds
    jnc @fima
    mov cod,0
    jmp @fima2
  @fima:
    mov cod,ax
  @fima2:
    end;
  if not(cod=0) then
  begin
    inc(num_arq);
    tab_arq[num_arq].nome:=nome;
    tab_arq[num_arq].modo:=md;
    tab_arq[num_arq].handle:=cod;
    tab_arq[num_arq].PID:=PID;
  end;
  abra:=cod;
end;
```

APÊNDICE D –Código Fonte da Função ‘feche’

```
function feche(dcr:integer):integer;
begin
  asm
    mov bx,dcr
    mov ah,3eh
    int 21h
    jnc @fimf
    mov cod,1
    jmp @fimf2
  @fimf:
    mov cod,0
  @fimf2:
  end;
  feche:=cod;
end;
```


APÊNDICE E –Código Fonte da Função ‘grave’

```
function grave(dcr,segmento,deslocamento:word):integer;
begin
  asm
    mov cx,2
    mov bx,dcr
    push ds
    mov ds,segmento
    mov dx,deslocamento
    mov ah,40h
    int 21h
    pop ds
    jnc @fimg
    mov cod,ax
    jmp @fimg2
  @fimg:
    mov cod,2
  @fimg2:
    end;
  grave:=cod;
end;
```

APÊNDICE F –Código Fonte da Função ‘leia’

```
function leia(dcr,segmento,deslocamento:word):integer;
begin
  asm
    mov cx,2
    mov bx,dcr
    push ds
    mov ds,segmento
    mov dx,deslocamento
    mov ah,3fh
    int 21h
    pop ds
    jnc @fim1
    mov cod,1
    jmp @fim2
  @fim1:
    mov cod,0
  @fim2:
    end;
  leia:=cod;
end;
```