TERMINAL INTELIGENTE:
SISTEMA MULTI-TERMINAL

Alcindo Ferreira Filho

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M. Sc.).

Aprovada por:

Guilherme Chagas Rodrigues
(Presidente)

Nelson Maculan Filho

Sueli Mendes dos Santos

RIO DE JANEIRO, RJ - BRASIL
OUTUBRO DE 1978

FERREIRA FILHO, ALCINDO
Terminal Inteligente: Sistema Multi-Terminal [Rio de Janeiro] 1978.
IX, 109p. 29,7cm (COPPE-UFRJ, Sc, Engenharia de Sistemas, 1978)

Tese - Univ. Fed. Rio de Janeiro. Coordenação dos Programas de Pós-Graduação em Engenharia

1.*Software* Básico I.COPPE/UFRJ II.Título (série).

A G R A D E C I M E N T O S

. Ao Prof. GUILHERME CHAGAS RODRIGUES, que nos orientou no desenvolvimento deste trabalho com dedicação incomum.

. À equipe de desenvolvimento e manutenção do Sistema Operacional do TI (SOCO) - PAULO IV, ANTONIO II e FRANCISCO DUTRA - que prestou um suporte inestimável para a elaboração do SMT.

. Ao pessoal do SERPRO - o Diretor de Desenvolvimento, GILCIO ROBERTO AMARAL MARTINS e os colegas da Assessoria de Apoio ao Desenvolvimento, HENRIQUE BORK e JOSÉ LUIZ THADEU - pelo incentivo e apoio constantes.

. À minha mulher - MARIA JOSÉ - cujo incentivo foi fundamental para que conseguíssemos elaborar este trabalho e que nos manteve com ânimo para as várias noites que passamos no NCE , quando da fase de testes.

. À nossa secretária - VERA FERRAZ - que teve a paciência e o capricho de datilografar nosso trabalho.

. Enfim, a todos que direta ou indiretamente nos ajudaram na SMT e que porventura não tenham vindo à nossa lembrança neste momento.

R E S U M O

O *SISTEMA MULTI-TERMINAL (SMT)* é um método de acesso para terminais remotos ou locais, desenvolvido sob o *Sistema Operacional em Disco (SOCO)* do *TERMINAL INTELIGENTE (TI)* do NÚCLEO DE COMPUTAÇÃO ELETRÔNICA da UNIVERSIDADE FEDERAL DO RIO DE JANEIRO (NCE/UFRJ).

De um modo geral é mostrada a filosofia do *SISTEMA* e o seu funcionamento interno, com a especificação detalhada de cada um de seus módulos: *INICIALIZADOR, DISPATCHER, INTERPRETADOR DE COMANDOS, EXECUTOR DE E/S, PROCESSADOR DE INTERRUPÇÕES* e *FINALIZADOR.*

Permite ao programador, ao desenvolver a aplicação, ver apenas um terminal e, em tempo de execução, serem definidos os terminais que usarão concorrentemente a aplicação.

Todas as operações de entrada e saída dos terminais e respectivos controles de áreas de trabalho são realizadas pelo *SMT* e são transparentes tanto ao usuário (operador do terminal) como ao programador.

O *SISTEMA* foi desenvolvido para aplicações com baixa taxa de utilização; o número de terminais associados a uma determinada aplicação em tempo de execução deverá ser de, no máximo, dez.

Aplicações típicas são: Controle de Almoxarifado, Marcação de Consultas Hospitalares, Controle de Leitos em Hospital, e outras de nível de utilização semelhante.

A B S T R A C T

The *MULTI-TERMINAL SYSTEM (SMT)* is a terminal acess method for local or remote use, developed for operating under the *Disk Operational System (SOCO)* of the *INTELLIGENT TERMINAL (TI)* built in the NUCLEO DE COMPUTAÇÃO ELETRÔNICA of the UNIVERSIDADE FEDERAL DO RIO DE JANEIRO (NCE/UFRJ).

The *SYSTEM* philosophy and its internal logic is shown in a global way and it is given a detailed specification of each module which compounds the *SYSTEM: INICIATOR, DISPATCHER, COMMAND INTERPRETER, I/O CONTROL, INTERRUPT HANDLER* and *TERMINATOR.*

It permits, to the application programmer, while developing the application, to see only *one terminal* and the network definition will take place only at the execution time.

SMT performs all of I/O operations with the terminals and respective work area controls and do them in a transparent way to the user (terminal operator) and the application programmer.

Aplications with low ratio of utilization are typical *SMT* users. The network maximum number of lines is represented by one digit, i. e., no more than ten terminals are permitted.

In the practice, typical applications are: On-line Inventory Control, Health Care Clinical Appointment Scheduling, Patient Monitoring, and other applications with the same level of on-line utilization ratio.

Í N D I C E

I.	INTRODUÇÃO	1
II.	DEFINIÇÃO DO SISTEMA	5
II.1.	COMENTÁRIOS GERAIS	6
II.2.	CONCEITO DE PROGRAMA VIRTUAL	7
II.3.	CLASSES DE DADOS DO PROGRAMA	8
II.4.	USO DO TERMINAL	9
II.5.	PROGRAMAÇÃO DO TERMINAL	10
	II.5.1. DETALHES DO ARQUIVO LÓGICO	10
	II.5.2. ESPECIFICAÇÃO DE TERMINAIS	11
	II.5.3. TÉRMINO DO PROGRAMA VIRTUAL	12
	II.5.4. USO DE OUTROS ARQUIVOS	12
	II.5.5. USO DE COMANDOS ESPECIAIS DA LINGUAGEM PLTI	13
III.	DESCRIÇÃO DO SISTEMA	14
III.1.	COMENTÁRIOS GERAIS	15
III.2.	TABELAS E ÁREAS MANIPULADAS PELO SMT	16
	III.2.1. ÁREAS E TABELAS DO SOCO UTILIZADAS PELO SMT	16
	III.2.1.1. Tabela Resolvida de Arquivos Lógicos	16
	III.2.1.2. Tabela de Arquivo Físico	16
	III.2.1.3. Núcleo Residente	19

III.2.2. ÁREAS E TABELAS CRIADAS PELO SMT, NA MEMÓRIA 21
III.2.2.1. Tabela de Usuários 21
III.2.2.2. Ponteiro para o Elemento de TABUS 21
III.2.2.3. Endereço e Tamanho da Área de Dados do Programa 22
III.2.2.4. Savearea da Rotina EXIT 22
III.2.2.5. Ponteiro para Área Reserva em Disco 22
III.2.2.6. Ponteiro para a Próxima Área de Disco Disponível 23
III.2.2.7. Tamanho do Registro Lógico do Arquivo que usa o SMT 23
III.2.2.8. Savearea Contendo PC Reserva 23
III.2.3. ÁREAS E TABELAS CRIADAS PELO SMT, EM DISCO 23
III.2.3.1. Área de Dados de Programas Virtuais 23

III.3. DIAGRAMA CONCEITUAL DO SMT 25

III.4. ESTADOS DE UM PROGRAMA VIRTUAL 27

III.5. DESCRIÇÃO DA LÓGICA DO SMT 30
III.5.1. INICIALIZAÇÃO DO SMT E PROGRAMAS VIRTUAIS 30
III.5.2. LÓGICA DO PROCESSAMENTO DE UM PROGRAMA VIRTUAL 32

III.6. ESPECIFICAÇÃO DOS MÓDULOS DO SMT 36
III.6.1. INICIALIZADOR 36
III.6.2. EXECUTOR DE E/S 41
III.6.3. DISPATCHER 43
III.6.4. INTERPRETADOR DE COMANDOS 47
III.6.4.1. Funções do INTERPRETADOR para o Estado INATIVO FE 47
III.6.4.2. Funções do INTERPRETADOR para o Estado ATIVO FE 49
III.6.4.3. Funções do INTERPRETADOR para o Estado INATIVO TESTE FE 50

III.6.5. FINALIZADOR 50

III.6.6. PROCESSADOR DE INTERRUPÇÃO 51

IV. IMPLEMENTAÇÃO 56

IV.1. CONFIGURAÇÃO DO EQUIPAMENTO 57

IV.1.1. EQUIPAMENTO PROPRIAMENTE DITO 57

IV.1.2. SOFTWARE DISPONÍVEL 58

IV.1.2.1. Simulação do Software do TI no B-6700 (Cross-Software) 58

IV.1.2.2. Sistema Operacional do TI (SOCO) 58

IV.2. PROBLEMAS ENCONTRADOS 59

IV.2.1. DESCRIÇÃO DOS PROBLEMAS E SOLUÇÕES ENCONTRADAS 59

IV.2.1.1. Comunicação entre Programa Virtual e o SMT 59

IV.2.1.2. Teste do SMT no TI 60

IV.2.1.3. Manuseio de Teclado de Terminais-Vídeo 61

IV.2.1.4. Obtenção do Endereço e Tamanho do Arquivo 62

IV.2.1.5. Como Testar o SMT no TI, sob o Controle do SOCO 62

IV.2.1.6. Como Fazer Referência ao Interpretador PLTI do Programa do Usuário 63

V. CONCLUSÕES 65

V.1. RESULTADOS OBTIDOS 66

V.1.1. TEMPO DE RESPOSTA VERSUS SOBREPOSIÇÃO DE CARACTERES NA RECEPÇÃO (OVERRUN) 66

V.1.2. GASTO DE MEMÓRIA PRINCIPAL 71

V.1.3. USO DE DISCO DE TRABALHO PELO SMT 73

V.1.4. USO DE TERMINAIS LOCAIS x REMOTOS 73

V.2. EXTENSÕES SUGERIDAS/POSSÍVEIS 75

V.2.1. IMPLEMENTAÇÃO DE NOVOS COMANDOS 75

V.2.2. IMPLEMENTAÇÃO DE NOVOS TERMINAIS DE E/S E/OU MUDANÇAS DE PROTOCOLO 76

V.2.3. IMPLEMENTAÇÃO DE NOVAS FUNÇÕES DE E/S PARA OS TERMINAIS 77

V.3. ANÁLISE FINAL DO TRABALHO 78

VI. BIBLIOGRAFIA 80

VII. APÊNDICES 82

VII.1. APÊNDICE 1 - ALTERAÇÕES PROVISÓRIAS DO SMT PARA FUNCIONAR COM NÚCLEO NÃO RESIDENTE 83

VII.2. APÊNDICE 2 - CÁLCULO DOS TEMPOS APROXIMADOS DE CADA FUNÇÃO DO PROCESS.INTERRUPÇÕES 93

VII.3. APÊNDICE 3 - TABELA DE TEMPOS DAS MICRO-INSTRU - ÇÕES PLTI 104

VII.4. APÊNDICE 4 - PARTES DO SOCO ALTERADAS/UTILIZADAS PELO SMT 107

I. INTRODUÇÃO

O *TERMINAL INTELIGENTE (TI)* é um micro-computador, baseado no micro-processador INTEL-8008, desenvolvido e construído pelo NÚCLEO DE COMPUTAÇÃO ELETRÔNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO (NCE/UFRJ).

Possui 16 *Kbytes* de memória e a ele pode-se conectar qualquer dispositivo de entrada e/ou saída, através de interfaces convenientes. Atualmente, estão disponíveis para instalação os periféricos: Leitora de Cartões Perfurados, Impressora, Unidade de Fita Cassete, Unidade de Disco, Console (composto de Teclado e Unidade de Vídeo), e *Links* de Comunicação.

As aplicações típicas do *TI* são aquelas desenvolvidas nos pequenos computadores. Aplicações comerciais podem ser desenvolvidas, estando limitadas praticamente pelo tamanho da memória, problema facilmente superável pelo fato de todo o *software* ser desenvolvido na linguagem *PLTI*, que é compatível com qualquer mudança do micro-processador.

Existe ainda a possibilidade de ligação a computador de maior porte, facilitando aplicações que requeiram maiores recursos ou um pré-processamento.

A comercialização do *TI* exige que seja possível estender sua capacidade de atendimento a usuários em locais diversos de sua instalação.

Uma outra necessidade é o compartilhamento, por diversos usuários, das aplicações em execução no *TI*, onde cada um deve ter a sensação de ser o único a estar sendo atendido.

O *SISTEMA MULTI-TERMINAL (SMT)*, a seguir descrito, viabiliza a ligação do *TI* com terminais tipo vídeo ou impressores para atendimento da necessidade de uso remoto, bem como da utilização de uma mesma aplicação por vários usuários.

O *SMT* é o intermediário entre um terminal vídeo ou impressor, acoplado ao *TI*, e o programa que o atende.

A arquitetura do *SMT* está dimensionada para suporte a

um número de terminais que não exceda a um (1) dígito, ou seja, até dez terminais.

Para que um programa use o *SMT* como monitor de suas operações de E/S com terminais, basta que seja fornecido o parâmetro adequado, especificando qual arquivo lógico está associado a qual arquivo físico do tipo terminal, no comando para execução do programa.

O meio utilizado para se passar ao *SMT* quantos e quais são os terminais em uso pelo programa é a concatenação de vários arquivos físicos a um único arquivo lógico, que é uma facilidade permitida pelo *Sistema Operacional do TI (SOCO)*. Com isso, fica extremamente fácil a inclusão ou exclusão de usuários (ou seja, terminais), efetivada pela inclusão ou exclusão de arquivos físicos da referida concatenação.

A grande vantagem é que o programador apenas precisará se preocupar em trabalhar com um único arquivo lógico, cabendo ao *SMT* administrar toda a complexidade de associar vários terminais a tal arquivo lógico em tempo de execução.

Na fase de desenvolvimento dos programas que compõem um sistema é desejável testar e executar tais programas independentemente do dispositivo de E/S dos dados, isto é, o programa que receberá/enviará dados deverá ser o máximo possível independente do dispositivo onde estarão os dados.

O *SMT* é ativado pelo *Sistema Operacional do TI (SOCO)* em função da especificação de parâmetros indicando uso de terminais no comando de execução; a independência do programa com relação a seus dispositivos de E/S é mantida, limitada evidentemente pelo uso de características específicas de cada dispositivo.

Exemplificando, se um programa pretende utilizar a característica conversacional de um terminal, logicamente não haverá sentido nem possibilidade de, em tempo de operação, ter o dispositivo de E/S *terminal* trocado por outro qualquer. Haverá sempre um compromisso entre a flexibilidade para troca de dispositivos de E/S do programa e a eficiência com que os mesmos serão utilizados.

O principal objetivo do *SMT* é a gerência da comunicação

programas-terminais, devendo o programador raciocinar como se houvesse um único terminal ligado ao programa, despreocupando-se quanto ao número de terminais conectados. O número de usuários é limitado apenas pelas características de configuração escolhidas para a instalação do *TI*, ou seja, pelo número máximo de dispositivos previstos para serem ligados ao *TI* e pela memória total instalada (a utilização de memória pelo programa crescerá com o número de terminais a ele vinculados).

II. DEFINIÇÃO DO SISTEMA

II.1. COMENTÁRIOS GERAIS

O *SISTEMA MULTI-TERMINAL (SMT)* é um método de acesso a terminais ligados ao *TI* via *links* de comunicação. Ele implementa o conceito de *programa virtual*, que é o fato de um programa ser acessado por vários terminais, tudo se passando como se cada terminal tivesse um programa que exclusivamente o atendesse.

As aplicações típicas para o *SMT* são aquelas de consultas a arquivos em acesso aleatório ou indexado, atualização *on-line* de arquivos indexados ou aleatórios, e aquelas que tenham as características de teleprocessamento.

O *SMT* é considerado pelo *Sistema Operacional do TI (SOCO)* como uma rotina de E/S, sendo portanto carregado junto com o programa de aplicação quando for especificado o uso de terminais para o mesmo.

A memória inicial ocupada é da ordem daquela gasta pelas rotinas de E/S convencionais, porém crescendo de acordo com o número de terminais que usarão o programa.

Como todo o *software* básico do *TI*, o *SMT* é escrito na linguagem *PLTI*, possibilitando sua total compatibilidade com qualquer micro-processador que, no futuro, venha a ser usado pelo *TI*.

O programa de aplicação deverá ser escrito na linguagem *PLTI*, comunicando-se com o *SMT* via comandos *READ* ou *WRITE* solicitados para um arquivo lógico, ao qual foram associados, em tempo de carga do programa, um ou mais terminais.

II.2. CONCEITO DE PROGRAMA VIRTUAL

Na memória somente existirá uma cópia do programa - o *programa real* - que atenderá a todos os terminais que o estiverem usando em modo *virtual*.

Cada terminal terá acesso a um *programa virtual*, que é o *programa real* com os dados correspondentes ao terminal que o estiver usando. O acesso ao *programa virtual* por um terminal é administrado pelo *SMT* de modo a que todos os terminais tenham a mesma chance de uso do *programa virtual* correspondente.

Quando o nível de atividade dos terminais não for o mesmo para todos os terminais, ou seja, existirem terminais *mais ativos* e terminais *menos ativos*, a chance de uso que os *menos ativos* dispensarem será dada àqueles *mais ativos*, isto devido ao algoritmo de administração mencionado no parágrafo anterior.

II.3. CLASSES DE DADOS DO PROGRAMA

Os dados usados pelo programa são classificados em dois tipos: *globais* e *locais*.

Os dados *globais* são aqueles comuns a todos os programas virtuais, isto é, não se alteram com o programa virtual que estiver com o controle. Normalmente deverão ser declarados nos blocos mais internos do programa real, e não serão atualizados pelo *SMT*.

Os dados *locais* são aqueles específicos a um determinado programa virtual, alterando-se em função do programa virtual que estiver sendo processado. Deverão ser declarados obrigatoriamente no bloco mais externo do programa real, porque deverão ser atualizados para cada programa virtual que for processado.

Para o programa real os dados *locais* serão os declarados no bloco mais externo, e os dados *globais* serão os declarados nos blocos mais internos do programa.

II.4. USO DO TERMINAL

Os terminais associados ao programa real poderão estar em dois estados: *ativos* e *inativos*.

Os terminais *ativos* são aqueles que estejam usando os respectivos programas virtuais.

Os terminais *inativos* são aqueles que, embora estejam fazendo parte da lista de terminais associados ao programa real, não estejam usando nenhum programa virtual.

Para a ativação e desativação de terminais são usados comandos específicos do *SMT*: *$ATIVE* e *$FIM*.

O comando *$ATIVE* deve ser dado quando um terminal *inativo* desejar se tornar *ativo*, isto é, associar-se a um programa virtual e iniciar processamento.

O comando *$FIM* deve ser dado quando um terminal desejar encerrar suas atividades, isto é, fechar o arquivo lógico a que está associado.

Inicialmente, todos os terminais estão em estado *inativo*, devendo ser dado o comando *$ATIVE* em cada terminal que desejar se tornar *ativo*.

Um terminal que já tenha sido *desativado*, isto é, dado o comando *$FIM*, poderá ser novamente *ativado*, bastando para isso que se dê o comando *$ATIVE*.

Além dos comandos acima descritos existem mais dois comandos que implementam uma função de teste de *hardware* dos terminais (teclado). São eles: *$TESTE* e *$FIMTESTE*.

Para se iniciar o teste de teclado do terminal o mesmo deverá estar em estado *inativo*, devendo então ser dado o comando *$TESTE*. Este procedimento só se encerrará quando o comando *$FIMTESTE* for dado.

Todos os textos (sequência de caracteres seguidos de ⟨cr⟩) serão retransmitidos para o terminal para verificação.

II.5. PROGRAMAÇÃO DO TERMINAL

II.5.1. DETALHES DO ARQUIVO LÓGICO

O programa somente deverá usar um único arquivo lógico associado a terminais; em caso de existir mais de um, os resultados são imprevisíveis.

Para o arquivo lógico do programa real associado a terminais duas funções serão atendidas pelo *SMT*: *READ* e *WRITE*.

Para a função *WRITE* será assumido o uso de caracteres de controle indicativos de posicionamento, na tela, do texto a ser escrito.

Não é permitido o uso de operações de E/S com o arquivo lógico em mais de uma *procedure* do programa, isto é, não é possível serem solicitadas operações de E/S num mesmo programa no bloco mais externo (principal) e em blocos internos do programa (*procedures*). Isto é decorrente do fato de existirem parâmetros de controle de *procedure* ativa para o *Interpretador PLTI* na *stack* do *TI*, e, ao programa virtual perder o controle numa operação de E/S, nada indica que ao ganhá-lo de novo a *stack* não tenha sido alterada por outros programas virtuais posicionados em outras partes do programa real. Se todas as operações de E/S estiverem no mesmo bloco, a integridade da *stack* é mantida e consequentemente o programa virtual será executado sem problemas.

Os caracteres de controle possíveis são os do protocolo *TTY* (*Teletype*), característica dos terminais que podem ser conectados ao *TI*.

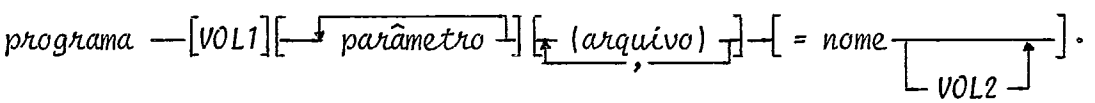
Poderão existir num mesmo texto a ser escrito no terminal várias ocorrências dos caracteres de controle - *FORM-FEED* (*#0C*), *LINE-FEED* (*#0A*), e *CARRIAGE-RETURN* (*#0D*). A única restrição imposta é que não poderão existir num mesmo texto dois *CARRIAGE-RETURN* adjacentes, por razões de controle interno do *SMT*.

O maior tamanho de registro possível é de 81 *bytes*, devido ao uso do caractere de controle para linhas a serem escritas no terminal, e de 80 *bytes* para linhas a serem lidas do terminal pelo programa real. Como consequência, o tamanho de uma linha lida do arquivo lógico será igual ao tamanho do registro lógico, especificado para o arquivo, menos 1 (- 1).

No caso de ser declarado pelo programa real um tamanho de registro maior que 81 *bytes* para o arquivo lógico a que estarão associados terminais, o *SMT* cancelará o programa com mensagem correspondente no painel do *TI*.

II.5.2. ESPECIFICAÇÃO DE TERMINAIS

A especificação dos terminais que estarão associados ao programa real deverá ser feita pelo *comando de execução do programa*, que tem o seguinte formato:



onde, *VOL1* - volume do arquivo que contém o programa;

parâmetro - parâmetros que são passados ao programa;

arquivo - *interno* - = *externo* [*VOL3*] : *periférico*

interno - nome do arquivo definido no programa-fonte;

externo - nome de arquivo existente em determinado disco, criado por utilitário específico, que descreve um terminal em termos de endereço físico, formato de transmissão e velocidade de transmissão.

VOL3 - nome do volume onde está externo; se omitido, é assumido disco do sistema residente.

periférico - mneumônico que indica tipo do periférico; especificar *SMT*.

nome - nome da imagem do programa, guardado pelo Carregador de Programa (CPROG) no disco de volume *VOL2*, correspondente à concatenação feita no comando de execução. Evita especificação de todos os dados a cada vez que se executar o programa.

II.5.3. TÉRMINO DO PROGRAMA VIRTUAL

Quando um terminal se desativar, isto é, der o comando *$FIM*, será passado ao programa virtual correspondente à condição de *EOF*, que poderá ser aceita ou não pelo programa virtual em função do que estiver programado, pois o terminal só será efetivamente desativado quando o programa virtual der *EXIT*.

Mesmo depois de ocorrida a condição de *EOF* para o arquivo lógico, o programa poderá executar operações de E/S (*READ* e/ou *WRITE*) com o mesmo, cabendo ao programador a responsabilidade de prever que em operações de entrada (*READ*) possa ocorrer a condição de *EOF*.

O término de todos os programas virtuais implicará no término do programa real, isto é, quando todos os programas virtuais derem *EXIT*, mais nenhum estará em processamento, e o programa real terminará.

II.5.4. USO DE OUTROS ARQUIVOS

O *SMT* somente administra a atualização dos dados locais a cada programa virtual, nada fazendo em relação a dados globais. Por esta razão, caso outros arquivos lógicos (em fita , disco, impressora, etc.) sejam utilizados pelo programa real, serão comuns a todos os programas virtuais.

Exceção se fará a arquivos lógicos de acesso direto, onde existe a possibilidade de serem particulares a cada programa virtual desde que sejam tomados os cuidados necessários.

Após a solicitação de E/S por um programa virtual, como o controle pode ser dado a outro programa virtual pelo *SMT*, todos os dados referentes ao arquivo de acesso direto (*SEEK*,etc.) devem ser reposicionados pelo programa virtual.

II.5.5. USO DE COMANDOS ESPECIAIS DA LINGUAGEM PLTI

Os comandos *SET* e *RESET*, que servem para identificar no painel qual trecho do programa está sendo executado, não devem ser usados em trechos que contenham operações de E/S com o *SMT*. Isto porque o programa virtual perde o controle a cada operação de E/S, sendo este controle dado a outro programa virtual, destruindo a sequência numérica dos conjuntos *SET/RESET*.

O comando *TRACE*, admitido pela linguagem *PLTI*, pode ser usado para teste do programa real, desde que somente um (1) programa virtual seja ativado. Caso seja ativado mais de um (1) , a sequência de instruções mostrada no vídeo será uma mistura de sequências de instruções dos vários programas virtuais.

III. DESCRIÇÃO DO SISTEMA

III.1. COMENTÁRIOS GERAIS

Como foi dito no capítulo anterior, o *SISTEMA MULTI-TERMINAL (SMT)* é um administrador (monitor) de programas virtuais, caracterizado um programa virtual quando o programa real estiver sendo executado para um determinado terminal.

Para a realização disto o *SMT* utiliza um conjunto de tabelas e áreas internas, bem como áreas mantidas pelo *Sistema Operacional do TI (SOCO)*, que servem como elementos de comunicação entre os diversos módulos, cada um com uma função específica, e que compõem o *SMT*.

O *SMT* está estruturado em módulos com funções específicas que criam e utilizam as tabelas e áreas anteriormente mencionadas.

Os módulos do *SMT* são os seguintes:

INICIALIZADOR - Responsável pela inicialização global do *SMT*.

PROCESSADOR DE INTERRUPÇÕES - Responsável por identificar e processar as interrupções causadas pela presença de um caractere na interface com os *links* ou pelo término da transmissão de um caractere.

EXECUTOR DE E/S - Responsável pela execução das operações de entrada e saída solicitadas pelo programa.

DISPATCHER - Responsável pela ativação dos diversos programas virtuais, de acordo com o estado em que estiverem (salvando e atualizando as áreas de dados respectivas).

INTERPRETADOR DE COMANDOS - Responsável pela interpretação de todos os textos recebidos dos terminais.

FINALIZADOR - Responsável pela finalização de cada programa virtual, quando for recebido um *EXIT* do mesmo, bem como finalização total do *SMT* quando o último programa virtual se desativar.

III.2. TABELAS E ÁREAS MANIPULADAS PELO SMT

Cada uma das áreas e tabelas, tanto do *SMT* como do *SOCO*, são criadas e alteradas pelos módulos enumerados no item anterior (III.1.), da forma descrita nos sub-itens seguintes.

III.2.1. ÁREAS E TABELAS DO SOCO UTILIZADAS PELO SMT

III.2.1.1. Tabela Resolvida de Arquivos Lógicos (TRAL)

É criada pelo *Editor de Referências Externas (REFEX)*, quando gera o programa executável (pelo *Interpretador PLTI*) a partir da saída objeto do *Compilador PLTI*. É utilizada pelo *INICIALIZADOR* para verificação e obtenção do tamanho do registro lógico.

Cada elemento da *TRAL* corresponde a um arquivo lógico do programa e contém (FIGURA III.1.): (*a*) instrução que carrega o seu identificador; (*b*) um desvio para a *TAF* ativa associada; (*c*) o tamanho do registro do arquivo.

III.2.1.2. Tabela de Arquivo Físico (TAF)

É criada pelo *CPROG* após a carga do programa, com base nos dados fornecidos pelo comando de execução do mesmo. É usada pelo *PROCESSADOR DE INTERRUPÇÕES* para receber/transmitir o texto, recebido/transmitido caractere a caractere; pelo *INTERPRETADOR DE COMANDOS* quando da movimentação de textos para a área de leitura do programa; e pelo *EXECUTOR DE E/S* que, quando do início de operação de saída, move o texto para a *TAF*, deixando a cargo do *PROCESSADOR DE INTERRUPÇÕES* o término da mesma.

A *TAF* contém:

a. o tipo da *TAF* (tipo do Arquivo Físico);

b. um desvio para a rotina de E/S correspondente ao Arquivo Físico associado;

c. um ponteiro para a próxima *TAF* concatenada;

d. informações adicionais que dependem do tipo de Arquivo Físico.

No caso de terminais, as informações adicionais são (FIGURA III.1.):

a. endereço físico do terminal (*link*);

b. formato da transmissão;

c. velocidade de transmissão;

d. ponteiro para o último caractere recebido/transmitido (1 *byte*);

e. área de recepção/transmissão (82 *bytes*).

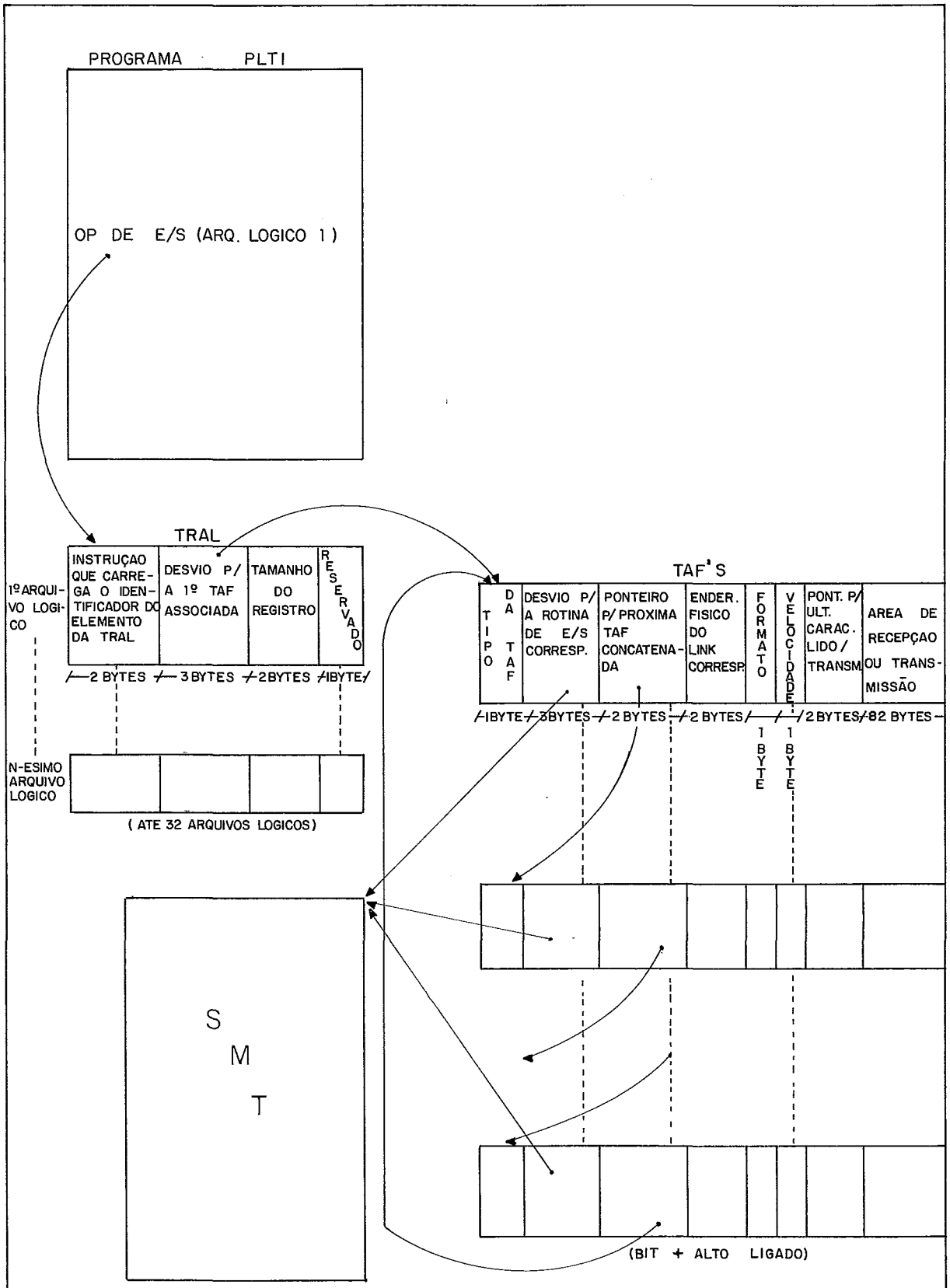


FIGURA III. 1: FORMATO DA TRAL, TAF E SUAS LIGAÇÕES COM O PROGRAMA

III.2.1.3. Núcleo Residente (NRES)

O *NRES* é uma área fixa na memória contendo uma série de informações de controle para o *SOCO*, bem como algumas rotinas auxiliares muito usadas (FIGURA III.2.).

O *SMT* usará, do *NRES*, algumas informações já existentes e criará outras para seu próprio controle.

Informações já existentes:

a. Desvio para o *entry-point* do programa (Pos 35 a 37 Hex) usada para cálculo do endereço e tamanho da área de dados do programa pelo *INICIALIZADOR*.

b. *EXIT* (Pos 38 a 55 Hex) alterada pelo *INICIALIZADOR*, para ser um desvio para o *SMT* processar o fim do programa virtual, quando este receber *$FIM* do terminal. Restaurada pelo *FINALIZADOR* ao término do *SMT*.

c. Endereços da *TRAL* e da *TAF* em uso (Pos 70 a 73 Hex) atualizada pela rotina auxiliar *TRATAF* com os endereços da *TRAL* e da *TAF* em uso no momento.

d. Endereço da *savearea* do *Interpretador PLTI* (Pos 74 e 75 Hex) criado pelo núcleo do *Interpretador PLTI*, apontando para a *savearea* cujo formato é:

- *STATUS* - resultado (retorno) da última operação de E/S realizada (micro-rotina *INTES*);
- registradores *B*, *C*, *D*, *E*, *A*, *H* e *L*, nesta ordem;
- endereço de início e de fim para opção *DUMP* de memória quando em uso o *TRACE PLTI*;
- *Program Counter* do *Interpretador PLTI*, salvo quando o *TRACE PLTI* estiver em uso;
- *Program Counter* da última operação de E/S executada pela micro-rotina *INTES* (apontando para a posição seguinte ao código da operação de E/S).

ÁREA RESERVADA PARA USO FUTURO	80
Ponteiro para o núcleo do *Interpretador PLTI*	7B
Indicador de *overflow* (qualquer tipo)	79
Ponteiro para *stack* de *SET/RESET* do *Compilador PLTI*	78
Savearea dos registradores: *STATUS B, C, D, E, A, H* e *L*	76
Endereço da *TAF* em uso (última referenciada)	74
Endereço da *TRAL* em uso (última referenciada)	72
Número de linhas da tela do vídeo (console)	70
Número de caracteres da linha do vídeo (console)	6F
Número de linhas úteis escritas no vídeo (console)	6E
LAMPA	6D
Entry-point do *CPROG*	5E
Endereço em disco onde está o *CPROG*	5C
Número de *bytes* do *CPROG*	5A
Endereço de carga do *CPROG*	58
EXIT	56
Desvio para o *entry-point* do programa (*JMP*)	38
Número de registros lidos pelo *BATCH* (*CPROG*)	35
LUZES (Desvio para *LAMPA*)	33
Ponteiro para o último *byte* da área livre	30
Endereço do diretório do disco	2E
INCHL	2C
Data do dia: *DDMMAA* (*3 bytes*)	28
INDEX	25
Ponteiro para os parâmetros	20
Indicador de parâmetro: ≠ 0 tem parâmetro; = 0 não tem	1E
LDEND	1D
STEND	18
DECHL	10
Tamanho da memória (em blocos de *256 bytes*)	08
INDDX	07
	00

FIGURA III.2. *NÚCLEO RESIDENTE DO SOCO (NRES)*

III.2.2. ÁREAS E TABELAS CRIADAS PELO SMT, NA MEMÓRIA

III.2.2.1. Tabela de Usuários (TABUS)

A *TABUS* é criada pelo módulo *INICIALIZADOR* quando da ocorrência da primeira operação de E/S, atualizada pelos módulos *EXECUTOR DE E/S*, *INTERPRETADOR DE COMANDOS*, *PROCESSADOR DE INTERRUPÇÕES* e *FINALIZADOR*, e utilizada pelo *DISPATCHER* no ciclo de busca de programas virtuais prontos para serem processados.

A *TABUS* contém:

a. endereço físico do terminal (*link*), usado pelo *PROCESSADOR DE INTERRUPÇÕES* para localizar área de leitura/transmissão correspondente ao terminal que causou a interrupção;

b. *savearea* contendo *Program Counter* do *Interpretador PLTI (PC)*;

c. endereço da área em disco correspondente à área de dados associada ao programa virtual quando de sua primeira ativação;

d. retorno da operação de E/S (última realizada);

e. ponteiro para a *TAF* que contém área de recepção/transmissão correspondente ao programa virtual;

f. endereço da área de E/S para a qual deve ser movido o texto quando da realização da operação de leitura do terminal;

g. estado do programa virtual, atualizado por vários módulos do *SMT* de acordo com autômato e matriz de estados correspondentes (FIGURA III.4.).

III.2.2.2. Ponteiro para o Elemento de TABUS

O *Ponteiro para o Elemento de TABUS* contém o endereço

do elemento correspondente ao último programa virtual ativo. É criado e atualizado pelo *DISPATCHER* quando da ativação de programas.

III.2.2.3. Endereço e Tamanho da Área de Dados do Programa

O *Endereço* e o *Tamanho da Área de Dados do Programa* são criados pelo *INICIALIZADOR* e usados pelo *DISPATCHER* para salvar e atualizar a área de dados de cada programa virtual.

O *Tamanho da Área de Dados* é guardado de suas formas: o tamanho exato e o tamanho arredondado para um número inteiro de setores (512 *bytes*) de disco, este último para possibilitar a gravação em disco das áreas de cada programa virtual.

III.2.2.4. Savearea da Rotina EXIT

A *Savearea da Rotina EXIT* contém a parte destruída pelo *SMT* para criar um desvio para o mesmo processar qualquer *EXIT* dado pelo programa. É criada pelo *INICIALIZADOR* quando este altera a *EXIT*, e utilizada pelo *FINALIZADOR* para reconstruir a *EXIT* ao término do programa real.

III.2.2.5. Ponteiro para Área Reserva em Disco

O *Ponteiro para Área Reserva em Disco* tem o endereço da área de disco que contém a imagem de área de dados do programa quando da ocorrência da primeira operação de E/S. É criado pelo *INICIALIZADOR* e utilizado pelo *INTERPRETADOR DE COMANDOS* para inicializar a área de dados de terminal que transmita o comando *$ATIVE*.

III.2.2.6. Ponteiro para a Próxima Área de Disco Disponível

O *Ponteiro para a Próxima Área de Disco Disponível* contém o endereço de área de disco a ser utilizado pelo *INTERPRETADOR DE COMANDOS* para guardar área de dados do próximo terminal que pela primeira vez der o comando *$ATIVE*.

III.2.2.7. Tamanho do Registro Lógico do Arquivo que usa o SMT

O *Tamanho do Registro Lógico do Arquivo que usa o SMT* contém o tamanho do registro a ser escrito no terminal (incluído o caractere de controle) e o tamanho mais um (+ 1) do registro a ser lido do terminal. É criado pelo *INICIALIZADOR* e utilizado pelo *EXECUTOR DE E/S* e pelo *INTERPRETADOR DE COMANDOS* na realização, respectivamente, das operações de saída e de entrada.

III.2.2.8. Savearea Contendo PC Reserva

O *Savearea contendo o PC Reserva*, que corresponde àquele da primeira operação de E/S, é criada pelo *INICIALIZADOR* e utilizada pelo *INTERPRETADOR DE COMANDOS* para criação do PC no elemento de *TABUS* quando um programa virtual for ativado via comando *$ATIVE*.

III.2.3. ÁREAS E TABELAS CRIADAS PELO SMT, EM DISCO

III.2.3.1. Área de Dados de Programas Virtuais

A *Área de Dados*, em disco, uma (1) para cada programa virtual ativo, é criada pelo *INTERPRETADOR DE COMANDOS* para cada terminal que transmitir *$ATIVE* (tornar-se ativo). É acessa-

da pelo *DISPATCHER* quando passa o controle de um programa virtual para outro, para salvar e atualizar, respectivamente, a área de dados.

Além dessas *Áreas de Dados* acima, existe a *Área de Dados Reserva*, que serve para ativação de programas virtuais, e é a primeira área do disco.

III.3. DIAGRAMA CONCEITUAL DO SMT

Para melhor compreensão da descrição lógica de cada módulo do *SMT*, a seguir mostramos o diagrama conceitual do *SMT* (FIGURA III.3.), que procura mostrar graficamente como se processa o fluxo de comunicações entre o *Interpretador PLTI*, o programa, e cada módulo do *SMT*.

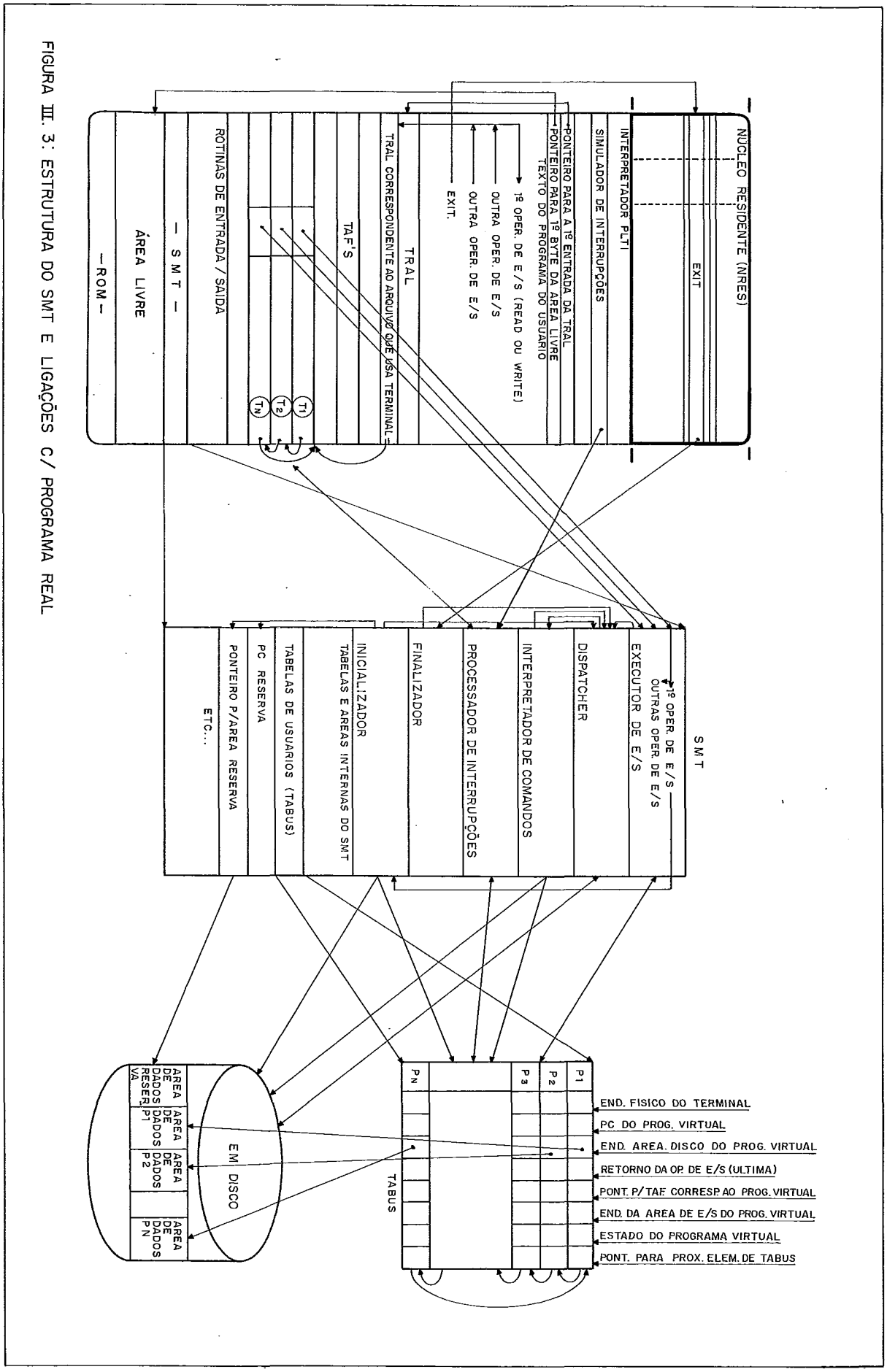


FIGURA III. 3: ESTRUTURA DO SMT E LIGAÇÕES C/ PROGRAMA REAL

III.4. ESTADOS DE UM PROGRAMA VIRTUAL

Um programa virtual pode estar em onze (11) estados diferentes, conforme a sequência das operações de E/S ocorridas:

INATIVO ENTRADA PENDENTE (INATIVO EP) - Programa virtual não ativado, isto é, não foi recebido ainda o comando *$ATIVE*.

INATIVO FIM DE ENTRADA (INATIVO FE) - Programa virtual inativo, após receber uma mensagem ainda não analisada.

ATIVO OK - Programa virtual pronto para ser executado, a partir de um determinado *PC*.

ATIVO SAÍDA PENDENTE (ATIVO SP) - Programa cuja última operação de E/S foi um *WRITE* que ainda não está completado, porém a operação já está sendo executada.

ATIVO ENTRADA PENDENTE (ATIVO EP) - Programa virtual que solicitou a operação de E/S *READ*, que ainda não foi completada.

ATIVO FIM DE ENTRADA (ATIVO FE) - Programa cuja última operação de E/S foi um *READ* que já está completado (*Cr* recebido).

INATIVO TESTE ENTRADA PENDENTE (INATIVO TESTE EP) - Terminal que pediu teste do *hardware* do seu teclado (digitou *$TESTE*).

INATIVO TESTE FIM DE ENTRADA (INATIVO TESTE FE) - Texto recebido de terminal que está testando seu teclado.

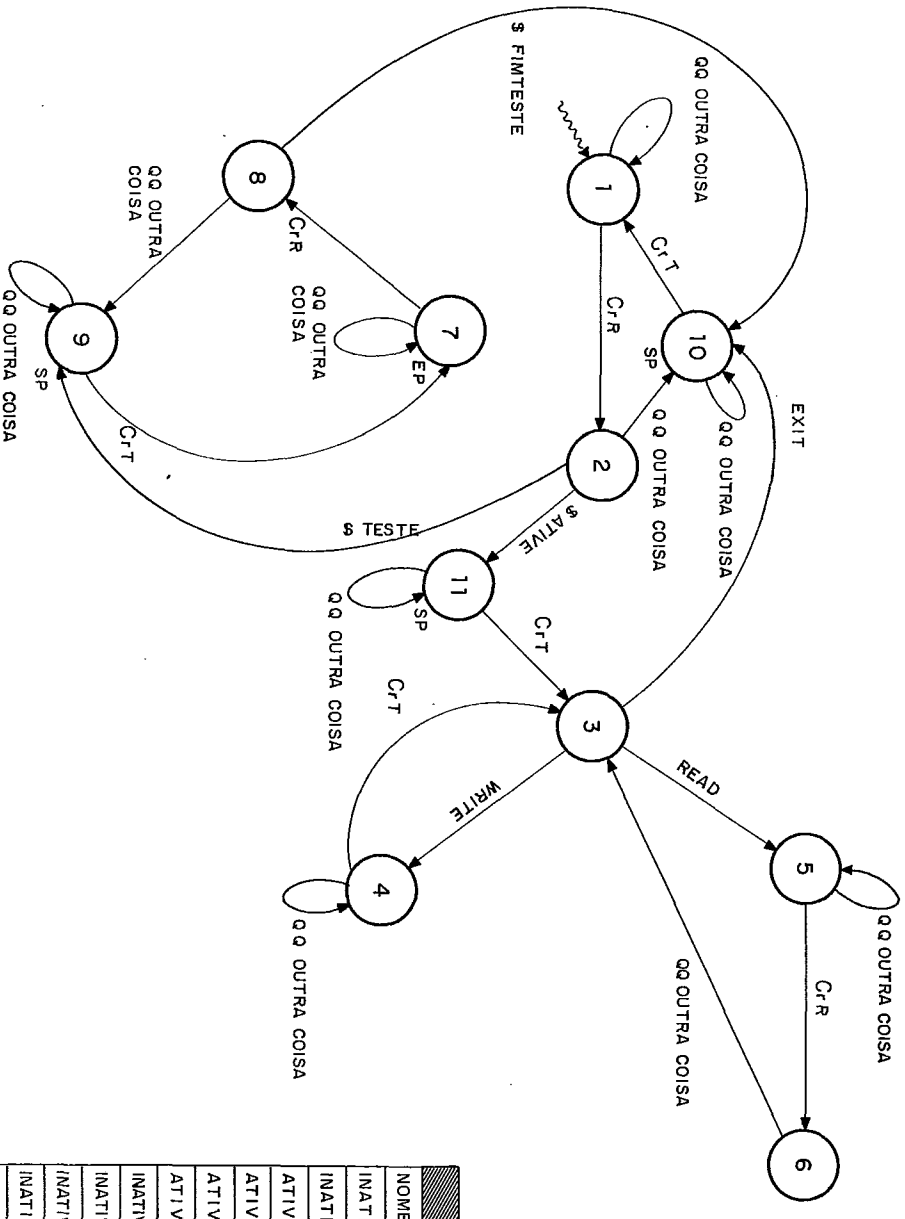
INATIVO TESTE SAÍDA PENDENTE (INATIVO TESTE SP) - Texto sendo enviado de volta a terminal que está testando seu teclado.

INATIVO SAÍDA PENDENTE (INATIVO SP) - Programa virtual inativo, com mensagem de erro a transmitir ainda não

completada.

INATIVO MENSAGEM PENDENTE (INATIVO MP) - Programa virtual inativo, com mensagem de ativação sendo enviada ao terminal que o ativou.

O autômato que descreve as mudanças de estado de cada programa virtual, em função da evolução do mesmo, isto é, ocorrência de *READS*, *WRITES*, *EXITS*, interrupções, etc., é apresentado na (FIGURA III.4.) a seguir.



		RESPONSAVEL PELA MUDANCA DE ESTADO											
		INTERPRETADOR COMANDOS				EXECUTOR E/S			PROCESS INTERRUP				
NOME DO ESTADO	Nº ESTADO	$ ATIVE	$ FIM	$ TESTE	$ FIMTESTE	READ	WRITE	EXIT	Cr RECEBIDO	Cr TRANSMITIDO	QQ OUTRA COISA		
INATIVO EP	1	—	—	—	—	—	—	—	2	—	1		
INATIVO FE	2	11	10	9	10	—	—	—	—	—	10		
ATIVO OK	3	—	—	—	—	5	4	1	—	—	—		
ATIVO SP	4	—	—	—	—	—	—	—	—	3	4		
ATIVO EP	5	—	—	—	—	—	—	—	6	—	5		
ATIVO FE	6	3	3	3	3	—	—	—	—	—	3		
INATIVO TESTE EP	7	—	—	—	—	—	—	—	8	—	7		
INATIVO TESTE FE	8	9	9	9	10	—	—	—	—	—	9		
INATIVO TESTE SP	9	—	—	—	—	—	—	—	—	7	9		
INATIVO SP	10	—	—	—	—	—	—	—	1	10			
INATIVO MP	11	—	—	—	—	—	—	—	3	11			

FIGURA III. 4: AUTOMATO E TABELA DE TRANSIÇÕES CORRESPONDENTES A MUDANÇAS DE ESTADO DO PROGRAMA VIRTUAL.

III.5. DESCRIÇÃO DA LÓGICA DO SMT

O diagrama conceitual do *SMT* (FIGURA III.3.) mostrou graficamente as interrelações entre o programa real e o *SMT* com os seus diversos módulos e tabelas. A seguir é descrito, passo a passo, o caminho que é seguido quando as operações de E/S vão sendo atendidas durante a execução dos vários programas virtuais.

III.5.1. INICIALIZAÇÃO DO SMT E PROGRAMAS VIRTUAIS

A primeira ocorrência é a carga pelo *CPROG*, na memória, do programa real e das rotinas de E/S solicitadas pelo comando de execução do programa, inclusive o *SMT*, que, como já dissemos, faz parte do grupo de rotinas de E/S.

O controle é então passado pelo *CPROG* ao programa real que se inicia, sendo então executadas todas as instruções iniciais, entendidas como sendo aquelas executadas antes da primeira operação de E/S com terminais. Tais instruções, comuns a todos os programas virtuais, serão executadas uma única vez e pelo programa real.

Ao ser solicitada a primeira operação de E/S, o controle é passado ao *SMT* via desvios sucessivos: do programa real para a *TRAL (CAL)*, da *TRAL* para a primeira *TAF* associada *(JMP)*, e da *TAF* para o módulo *EXECUTOR DE E/S* do *SMT (JMP)*.

O *EXECUTOR DE E/S* chamará a rotina auxiliar *TRATAF*, cuja função é atualizar os campos *TRAL* e *TAF* em uso no *NRES* e em seguida determinará que tal operação de E/S é a primeira a ocorrer, passando imediatamente o controle ao módulo *INICIALIZADOR*.

O *INICIALIZADOR* criará então os elementos de *TABUS* (parte) a partir dos dados de cada *TAF* associada à *TRAL*, mensagem de ativação de programa virtual para a área de E/S na *TAF*, movendo para o campo *Estado do Programa Virtual* de cada elemento

de *TABUS*, estado *INATIVO SP*, e preparando os interfaces para as operações de E/S com as linhas de comunicação.

Será guardado também o *PC* do *Interpretador PLTI* (referente à própria operação de E/S solicitada) numa área reserva do *SMT* para futuras ativações de programas virtuais por terminais que já tenham sido desativados (*$FIM*) e estejam se reativando (*$ATIVE*).

Em seguida, com base na estrutura padrão dos programas, o *INICIALIZADOR* calculará o endereço e tamanho da área de dados do programa real, que é local a cada programa virtual.

A área de dados será então guardada em área própria do *SMT* para uso em futuras ativações de programas virtuais, à semelhança do que é feito com o *PC* reserva.

O endereço da área de dados, bem como seu tamanho calculado, são guardados em área própria do *SMT* para uso pelo *DISPATCHER* quando na passagem de controle aos vários programas virtuais para salvar e/ou atualizar as respectivas áreas de dados.

É também guardado o tamanho da área de dados, arredondando para um número inteiro de setores do disco para possibilitar a gravação das áreas de dados locais, correspondentes aos programas virtuais, em disco.

É verificado pelo *INICIALIZADOR* se a área disponível em disco comporta o número de terminais associados ao programa real, através de comparação do tamanho da área disponível (existente no *NRES*) com a área total solicitada. Em caso negativo, o programa real é cancelado com mensagem correspondente.

Nesta posição o *SMT* está inicializado e o *INICIALIZADOR* executará suas tarefas finais: alterar o início da rotina *EXIT* (3 primeiros *bytes*) para serem um desvio incondicional para o *FINALIZADOR* e, por último, colocar um desvio no núcleo do *Interpretador PLTI* para o *PROCESSADOR DE INTERRUPÇÕES*, fazendo com que a partir deste momento sejam simuladas as interrupções.

O *Interpretador PLTI* então se encarregará de, a cada conjunto de instruções *PLTI* executadas (no momento fixado em uma (1) instrução), passar o controle ao *PROCESSADOR DE INTERRUPÇÕES* para que o mesmo simule e processe as interrupções.

Após colocar o desvio no *NRES*, o *INICIALIZADOR* passa o controle ao *DISPATCHER*.

Até este momento, todos os procedimentos são comuns a todos os programas virtuais, isto é, são executados a nível de programa real e uma única vez, pois são procedimentos de inicialização.

Após isto, os procedimentos são particulares a cada programa virtual e ocorrem em função da mudança de estado de cada um, de modo assíncrono, controlado pelo *DISPATCHER*, que sempre pesquisará em *TABUS* o próximo programa virtual pronto para ser processado, segundo o estado em que cada um se encontre.

III.5.2. LÓGICA DO PROCESSAMENTO DE UM PROGRAMA VIRTUAL

Para facilidade de entendimento será descrito daqui em diante o que ocorre com um (1) determinado programa virtual. Na realidade, tudo isso acontece para todos os programas virtuais, em paralelo, e, como já foi dito, assincronamente, controlado pelo *DISPATCHER*, que sempre reiniciará a pesquisa do próximo programa virtual pronto para ser processado a partir do elemento de *TABUS* seguinte ao último programa virtual que estava sendo processado. Isto assegurará uma distribuição uniforme de atendimento a todos os programas virtuais, em função da atividade própria de cada um.

Após a inicialização do *SMT*, o *DISPATCHER*, escrito em *PLTI* de modo a permitir a execução concorrente com o *PROCESSADOR DE INTERRUPÇÕES*, entrará num ciclo de busca em *TABUS* até que algum terminal solicite a ativação de um programa virtual via comando *$ATIVE*. Isto significa que o *PROCESSADOR DE INTERRUPÇÕES* terá recebido algum texto completo, e mudado o estado do programa virtual associado ao terminal que enviou o texto para *INATIVO FE*.

O *DISPATCHER* então identificará que esta ação ocorreu, e passará o controle ao *INTERPRETADOR DE COMANDOS* para processamento do comando recebido.

O *INTERPRETADOR DE COMANDOS* fará a análise sintática do comando ($ como primeiro caractere, validade em relação ao estado atual do programa virtual, etc.). Em caso de erro, enviará mensagem correspondente ao terminal e passará o programa virtual de novo ao estado *INATIVO EP*. Em caso positivo, isto é, do texto recebido ter sido o comando *$ATIVE*, o *INTERPRETADOR DE COMANDOS* ativará o programa virtual, alterando o elemento de *TABUS* correspondente, movendo para ele o *PC* reserva e, para retorno da operação de E/S, *FF* (indicação para o *Interpretador PLTI* reexecutar a operação de E/S; alterará o estado do programa virtual para *INATIVO MP* (mensagem sendo transmitida), e verificará se existe área de disco apontada pelo elemento de *TABUS*, caso de *$ATIVE* após *$FIM;* se não existe, obtém a próxima posição disponível, primeiro *$ATIVE* para este programa virtual, colocando o endereço desta posição no elemento de *TABUS*, enviando mensagem de programa virtual ativo ao terminal correspondente.

Novamente o controle será passado ao *DISPATCHER*, que reiniciará o ciclo de busca em *TABUS*, só que agora encontrará um programa virtual pronto para ser processado, assim que a mensagem tenha sido completamente transmitida pelo *PROCESSADOR DE INTERRUPÇÕES*, que mudará o estado do programa virtual para *ATIVO OK*. O programa virtual receberá o controle do *DISPATCHER*, isto é, receberá no REG A o retorno da operação de E/S obtido em *TABUS*, terá sua área de dados atualizada se o último programa virtual processado não for ele mesmo (a partir da área correspondente em disco), e será reiniciado a partir do *PC* de *TABUS*.

Como o *PC* apontava para a primeira operação de E/S, de novo o controle será passado ao *SMT (EXECUTOR DE E/S)* que, de acordo com o tipo da operação de E/S, colocará o seu estado (o do programa virtual) em *ATIVO ENTRADA PENDENTE (READ)* ou *ATIVO SAÍDA PENDENTE (WRITE)*, iniciará a operação de E/S, passando em seguida o controle ao *DISPATCHER*. Se o programa virtual estiver no estado *ATIVO SP*, o *DISPATCHER* aguardará que a operação se complete (estado do programa virtual mude para *ATIVO OK*), continuando o ciclo de busca a partir do próximo elemento de *TABUS*.

Caso, enquanto o programa virtual prossegue sua execução, a saída deixe de ser pendente, isto é, o estado do programa virtual seja mudado para *ATIVO OK* (pelo *PROCESSADOR DE INTER*

RUPÇÕES), a próxima solicitação de execução de uma operação de E/S será processada do mesmo modo que qualquer solicitação de operação de E/S do programa, conforme descrito anteriormente, sendo em seguida passado o controle ao *DISPATCHER*, que recomeçará o ciclo de busca.

Se o programa virtual estiver no estado *ATIVO EP*, e portanto a operação de E/S for de entrada (*READ*), o *DISPATCHER* aguardará que a operação se complete (estado do programa mude para *ATIVO FE*), continuando o ciclo de busca a partir do próximo elemento de *TABUS*, do mesmo modo como é feito para o estado *ATIVO SP*. Aguardará, através do ciclo de busca em *TABUS*, que o terminal envie um texto qualquer que o *PROCESSADOR DE INTERRUPÇÕES* receberá para a área de recepção/transmissão na *TAF* associada ao terminal, mudando então o estado do programa virtual para *ATIVO FE*. Como o programa virtual está no estado *ATIVO FE*, o texto recebido deverá ser interpretado e o *DISPATCHER* salvará a área de dados do último programa virtual ativo, atualizará área de dados do programa virtual em processamento (se for o caso), e passará o controle ao *INTERPRETADOR DE COMANDOS*.

O *INTERPRETADOR DE COMANDOS*, em função do estado do programa virtual, identificará que a operação de entrada se completou e o texto deve ser movido para a área de dados do programa virtual. Se o texto for *$FIM*, o *INTERPRETADOR DE COMANDOS* colocará no elemento de *TABUS* correspondente ao programa virtual o retorno da operação de E/S com valor igual a um (= 1) (condição de *EOF* existente). Caso contrário, o texto será movido para a área de dados do programa virtual e será colocado no elemento de *TABUS* correspondente o retorno da operação de E/S com valor igual a zero (= 0) (operação realizada normalmente).

Em seguida, mudará o estado do programa virtual para *ATIVO OK* e passará o controle ao *DISPATCHER*, que recomeçará o ciclo de busca.

As posteriores operações de E/S solicitadas pelo programa estarão enquadradas nos procedimentos acima descritos, sempre se observando que tais procedimentos, embora descritos para um mesmo programa virtual, irão ocorrendo assincronamente para todos os programas virtuais que sejam ativados.

Quando o programa virtual receber a condição de *EOF* do terminal que o ativou, ele estará no estado *ATIVO OK*, pois terá sido reiniciado.

Desta maneira, o programa poderá seguir três alternativas:

a. ignorar a condição de *EOF*;

b. terminar normalmente dando *EXIT*, sem que antes disso solicite alguma outra operação de E/S;

c. solicitar, antes de dar *EXIT*, algumas operações de E/S de finalização do programa.

Na primeira alternativa a execução do programa virtual prosseguirá normalmente como se não houvesse o *EOF*. Na segunda alternativa a rotina *EXIT*, modificada pelo *INICIALIZADOR*, passará o controle ao *FINALIZADOR*, que desativará o programa virtual, passando seu estado para *INATIVO SP* e movendo mensagem de término de programa virtual para a de recepção/transmissão na *TAF* de modo a que o *PROCESSADOR DE INTERRUPÇÕES* providencie o envio da mensagem ao terminal e depois mude o estado do programa virtual para *INATIVO EP*. Na terceira alternativa as operações de E/S serão executadas normalmente até que o programa dê *EXIT*, quando então os procedimentos seguidos serão idênticos aos da segunda alternativa.

Após desativar o programa virtual, o *FINALIZADOR* verificará se ainda existe algum programa virtual ativo. Se todos estiverem no estado *INATIVO EP*, o *FINALIZADOR* terminará o *SMT*, passando o controle à rotina *EXIT* (término do programa real); caso contrário, passará o controle ao *DISPATCHER*, que começará o ciclo de busca, em *TABUS*, de programas virtuais prontos para serem processados segundo os procedimentos já descritos.

Esta é a forma geral de funcionamento do *SMT*, ressaltando-se que, embora descrito para um (1) programa virtual, na realidade ocorre de modo concorrente entre todos os programas virtuais que se ativem.

III.6. ESPECIFICAÇÃO DOS MÓDULOS DO SMT

III.6.1. INICIALIZADOR

O módulo *INICIALIZADOR*, pelo fato de ser executado uma única vez, está disposto na estrutura do *SMT* de modo tal que a área de memória ocupada por ele possa ser reutilizada pelo *SMT*, diminuindo com isso o gasto de memória.

Ele recebe o controle através do módulo *EXECUTOR DE E/S* quando este identifica a ocorrência da primeira operação de E/S no programa real. A sequência de instruções executadas até o *INICIALIZADOR* receber o controle é a seguinte:

a. Início da interpretação da instrução *READ* pelo *Interpretador PLTI*, que chama a micro-rotina, que a tratará, através do código de operação (1º *byte* da instrução).

b. O controle é passado à micro-rotina, que, no caso do *SMT*, será a correspondente ao *READ* ou *WRITE*, respectivamente *INT2B* e *INT2C*.

c. Tais rotinas preparam os registradores e o *stack* e desviam para a micro-rotina *INTES*, que é a saída geral das micro-rotinas de E/S.

d. A micro-rotina *INTES* guarda o PC do *Interpretador*, na *savearea* do mesmo (*SVAREA*), e constrói e executa um *CAL* para a *TRAL* do elemento correspondente ao arquivo lógico que solicitou operação de E/S.

e. É executado um *LBI* da ordem do elemento da *TRAL*, isto é, o REG B passa a conter a ordem do elemento da *TRAL* correspondente ao arquivo que solicitou a operação de E/S.

f. É executado um *JMP* para a *TAF* associada, isto é, desvio incondicional para a *TAF*.

g. É executado um *JMP* da *TAF* para o módulo *EXECUTOR DE E/S* do *SMT*, isto é, desvio incondicional para a rotina de E/S (*SMT*) que irá atender a solicitação do programa real.

h. Chamada, pelo *EXECUTOR DE E/S*, da rotina auxiliar *TRATAF*, que irá atualizar no *NRES* o endereço da *TRAL* e da *TAF* em uso.

i. Por ser a primeira vez que o *EXECUTOR DE E/S* recebe o controle, desvio incondicional para o *INICIALIZADOR*.

A sequência (item *a* até item *h*) será sempre executada quando da interpretação de instrução de E/S para o arquivo lógico que usa terminais, porém a partir da segunda vez o *EXECUTOR DE E/S* não mais desviará para o *INICIALIZADOR* (item *i*).

Tendo recebido o controle, o *INICIALIZADOR* começará a execução de suas tarefas.

A primeira delas é guardar o *PC* da operação de E/S, obtido na *SVAREA* do *Interpretador* (cujo endereço está na Pos 94 Hex do *NRES*) pela micro-rotina *INTES* no campo *Savearea que Contém o PC Reserva*.

A seguir é movido para o campo *Tamanho do Registro Lógico do Arquivo do SMT* o tamanho do registro contido na *TRAL*, cujo endereço foi colocado no *NRES* (Pos 70 Hex) pela rotina auxiliar *TRATAF*. Isto é feito através de uma estrutura de dados, que descreve a *TRAL*, *BASED* no endereço da *TRAL*.

A partir daí começará a criação de *TABUS* a partir da primeira *TAF* associada à *TRAL* do arquivo lógico que usa o *SMT*.

Para isso obtém no *NRES* (Pos 72 Hex) o endereço da *TAF* (que será a primeira apontada pela *TRAL*) em uso, atualizado quando o *EXECUTOR DE E/S* chamou a rotina auxiliar *TRATAF*.

Usando uma estrutura *BASED* no endereço da *TAF*, o endereço físico do terminal (linha de comunicação) é movido da *TAF* para o primeiro elemento de *TABUS*. Também é movido para o mesmo elemento o endereço da *TAF* (ponteiro para a *TAF* correspondente ao programa virtual) e estado *INATIVO EP* para o estado do pro-

grama virtual.

Em seguida prepara a interface da linha de comunicação, obtendo o endereço físico da *TAF*, carregando-o no acumulador e executando uma instrução *SELECT*, selecionando assim a interface.

A seguir são executados os diversos *controls* para preparação da interface: (*a*) é dado um *CNTR4* (*RESET* de *BUFFER* e *STATUS* ; (*b*) seguido de um *CNTR2*, antes colocando no acumulador (REG A) o formato da transmissão obtido da *TAF*; e, por fim (c) um *CNTR3*, precedido pela carga no acumulador (REG A) da velocidade de transmissão, também obtido da *TAF*.

Se o terminal estiver conectado por *modem* à interface, é executado ainda um comando *CNTR5* para requisição da portadora de modo permanente, e aguardada a resposta da portadora ou via comando *STAT1* (segunda palavra de *STATUS*) até que a mesma chegue. Caso contrário, se o terminal estiver conectado diretamente à interface (*loop* de corrente) tal procedimento não é feito.

É verificado ainda nesta *TAF* o indicador de última *TAF* associada à *TRAL*.

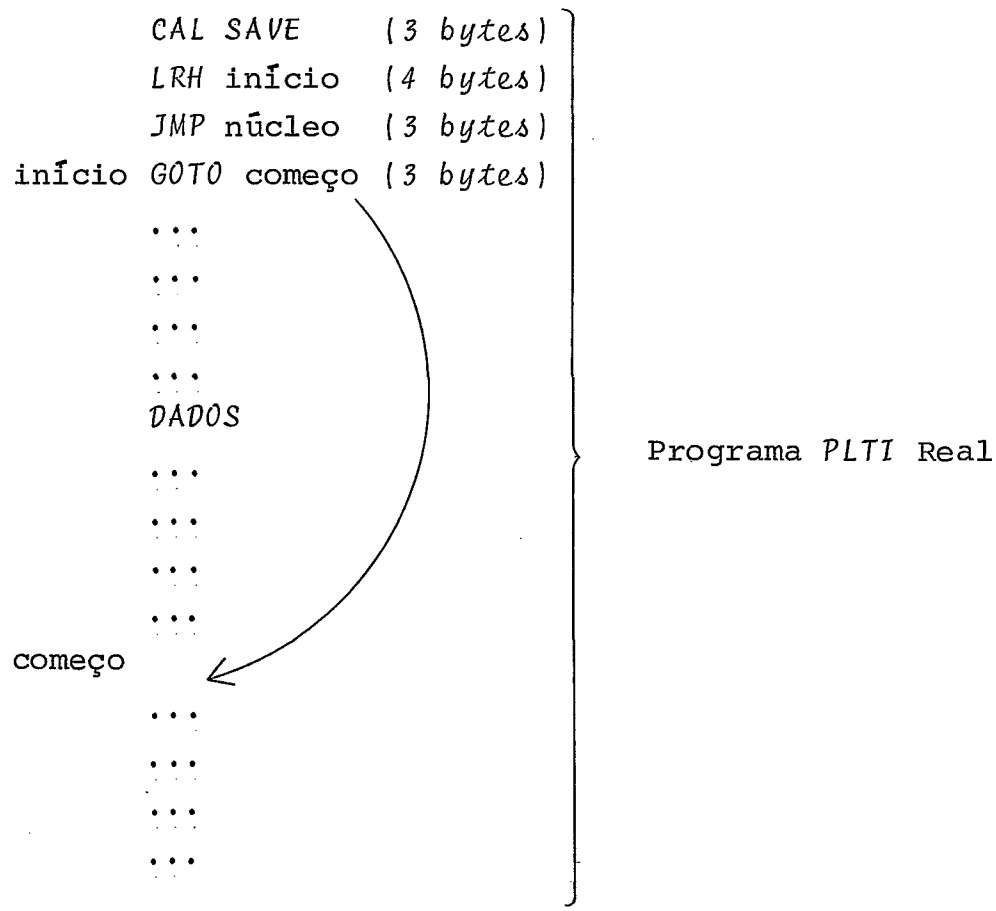
Em caso do indicador ligado, significa *TABUS* criado integralmente, sendo colocado no ponteiro do próximo elemento em *TABUS*, um ponteiro para o primeiro elemento de *TABUS*, fechando assim a lista circular.

Em caso contrário é colocado no último elemento de *TABUS* criado o ponteiro para o próximo, ou seja, para o que será criado a partir desta *TAF*, sendo repetido o processo de criação de elemento de *TABUS* e preparação da interface da linha de comunicação a partir da estrutura *BASED* no endereço da *TAF*.

O ciclo acima será repetido até que a última *TAF* seja processada, quando então o *INICIALIZADOR* guardará no campo *Ponteiro para o Elemento de TABUS Correspondente ao Programa Virtual em Processamento* o endereço do último elemento de *TABUS*.

A próxima tarefa será o cálculo do endereço e tamanho da área de dados do programa real. Isto é feito obtendo do *NRES* (Pos 36 Hex) o endereço do início do programa real, que está invertido (formato LH), colocando-o na forma correta (formato HL)

e somando-se 13 *bytes*, correspondentes às instruções-padrão existentes no início de um programa *PLTI*.



O endereço assim obtido é o endereço da área de dados do programa real que é então guardado na área de memória do *SMT* específica (*2 bytes*).

O tamanho da área de dados é calculado pela diferença entre os endereços do começo do programa real e do começo da área de dados, sendo que o endereço do começo do programa real está imediatamente antes do começo da área de dados.

A tarefa seguinte é calcular o tamanho da área de dados arredondado para um número inteiro de setores (em *bytes*). Este tamanho é o menor múltiplo de 512 em que caiba a área de dados. Isto é feito com o seguinte algoritmo:

. subtrair *1* do tamanho exato da área de dados;

. zerar os 9 *bits* de mais baixa ordem (à direita);

. somar 512 (#200) ao valor obtido no item anterior.

Fórmula para cálculo: *TAMARR = ((TAM-1) AND #FE00) +#200*

O tamanho arredondado é também guardado em área de memória específica do *SMT*.

Em sequência às suas tarefas, o *INICIALIZADOR* verificará se a área de disco disponível para uso pelo *SMT* é suficiente para guardar as áreas de dados de cada terminal e a área de dados reserva do seguinte modo:

a. Multiplicar o tamanho arredondado da área de dados pelo número de elementos de *TABUS* mais um (+ 1), correspondente à área reserva, e dividir por 512 (tamanho de um (1) setor de disco), obtendo assim a área total em setores de disco necessária.

b. Por comparação desse valor com o obtido no diretório do disco (o arquivo do *SMT* tem nome padrão - *SMTWRK* - e os dados *Endereço* e *Número de Setores* são obtidos do diretório do disco pela sub- rotina que trata diretório - *TDIRET*), é verificado se a área de disco é suficiente. Se for insuficiente, é dada mensagem correspondente, e o *SMT* encerra o programa real.

Sendo a área de disco suficiente, o *INICIALIZADOR* move para o campo *Ponteiro para a Próxima Área de Disco Disponível* o endereço da primeira área em disco disponível (obtido conforme acima).

A seguir, grava a área de dados do programa real na primeira área de disco, que é a área de dados reserva, chamando a rotina auxiliar *DSKAUX (ROM)*, que receberá como parâmetros apontados pelos registros HL o seguinte:

a. Endereço de disco (obtido do campo *Ponteiro para a Próxima Área de Disco Disponível* na forma disco/cil/trilha/setor.

b. Endereço e tamanho da área de dados obtidos do campo *Endereço e Tamanho de Área de Dados do Programa Real* (tamanho arredondado para número inteiro de setores).

No REG A o *INICIALIZADOR* receberá a função a executar (*2 = READ, 1 = WRITE*); a rotina é chamada para *WRITE*. Em seguida o campo *Ponteiro para a Próxima Área de Disco Disponível* é atualizado, passando a apontar para a segunda área de disco.

A partir deste ponto, o *SMT* está completamente inicializado e o *INICIALIZADOR* executa então suas duas últimas tarefas.

A penúltima tarefa é guardar os três primeiros *bytes* da rotina *EXIT*, que está no *NRES* (Pos 38 Hex), na *Savearea da Parte da Rotina EXIT* (área de memória específica do *SMT*) e montar em seu lugar um *JMP* para o *FINALIZADOR*, ou seja, mover o código de operação *44 Hex* para o 1º *byte*, obter o endereço do *FINALIZADOR*, e colocar na forma invertida (LH) no 2º e 3º *bytes*.

Finalmente, como última tarefa, o *INICIALIZADOR* guarda os três primeiros *bytes* do núcleo do *Interpretador PLTI* nas três primeiras posições do *PROCESSADOR DE INTERRUPÇÕES* e monta, nas três primeiras posições do núcleo do *Interpretador PLTI*, um *JMP* para o *PROCESSADOR DE INTERRUPÇÕES*, ou seja, obtém o endereço do *PROCESSADOR DE INTERRUPÇÕES* e coloca na forma invertida (LH) a partir do 2º *byte* e move código de operação *44 Hex* para o 1º *byte*.

Com isto o *INICIALIZADOR* encerrou suas tarefas e chama o *DISPATCHER*, módulo que está especificado adiante.

III.6.2. EXECUTOR DE E/S

O módulo *EXECUTOR DE E/S* é o *entry-point* principal do *SMT*, sendo aquele que recebe o controle quando alguma operação de E/S é solicitada para o arquivo lógico a que estão associados terminais.

Assim que recebe o controle, o *EXECUTOR DE E/S* guarda os registros que contêm os dados da operação de E/S (REGs C, D, E, H e L).

A primeira tarefa é chamar a rotina auxiliar *TRATAF* para atualizar no *NRES* os endereços da *TRAL* e da *TAF* em uso.

Em sequência, verifica se esta operação de E/S é a primeira solicitada. Caso afirmativo, é chamado o módulo *INICIALIZADOR*.

O *EXECUTOR DE E/S* identifica então qual operação de E/S está sendo solicitada: *READ* ou *WRITE*.

Isto é feito examinando o REG C. Se o seu conteúdo for 1, a operação é *READ*; se for 2, a operação é *WRITE*. Caso o conteúdo do REG C seja diferente de 1 e de 2, a operação não é prevista pelo *SMT*, sendo movido para o campo *Retorno*, no elemento de *TABUS* correspondente, o valor 10 (Hex A), que significa operação solicitada não suportada pelo *SMT*, chamando o módulo *DISPATCHER* (item *III.6.3.*) a seguir.

Sendo a operação de E/S válida, o *EXECUTOR DE E/S* preparará o elemento de *TABUS* e a *TAF* correspondente para início da operação de E/S.

Isto é feito, primeiro alterando-se na *TAF* o ponteiro para o último caractere lido/transmitido para zero através de estrutura *PLTI*, que descreve a *TAF*, *BASED* no endereço da *TAF* contida no elemento de *TABUS*.

Se a operação for *WRITE*, é movido o texto (81 Pos) apontado pelo REG DE, quando do recebimento do controle pelo *EXECUTOR DE E/S*, para a área de leitura/escrita da *TAF*, sendo movido para os dois caracteres após o último do texto o símbolo *Cr* (*Carriage-Return*), para sinalizar ao *PROCESSADOR DE INTERRUPÇÕES*, fim-de-texto.

No caso da operação ser *READ*, o *EXECUTOR DE E/S* muda o estado do programa virtual no elemento correspondente em *TABUS* para *ATIVO EP* (*Ativo Entrada Pendente*) e, no caso de ser *WRITE*, para *ATIVO SP* (*Ativo Saída Pendente*).

Ainda no caso da operação de E/S ser *READ*, guarda o endereço contido no REG DE e no elemento de *TABUS* correspondente ao programa virtual (campo *Endereço da Área de E/S*), para uso pelo *INTERPRETADOR DE COMANDOS*, quando da efetivação da operação de Entrada.

Isto faz com que, a partir deste momento, sempre que o *PROCESSADOR DE INTERRUPÇÕES* já execute as operações de E/S pedi

das pelo programa virtual, segundo algoritmo descrito adiante na especificação do referido módulo.

Após alterar *TABUS* e a *TAF*, o módulo *EXECUTOR DE E/S* desvia incondicionalmente para o *DISPATCHER*.

Estas são as tarefas do *EXECUTOR DE E/S*, que na realidade não executa operações de E/S, mas sim as inicia indiretamente, indicando para o *PROCESSADOR DE INTERRUPÇÕES* que operações devem ser iniciadas e controladas até o seu final, sendo portanto o *PROCESSADOR DE INTERRUPÇÕES* que na realidade centraliza as execuções de operações de E/S.

III.6.3. DISPATCHER

O módulo *DISPATCHER* é o responsável pela ativação de programas virtuais, através da pesquisa em *TABUS* de qual programa virtual está apto a ser ativado.

Sempre que recebe o controle, o *DISPATCHER* inicia o ciclo de busca em *TABUS* de um programa virtual pronto para ser processado, a partir do elemento de *TABUS* seguinte ao último que estava ativo.

Para executar esta operação é utilizada uma estrutura *PLTI* que descreve o elemento de *TABUS*, *BASED* no endereço do elemento de partida do ciclo. Este endereço é obtido do ponteiro para o próximo elemento de *TABUS* existente no elemento de *TABUS* apontado pelo campo *Ponteiro para o Elemento de TABUS Correspondente ao Programa Virtual em Processamento*.

Este processo é seguido sempre que o *DISPATCHER* continua o ciclo de busca em *TABUS*.

Em cada elemento, o *DISPATCHER* pesquisa o campo *Estado do Programa Virtual*, prosseguindo o ciclo de busca ou ativando o programa virtual, conforme o conteúdo do campo referido.

As ações do *DISPATCHER*, em função do estado de um programa virtual, são os seguintes:

ESTADO DO PROGRAMA VIRTUAL	AÇÃO DO *DISPATCHER*
1. *INATIVO EP*	Programa virtual inativo. Passa a pesquisar próximo elemento de *TABUS*.
2. *INATIVO FE*	Texto recebido para programa virtual inativo. Controle é passado ao *INTERPRETADOR DE COMANDOS*.
3. *ATIVO OK*	Programa virtual pronto para ser processado. Controle é passado ao mesmo a partir do *PC* contido em *TABUS*.
4. *ATIVO SP*	Programa virtual solicitou operação de saída que ainda está pendente. Passa a pesquisar próximo elemento de *TABUS*.
5. *ATIVO EP*	Programa virtual solicitou operação de entrada, que ainda está pendente. Passa a pesquisar próximo elemento de *TABUS*.
6. *ATIVO FE*	Operação de Entrada (*READ*) terminada. Controle é passado ao *INTERPRETADOR DE COMANDOS*.
7. *INATIVO TESTE EP*	Terminal associado a este programa virtual, que está inativo, solicitou teste de teclado. Passa a pesquisar próximo elemento de *TABUS*.
8. *INATIVO TESTE FE*	Texto recebido para programa virtual inativo, cujo terminal associado está testando teclado. Controle é passado ao *INTERPRETADOR DE COMANDOS*.
9. *INATIVO TESTE SP*	Terminal em teste de teclado. Passa a pesquisar próximo elemento de *TABUS*.
10. *INATIVO SP*	Terminal recebendo mensagem do *SMT*. Passa a pesquisar próximo elemento de *TABUS*.
11. *INATIVO MP*	Terminal recebendo mensagem de que programa virtual correspondente foi ativado. Passa a pesquisar próximo elemento de *TABUS*.

Antes de passar o controle, seja ao programa virtual , seja ao *INTERPRETADOR DE COMANDOS*, o *DISPATCHER* executa uma série de tarefas preparativas para a execução dos mesmos.

Se o programa virtual estiver no estado *INATIVO FE* ou *INATIVO TESTE FE*, o *DISPATCHER* passará o controle diretamente ao *INTERPRETADOR DE COMANDOS*.

Se estiver em qualquer outro estado, o *DISPATCHER* verificará se o programa virtual que está sendo ativado não é o mesmo que estava com controle, ou seja, se o endereço do elemento de *TABUS* correspondente ao programa virtual que está sendo ativado é diferente do *Ponteiro para o Elemento de TABUS Correspondente ao Programa Virtual em Processamento*, e nesse caso as seguintes ações são executadas:

a. Guarda endereço do elemento de *TABUS* correspondente ao programa virtual que será ativado.

b. Salva PC do programa virtual que estava sendo processado, contido na *savearea* do *Interpretador PLTI (SVAREA)*, cujo endereço está na Pos 74 Hex do *NRES*, no elemento correspondente em *TABUS*, que está apontado pelo campo *Ponteiro para o Elemento de TABUS...*

c. Salva área de dados do programa virtual que estava com o controle, na área de disco apontada pelo campo *Endereço da Área de Disco...* do elemento de *TABUS* correspondente, cujo endereço está contido no campo *Ponteiro para o Elemento de TABUS...* Para isto, chamar a rotina auxiliar, de acesso em disco , *DSKAUX (ROM)*, que recebe como parâmetros apontados pelos Registros HL uma área de memória com o seguinte conteúdo:

. endereço de disco (obtido em *TABUS*), na forma disco/cil/trilha/setor;

. endereço e tamanho da área de dados, obtidos do campo *Endereço e Tamanho da Área de Dados do Programa Real* (tamanho arredondado para número inteiro de setores);

e, no REG A, a função, codificada da seguinte forma:

. *1 - WRITE*

. *2 - READ*

d. Atualiza o *PC* do programa virtual que será ativado, existente na *savearea* do *Interpretador PLTI*, cujo endereço está na Pos 74 Hex do *NRES*, obtido do elemento correspondente em *TABUS*.

e. Atualiza área de dados do programa real, com a área de dados do programa virtual que será ativado, a partir da área de dados deste programa existente em disco, cujo endereço é obtido do elemento correspondente em *TABUS*. É chamada a rotina auxiliar *DSKAUX (ROM)* para leitura da área de disco diretamente para a área de dados do programa real, da mesma forma como anteriormente o foi para salvamento da área do programa virtual que estava com o controle, porém desta vez é usado como tamanho da área de dados o tamanho exato e não o arredondado.

Se o programa virtual que estiver sendo ativado for o mesmo que estava sendo processado, sua área de dados não é atualizada e portanto não é necessário ser feito o acima descrito.

O *DISPATCHER* verificará, em seguida, se o controle será passado ao programa virtual em processamento ou ao *INTERPRETADOR DE COMANDOS*.

Se o controle deve ser passado ao programa virtual, o *DISPATCHER* obtém do elemento correspondente em *TABUS* o código de retorno (1 *byte*) e o carrega no REG A, para logo após passar o controle ao programa via um comando *RETURN (RET)* que, pelo endereço contido no *stack* do *TI*, devolverá o controle à micro-rotina *INTES*, a partir de onde sempre será reativado todo programa virtual.

Se o controle deve ser passado ao *INTERPRETADOR DE COMANDOS*, o mesmo é chamado.

O *DISPATCHER*, para todos os efeitos, considera a ativação do *INTERPRETADOR DE COMANDOS* como se fosse uma ativação de

programa virtual, porque, na movimentação de um texto recebido na *TAF* pelo *PROCESSADOR DE INTERRUPÇÕES* para a área de leitura do programa virtual (feita pelo *INTERPRETADOR DE COMANDOS*), é necessário que a área de dados do programa real contenha a área de dados correspondente ao programa virtual, que será salva ou não dependendo do programa virtual que receber controle após o *INTERPRETADOR DE COMANDOS* ter realizado suas tarefas.

III.6.4. INTERPRETADOR DE COMANDOS

O módulo *INTERPRETADOR DE COMANDOS* tem a função de interpretar todos os textos recebidos do terminal associado a um programa virtual.

Estes textos podem ser *comandos* para o *SMT* ou *dados* para o programa virtual. Os *comandos* são identificados pela presença do caractere $ (*dolar*) na primeira posição, e, os *dados*, pela sua ausência na mesma posição. Em algumas situações, dependendo do estado do programa virtual, os textos recebidos serão considerados como *dados*, independentemente de haver ou não $ (*dolar*) na primeira posição dos mesmos, como veremos adiante.

O *INTERPRETADOR DE COMANDOS*, conforme descrito na especificação do *DISPATCHER* (item *III.6.3.*), receberá o controle quando o programa virtual estiver em um dos três estados seguintes: *INATIVO FIM DE ENTRADA (INATIVO FE)*, *ATIVO FIM DE ENTRADA (ATIVO FE)* e *INATIVO TESTE FIM DE ENTRADA (INATIVO TESTE FE)*.

Para cada um destes estados o *INTERPRETADOR DE COMANDOS* tem uma função distinta.

III.6.4.1. Funções do INTERPRETADOR para o Estado INATIVO FE

Se o estado for *INATIVO FE*, o *INTERPRETADOR* verifica o texto que foi recebido na *TAF* pelo *PROCESSADOR DE INTERRUPÇÕES*, acessando a mesma por uma estrutura *BASED* no endereço da *TAF* con

tido no elemento de *TABUS* correspondente ao programa virtual (os módulos do *SMT* que acessam a *TAF* sempre o fazem deste modo).

No caso do texto não conter $ (*dolar*) na sua primeira posição, o mesmo não é um *comando* e portanto o *INTERPRETADOR DE COMANDOS* o ignora, mudando o estado do programa virtual em *TABUS* para *INATIVO SP*, movendo mensagem de erro para *TAF*, posicionando ponteiro para próximo caractere recepção/transmissão.

No caso do texto ser um *comando* ($ na primeira posição) o *INTERPRETADOR DE COMANDOS* verifica se o mesmo é *$ATIVE* ou é *$TESTE*. Se não é igual a nenhum dos dois *comandos* mencionados, é executada a mesma ação do parágrafo anterior, ou seja, o texto é considerado como não sendo *comando*.

Se o texto for *$ATIVE*, o *INTERPRETADOR DE COMANDOS* inicializa o programa virtual da seguinte forma:

a. Move o elemento de *TABUS* correspondente ao programa virtual o *PC reserva*, contido no campo *Savearea Contendo o PC Reserva*.

b. Move para o campo *Retorno da Operação de E/S*, no elemento de *TABUS*, FF.

c. Altera o estado do programa virtual para *INATIVO MP*, no elemento correspondente em *TABUS*.

d. Move para a *área de recepção/transmissão* na *TAF* a mensagem "#0A,* *Programa Virtual Ativo* *, (CR)" e, para o *ponteiro para último caractere recepção/transmissão*, zero.

e. Verifica se o endereço da área de dados em disco, existente no correspondente elemento de *TABUS*, contém zeros. Senão, significa que o comando *$ATIVE* está sendo enviado por um terminal que já ativou algum programa virtual e o encerrou, e portanto já existe área em disco reservada para o programa virtual. Caso contrário (se contiver zeros), move o conteúdo do campo *Ponteiro para a Próxima Área de Disco Disponível* para o campo *Endereço da Área de Disco Correspondente à Área de Dados Associada ao Pro-*

grama Virtual no elemento de *TABUS* correspondente, e incrementa o endereço da próxima área de disco disponível do tamanho de uma área, fazendo o campo *Ponteiro para a Próxima Área de Disco Disponível* apontar para a área seguinte à que apontava.

Atualiza então a área de dados do programa real com uma cópia da área de dados reserva, apontada pelo campo *Ponteiro para a Área Reserva em Disco*, lendo diretamente da área de disco do *SMT* através da rotina auxiliar *DSKAUX (ROM)* usada conforme descrito na especificação do *DISPATCHER*.

6. Neste ponto o programa virtual está inicializado e o controle é passado ao *DISPATCHER*.

Se o texto for *$TESTE*, o *INTERPRETADOR DE COMANDOS* move mensagem de início de teste de teclado para a área de recepção/transmissão na *TAF*, altera o estado do programa virtual para *INATIVO TESTE SP*, e o controle é passado para o *DISPATCHER*.

III.6.4.2. Funções do INTERPRETADOR para o Estado ATIVO FE

Se o estado do programa virtual é *ATIVO FE*, significa que uma operação de E/S tipo *READ* que estava sendo executada para o programa virtual terminou.

Então o *INTERPRETADOR DE COMANDOS* acessa a *TAF* conforme descrito no item anterior e verifica se o texto é *$FIM*.

Se positivo move 1 (um) para o campo *Retorno da Operação de E/S* do elemento de *TABUS* correspondente ao programa virtual, significando fim de arquivo lógico. Muda o estado do programa virtual, no elemento de *TABUS*, para *ATIVO OK* e passa o controle ao *DISPATCHER*.

Se negativo a *TAF* contém um texto que deve ser movido para a área de leitura do programa virtual, cujo endereço está no campo *Endereço da Área de E/S* contido no elemento de *TABUS* correspondente ao programa virtual.

O *INTERPRETADOR DE COMANDOS*, através de estruturas *BASED*, move o texto da *TAF* para a área de E/S do programa virtual e move zeros para o campo *Retorno de Operação de E/S* (em *TABUS*), significando operação realizada normalmente. Muda o estado do programa virtual para *ATIVO OK*, no elemento correspondente em *TABUS*, e passa o controle ao *DISPATCHER*.

III.6.4.3. Funções do INTERPRETADOR para o Estado INATIVO TESTE FE

Se o estado do programa virtual é *INATIVO TESTE FE*, significa que acabou de chegar um texto do terminal que deve ser analisado, porque o terminal está executando a função teste do *hardware* do teclado.

Se o texto for *$FIMTESTE*, significa que o terminal encerrou o teste, e então o estado do programa virtual é alterado para *INATIVO SP*, movida mensagem de fim de teste de teclado para área de E/S na *TAF*, e sendo devolvido o controle ao *DISPATCHER*.

Caso contrário, o texto deve ser enviado de volta ao terminal e então o *INTERPRETADOR DE COMANDOS* move zeros para o campo *Ponteiro para o Último Caractere Transmitido/Recebido* (na *TAF*), muda o estado do programa virtual para *INATIVO TESTE SP* e passa o controle para o *DISPATCHER*.

O *INTERPRETADOR DE COMANDOS* não executa nenhuma operação de E/S. Sempre que precisa realizar alguma operação de E/S ele prepara tudo para a operação ser realizada pelo *PROCESSADOR DE INTERRUPÇÕES*, conforme descrito acima para o caso do estado do programa virtual ser *INATIVO TESTE FE*.

III.6.5. FINALIZADOR

O módulo *FINALIZADOR* tem a função de desativar programas virtuais. Recebe o controle diretamente da rotina auxiliar

EXIT, residente no *NRES* (Pos 38 Hex), modificada pelo *INICIALIZADOR*.

Assim que recebe o controle, o *FINALIZADOR* move mensagem de fim de programa virtual para área de E/S na *TAF*, altera o estado do programa virtual para *INATIVO SP* no elemento de *TABUS* correspondente (apontado pelo campo *Ponteiro para o Elemento de TABUS Correspondente ao Programa Virtual em Processamento*, desativando com isso o programa virtual. Espera que a mensagem seja completada no terminal, isto é, que o estado do programa virtual mude de *INATIVO SP* para *INATIVO EP*.

Em seguida inicia um ciclo de busca em *TABUS*, a partir do elemento seguinte ao corrente, verificando se todos os programas virtuais estão no estado *INATIVO EP* ou *INATIVO SP*. Caso algum programa virtual esteja no estado *INATIVO SP*, o *FINALIZADOR* aguarda que o mesmo passe para o estado *INATIVO EP* antes de continuar o ciclo de busca em *TABUS*. Se encontrar algum programa que não esteja em nenhum destes dois estados, o *FINALIZADOR* interrompe o ciclo de busca e assinala um indicador para o *DISPATCHER*, significando que, quando o mesmo ativar o próximo programa virtual, o fará através de desvio para a *INTES* (*JMP*) e não com um *RETURN* para a *INTES* satisfazendo a operação de E/S pendente, porque neste caso não haverá operação de E/S pendente, já que o programa que estava ativo deu *EXIT* (se encerrou).

Caso contrário, todos os programas virtuais estão desativados e portanto o *SMT* será desativado, encerrando sua execução. Isto é feito restaurando-se a rotina *EXIT* (3 primeiras posições a partir da Pos 38 Hex do *NRES*) a partir do campo *Save-area da Parte Rotina EXIT*, movendo zeros para o desvio para o *PROCESSADOR DE INTERRUPÇÕES* (*NRES*, Pos 74 Hex) e desviando para a rotina *EXIT* já restaurada, que agora realmente encerrará o *SMT*, chamando o *CPROG*.

III.6.6. PROCESSADOR DE INTERRUPÇÕES

O *PROCESSADOR DE INTERRUPÇÕES* é um módulo especial do

SMT que simula e processa as *interrupções* de E/S, alterando o estado de cada programa virtual conforme as operações de E/S *iniciadas* pelo *EXECUTOR DE E/S* se desenvolvam.

Como as *interrupções* de E/S são simuladas pelo *Interpretador PLTI*, o *PROCESSADOR DE INTERRUPÇÕES* será ativado a cada vez que for executada uma micro-instrução *PLTI*; por essa razão o mesmo é escrito em *ASSEMBLER*, se referenciando a dados do *SMT* via símbolos externos.

O *PROCESSADOR DE INTERRUPÇÕES* é ativado pelo *Interpretador PLTI* via um desvio incondicional (*JMP*) para o mesmo, ali construído pelo *INICIALIZADOR* quando da solicitação da primeira operação de E/S pelo programa real.

Ao receber o controle o *PROCESSADOR DE INTERRUPÇÕES* salva o conteúdo dos registros e em seguida acessa *TABUS*, e começa um ciclo de busca de programas virtuais que estejam em *estados* que indicam que uma operação de E/S está se realizando.

Estes *estados* são: *INATIVO ENTRADA PENDENTE*, *ATIVO SAÍDA PENDENTE*, *ATIVO ENTRADA PENDENTE*, *INATIVO TESTE ENTRADA PENDENTE*, *INATIVO TESTE SAÍDA PENDENTE*, *INATIVO SAÍDA PENDENTE* e *INATIVO MENSAGEM PENDENTE*.

Na primeira vez em que o *PROCESSADOR DE INTERRUPÇÕES* é ativado, o ciclo de busca é iniciado a partir do primeiro elemento de *TABUS*; nas demais é iniciado a partir do elemento seguinte ao correspondente ao terminal atendido pela última vez.

A cada vez que o *PROCESSADOR DE INTERRUPÇÕES* é ativado, ele atende a apenas uma (1) operação de E/S e retorna ao *Interpretador PLTI*, com o objetivo de uma melhor distribuição de tempo.

Ao identificar um programa virtual em um dos *estados* acima, o *PROCESSADOR DE INTERRUPÇÕES* guarda um ponteiro para este elemento, obtém de *TABUS* o endereço físico do terminal a ele associado, carrega-o no Acumulador (REG A) e executa um *SELECT*, seguido de um comando *STATUS* que coloca o estado (primeira palavra) da interface selecionada no REG A (Acumulador).

O REG A contém então uma configuração de *byte* que expressa o estado da interface do seguinte modo:

BIT 7 = 1 *BUFFER* de transmissão vazio
BIT 6 = 1 Registro de transmissão vazio
BIT 5 = X Não usado
BIT 4 = X Não usado
BIT 3 = 1 Erro de enquadramento (recepção)
BIT 2 = 1 Erro de paridade (recepção)
BIT 1 = 1 Erro de sobreposição (recepção)
BIT 0 = 1 Dado disponível para leitura

Se o *estado* do programa virtual é *ENTRADA PENDENTE*, a operação de E/S é *recepção*; se é de *SAÍDA PENDENTE*, a operação de E/S é *transmissão*.

No caso de ser *recepção*, o *PROCESSADOR DE INTERRUPÇÕES* verifica se algum dos *bits* 1, 2 ou 3 está ligado. Se sim, significa que houve *erro de recepção*, sendo dada mensagem correspondente mostrando o erro nas luzes do *TI*, um *CNTR4* (*RESET* de *BUFFER* e *STATUS*), e prossegue a operação conforme próximo parágrafo. Senão, o *PROCESSADOR DE INTERRUPÇÕES* verifica se o *bit* 0 está ligado; caso positivo, existe um caractere que chegou à interface e está pronto para ser lido, e então ele inicia uma operação de *READ* para a interface, de modo a transferir o *byte* recebido da interface para o Acumulador (REG A), incrementa o *Ponteiro para o Último Caractere Recebido/Transmitido* na *TAF*, e através do mesmo guarda o conteúdo do REG A na área de recepção /transmissão da *TAF*.

Verifica se o REG A contém (cr) ; se sim, ocorreu um fim de entrada, e então muda o estado do programa virtual para *FIM DE ENTRADA*, restaurando o conteúdo dos registros salvos ao receber o controle e retornando em seguida ao *Interpretador PLTI* através de um *RETURN*.

Se o *bit* 0 do *STATUS* não estiver ligado, significa que ainda não chegou nenhum caractere naquela interface, e o *PROCESSADOR DE INTERRUPÇÕES* simplesmente retorna ao *Interpretador PLTI* através de um *RETURN*, tendo antes restaurado os REGs (A, B, C, D, E, H, L).

No caso de ser *transmissão*, o *PROCESSADOR DE INTERRUPÇÕES* verifica se o *bit* 7 do *STATUS* está ligado (= 1). Caso não esteja, a transmissão do último caractere ainda não acabou e então o controle é devolvido ao *Interpretador PLTI*, conforme parâ

grafo anterior.

Caso o *bit* 7 esteja ligado, o *BUFFER de Transmissão* está vazio e a *transmissão* de um novo caractere pode ser iniciada. Nesse caso, antes de iniciar a *transmissão*, o *PROCESSADOR DE INTERRUPÇÕES* verifica se a interface da linha de comunicação relativa ao programa virtual está *OK* (pronto para *transmissão*) através de um comando *STAT1* que obtém da interface a segunda palavra de *STATUS* e a coloca no REG A.

O formato da segunda palavra de *STATUS* é o seguinte:

BIT 7 = 1 Portadora presente (*MODEM*) ou linha conectada (*LOOP*)
BIT 6 = 1 *MODEM* ligado
BIT 5 = 1 *MODEM* pronto para transmitir
BIT 4 = X Não usado
BIT 3 = X Não usado
BIT 2 = 1 Não usado
BIT 1 = 1 Linha de comunicação selecionada
BIT 0 = X Não usado

Se os *bits* 6 e 5 (Hex 60) não estiverem ligados, existe problema na linha de comunicação e a *transmissão* não pode ser realizada. Neste caso o *PROCESSADOR DE INTERRUPÇÕES* dá mensagem correspondente nas luzes e volta ao núcleo do *Interpretador PLTI*.

Caso contrário, a *transmissão* pode ser feita e o *PROCESSADOR DE INTERRUPÇÕES* incrementa na *TAF* o *Ponteiro para o Último Caractere Recebido/Transmitido* de um (1), e carrega o *byte* assim apontado no REG A.

Executa em seguida um *WRITE* que escreve o conteúdo do Acumulador (REG A) no *BUFFER de Transmissão*, iniciando a operação de E/S. Após transmitir o caractere, o *PROCESSADOR DE INTERRUPÇÕES* verifica se o caractere era (Cr) (*Carriage-Return*); se sim, verifica se o caractere seguinte também é (Cr); se sim, altera o estado do programa vertical para o estado seguinte , conforme descrito no autômato que descreve as mudanças de estado do programa virtual.

Em seguida retorna ao núcleo do *Interpretador PLTI*, restaurando os Registradores salvos quando recebeu o controle.

O critério adotado de existirem dois caracteres (Cr) seguidos para significar fim de *transmissão* para o *PROCESSADOR DE INTERRUPÇÕES* objetiva permitir que hajam caracteres de controle no texto, sem obrigatoriedade de posição, cabendo ao *EXECUTOR DE E/S* colocar os dois caracteres (Cr) no fim do texto quando inicia operação de E/S *WRITE*.

O módulo *PROCESSADOR DE INTERRUPÇÕES*, sempre que recebe controle, verifica em *TABUS* se o próximo elemento, que representa um programa virtual, está em um dos estados *ENTRADA PENDENTE* ou *SAÍDA PENDENTE*.

Ao encontrar um elemento que satisfaça tais condições, verifica se o mesmo pode prosseguir ou não uma operação de E/S e, após o processamento com este elemento se encerrar (tendo sido prosseguida ou não a operação de E/S), o controle é devolvido ao *Interpretador PLTI* para continuar a execução do programa virtual que estava com o controle, ao invés de só devolver o controle quando processar de fato uma operação de E/S para um determinado elemento de *TABUS*.

A razão disto é unicamente uma melhor distribuição de tempo de uso da CPU pelos diversos módulos do *SMT* e pelos programas virtuais. O ajuste deste tempo fica então restrito somente ao número de instruções executadas pelo *Interpretador PLTI* antes de desviar para o *PROCESSADOR DE INTERRUPÇÕES*, que atualmente está escolhido como uma (1).

IV. IMPLEMENTAÇÃO

IV.1. CONFIGURAÇÃO DO EQUIPAMENTO

IV.1.1. EQUIPAMENTO PROPRIAMENTE DITO

A implementação do *SISTEMA MULTI-TERMINAL (SMT)* no *TERMINAL INTELIGENTE (TI)* do NÚCLEO DE COMPUTAÇÃO ELETRÔNICA da UNIVERSIDADE FEDERAL DO RIO DE JANEIRO (NCE/UFRJ) foi executada com o equipamento já disponível para usuários em geral, cuja configuração é a seguinte:

a. CPU INTEL 8008 (micro-processador) com 16 *Kbytes* de memória principal (máximo).

b. Console composto de teclado com painel e vídeo.

c. Disco removível de 10 *Mbytes*, com setores de 512 *bytes*.

d. Leitora de cartões de 300 cpm.

d. Impressora serial de 165 caracteres/seg.

e. *Stack* de 1024 palavras de 1 *byte*, acessável como periférico via instrução *POP* (obtém uma (1) palavra do *stack*) e *PUSH* (põe uma (1) palavra no *stack*).

f. Linhas de comunicação assíncrona, com velocidades até 9600 *bits*/seg para ligação local ou remota. Estão instaladas duas linhas de comunicação nas quais foi testado o *SMT*:

- uma linha de comunicação com endereço #10 (velocidade 110 *bits*/seg e configuração ASCII, no terminal EMBRACOMP);
- uma linha de comunicação com endereço #11 (velocidade 4800 *bits*/seg e configuração ASCII, no terminal EMBRACOMP).

IV.1.2. SOFTWARE DISPONÍVEL

O *software* está dividido em duas partes, a seguir descritas.

IV.1.2.1. Simulação do Software do TI no B-6700 (Cross-Software)

Simula a configuração do *TI* no B-6700 e torna disponível o *Sistema Operacional do TI (SOCO)* sob esta configuração simulada.

Contém o Montador *ASSEMBLER*, *Compilador PLTI*, *Editor de Referências Externas (REFEX)*, e utilitários de manipulação com arquivos.

É executado sob o sistema operacional do B-6700 e utiliza os cartões de controle (*WFL*) para sua ativação.

IV.1.2.2. Sistema Operacional do TI (SOCO)

É o sistema operacional em disco já disponível no *TI*, e contém o mesmo *software* que é simulado no B-6700, à exceção do Montador *ASSEMBLER*, que ainda está sendo desenvolvido.

Para montagem de programas codificados em *ASSEMBLER* ainda é necessário usar o Simulador no B-6700.

O *Carregador de Programas (CPROG)* é o responsável pela ativação de programas a pedido do usuário. O processo normal de desenvolvimento de programas é a codificação dos mesmos em linguagem *PLTI*, compilação para obtenção de código interpretável e edição via *REFEX* que gera módulo executável, para a partir daí ser executado conforme explicado acima, via comando ao *CPROG*.

IV.2. PROBLEMAS ENCONTRADOS

Na construção do *SMT* para a configuração do *TI*, anteriormente descrita, algumas premissas foram estabelecidas. Dentre elas as mais importantes são:

a. A linguagem a ser usada deve ser de preferência a *PLTI*, de modo a se obter uma maior portabilidade do mesmo além das vantagens de se desenvolver em linguagem de alto nível.

b. Manter, o máximo possível, a independência de arquivo lógico com arquivo físico, ou seja, independência de periférico.

c. Preparar o *SMT* para futuras extensões, isto é, na sua construção prever facilidades que permitam que alterações em uma determinada função se concentrem apenas em um (1) módulo do *SMT* e que alterações neste módulo não influam em outros.

IV.2.1. DESCRIÇÃO DOS PROBLEMAS E SOLUÇÕES ENCONTRADAS

Como decorrência das premissas estabelecidas, alguns problemas surgiram, motivados por detalhes de construção do *SOCO* e do Simulador do *SOCO* no B-6700. Outros problemas ocorreram quando dos testes do *SMT* já no *TI*.

A seguir relatamos tais problemas e as soluções que, após estudos com a equipe de desenvolvimento do *software* do *TI*, foram desenvolvidas.

IV.2.1.1. Comunicação entre Programa Virtual e o SMT

a. PROBLEMA:

Como explicado anteriormente, um programa virtual é caracterizado quando o programa real está sendo executado para um dado terminal que se ativou. Durante o processo concorrente de execução de programas virtuais ocorrerá o término de alguns deles antes dos outros. Isto implica que o retorno para a microrotina *PLTI*, que é saída geral das operações de E/S para a rotina de E/S solicitada, não poderá ser feito via retorno para a rotina *ASSEMBLER* pelo *PLTI* (instrução *RET ASSEMBLER*), pois no processador não haverá instrução de chamada (*CAL*) pendente.

Sempre que ocorrer perda de controle de um programa virtual por seu término normal (*EXIT*) este problema fica configurado e a linguagem *PLTI* não tem recursos para resolvê-lo.

b. SOLUÇÃO:

Decidiu-se construir uma pequena parte do *SMT* em *ASSEMBLER*, sendo justamente a parte que faz a ligação *programa virtual/SMT/programa virtual*. Esta parte, cujo nome é o mencionado na concatenação de arquivos quando da execução do programa real, contém então a passagem de parâmetros necessários ao *SMT* propriamente dito, que está escrito na linguagem *PLTI* (*SMTMOD*); e no retorno deste, através de parâmetros de retorno, devolve o controle ao programa virtual através de instrução *RET ASSEMBLER*, quando o programa virtual anteriormente com o controle não tiver terminado (*EXIT*), ou através de um desvio incondicional (*JMP*), quando o programa anterior tiver terminado.

O endereço de retorno é obtido quando da primeira solicitação de E/S (pelo programa real) e construído no *JMP* através de uma rotina *ASSEMBLER* que só é executada uma vez, sendo o espaço que ocupa reutilizado pela *Tabela de Usuários* (*TABUS*).

IV.2.1.2. Teste do SMT no TI

a. PROBLEMA:

O *SOCO* não estava preparado para se poder testar o *SMT* diretamente no *TI* pelo fato do *CPROG* não estar construindo as

TAFs do *SMT*, por ser a primeira *rotina de E/S* composta de módulos em *ASSEMBLER* e *PLTI*, etc. .

b. SOLUÇÃO:

Montar ou compilar os módulos do *SMT* no Simulador do *SOCO* no B-6700 (*SOS*), compilar o programa de teste (*SMTEST*), montar *TRALs* e *TAFs* necessárias em *ASSEMBLER* e, após, editar o programa de teste para serem resolvidos os símbolos externos (comando */MONTE*, equivalente ao *REFEX* do *SOCO* no *TI*) de modo a gerar um programa executável no *TI* incluindo inclusive o *Interpretador PLTI* (único) para o programa de teste e a parte do *SMT* em *PLTI*.

Através do utilitário */CAD* gerar arquivo em cartão compatível com o programa de carga do *TI* e carregar pela leitora de cartões diretamente, tornando assim o teste do *SMT* possível.

Deste modo, nesta fase o *SMT* não foi tratado pelo *SOCO* como rotina de E/S, tendo sido compilado como sub-rotina, inclusive pelo fato de não necessitar do *SOCO* para ser executado (carga direta de leitora de cartões).

IV.2.1.3. Manuseio de Teclado de Terminais-Vídeo

a. PROBLEMA:

Os terminais conectados ao *TI* são do tipo EMBRACOMP com protocolo *Teletype* (*TTY*) e portanto não mostram na tela o que está sendo digitado quando em modo duplex (*full*), e muito menos permitem apagamento de caracteres já digitados, pelo simples fato de já terem sido transmitidos (a transmissão é caractere a caractere, e não de textos).

b. SOLUÇÃO:

Modificação no *PROCESSADOR DE INTERRUPÇÕES* para:

- dar *ECO* do caractere recebido, de modo que, ao se digitar uma tecla qualquer, o caractere correspondente apareça imediatamente na tela;
- detetar digitação da tela *RUB/OUT* (*DEL*) que passa a

significar *Apagar Último Caractere Digitado*, e como resposta fornecer na tela, entre barras invertidas ( ), o caractere apagado.

Exemplo: TERMAN
Teclado *RUB/OUT* fica: TERMAN\N\
Teclado *RUB/OUT* fica: TERMAN\N\A\
Corrigida a palavra: TERMAN\N\A\INAL

IV.2.1.4. Obtenção do Endereço e Tamanho do Arquivo de Trabalho do SMT (SMTWRK)

a. PROBLEMA:

O método inicial de obtenção do endereço em disco e do número de setores do arquivo do *SMT*, inicialmente adotado, foi através da rotina *TDIRET* com a função *Procurar*. Isto significou o aumento de memória mínima necessária pelo *SMT* em aproximadamente 2 *Kbytes*, que só é usada uma única vez.

b. SOLUÇÃO:

Estudando-se outras alternativas de obtenção, optou-se por uma que simplesmente dispensava o uso da *TDIRET* e consequentemente dos 2 *K* adicionais, que foi a seguinte:

O *CPROG*, ao detetar arquivo lógico associado ao *SMT*, quando da construção das *TAFs* concatenadas, aproveitando a rotina de procura de arquivo em diretório do *SOCO* já existente no próprio *CPROG*, obtendo o endereço e o tamanho em setores do arquivo do *SMT* e colocando na primeira *TAF* da concatenação, economizando toda a tarefa do *SMT* de fazê-lo e ainda os 2 *Kbytes* antes necessários.

IV.2.1.5. Como Testar o SMT no TI, sob o controle do SOCO

a. PROBLEMA:

Após o *SMT* estar funcionando em forma *stand-alone*, sur-

giu o problema de torná-lo disponível no disco do *SOCO* para uso por programas, em *PLTI*, de aplicações em desenvolvimento.

b. SOLUÇÃO:

b.1. Modificar o modo de concatenação de arquivos anteriormente previsto na definicação do *SMT*. No modo anterior o *nome externo* seria caracterizado como o endereço físico da linha de comunicação concatenada; o novo modo prevê um utilitário para a definição de linhas de comunicação através de mneumônicos. De uma forma conversacional o utilitário cria um arquivo vazio, apenas uma entrada no diretório de disco, que contém na própria entrada do diretório as características da linha de comunicação; o *nome externo* será então o nome do arquivo, cuja entrada no diretório descreve a linha de comunicação, assim por exemplo:

Seja o comando de execução:
PROG1 ARQTERM = (LINK01: SMT, LINK02: SMT, LINK03: SMT).

LINK01, LINK02 e LINK03 são nome de arquivos cujas entradas no diretório de disco contêm as características físicas das linhas de comunicação associadas, tais como endereço físico, velocidade de transmissão e formato de transmissão, tornando assim extremamente flexível e prática a definição da rede a ser usada pelo programa de aplicação.

b.2. Montar as rotinas do *SMT* em *ASSEMBLER* no *SOS* (B-6700), gerar *deck*-objeto em cartão, e gravá-lo no disco do *TI (SOCO)*. Compilar diretamente no *TI*, sob o *SOCO*, as rotinas *PL-TI* do *SMT*, gerando com isso *deck*-objeto

Compilar no *TI* o programa de teste do *SMT*, *SMTEST*, e *linkeditá-lo* com o *REFEX*. A partir daí o programa pode ser executado com o *CPROG*, usando concatenação, etc. .

IV.2.1.6. Como Fazer Referência ao Interpretador PLTI do Programa do Usuário

a. PROBLEMA:

Atualmente o *Interpretador* é editado junto com o progra

ma *PLTI* através do *REFEX*. Isto porque ele é tratado como um símbolo externo.

A rotina do *SMT*, por ser rotina de E/S, é carregada pelo *CPROG* e portanto não há meios de resolver a referência externa ao *Interpretador* já incorporado ao programa principal em *PLTI*.

Futuramente, o *Interpretador* será incorporado ao *Núcleo Residente (NRES)* e este problema estará resolvido. Porém, resta o problema atual.

b. SOLUÇÃO:

Transmitir o endereço do *Interpretador* para a rotina raiz do *SMT*, esta em *ASM*, através de uma posição no *NRES*. Esta posição é preenchida pela rotina *RPPLTI*, que é normalmente chamada ao início de todo programa *PLTI*. A rotina *SMTMOD* pega o endereço do *Interpretador* no *NRES* e desvia direto para lá, ativando o *SMT* escrito em *PLTI*.

As soluções acima foram implementadas e o *SMT* acha-se funcionando plenamente no *TI* do NCE/UFRJ, disponível a todos os seus usuários.

V. CONCLUSÕES

V.1. RESULTADOS OBTIDOS

Durante a definição do *SMT* inúmeros cuidados foram tomados com o objetivo de se garantir o máximo de confiabilidade ao *SMT*. Itens como tempo de resposta, gasto de memória e de disco, uso de terminais locais versus terminais remotos (com *Modem*), perda de caracteres digitados por sobreposição na chegada (*over run*) foram estudados, de modo a se garantir que a arquitetura do sistema, para atender a um número de aproximadamente dez terminais, se tornasse viável.

Ainda durante a codificação do *PROCESSADOR DE INTERRUPÇÕES*, onde a maioria dos itens acima mencionados influem diretamente, tomou-se o cuidado de otimização de tempo para obtenção dos resultados esperados.

Durante a implementação do *SMT* verificou-se que a importância dada a tais itens foi excessiva, principalmente pela adoção de eco (retransmissão) de cada caractere recebido de cada terminal (linha de comunicação), assim como pela eliminação da rotina *TDIRET* que, com a rotina de E/S *DISCO* necessária à *TDIRET*, economizaram cerca de 2 *Kbytes* na memória mínima necessária ao *SMT*.

Vejamos então, item a item, os resultados esperados e a conclusão a que se chegou.

V.1.1. TEMPO DE RESPOSTA VERSUS SOBREPOSIÇÃO DE CARACTERES NA RECEPÇÃO (OVERRUN)

Na época da definição do *PROCESSADOR DE INTERRUPÇÕES*, módulo do *SMT* encarregado de executar as operações de E/S, foi determinado que o mesmo deveria ser ativado de uma em uma micro-instrução *PLTI*, isto é, antes de ser executada a micro-instrução o *Interpretador PLTI* é interrompido e o controle passado ao *PROCESSADOR DE INTERRUPÇÕES*.

O fato de a *interrupção* ser gerada a cada micro-instrução *PLTI* foi arbitrariamente estabelecido, podendo, durante a implementação do *SMT*, ser ajustado para um intervalo maior de instruções, dependendo do tempo gasto pelo *PROCESSADOR DE INTERRUPÇÕES* para processar uma operação de E/S, do tempo médio de execução de uma micro-instrução *PLTI* e ainda do tempo de chegada de caracteres nas interfaces, isto é, o tempo decorrido entre a chegada de um caractere e outro em qualquer das interfaces.

Para atendimento uniformemente distribuído aos programas virtuais, o *PROCESSADOR DE INTERRUPÇÕES* utiliza a técnica de pesquisar, em tabela circular (*TABUS*). aqueles que estão com pendências de entrada ou saída. Isto é feito sempre a partir do elemento da tabela seguinte ao último elemento atendido, de modo a garantir para qualquer elemento que representa um programa virtual a mesma chance de atendimento.

Assim, para verificar se o número mínimo de uma micro-instrução *PLTI* entre cada *interrupção* é aceitável, vamos comparar então o Intervalo de Tempo Máximo que ocorre entre dois atendimentos a um mesmo Terminal (*ITMT*) com o Intervalo de Tempo entre a chegada de dois Caracteres consecutivos, independentemente de Interface (*ITCI*).

Este último, o *ITCI*, para um terminal, em termos médios, seria de 200 ms, admitida a média de 300 car/min de um datilógrafo. Como o que precisamos é do *ITCI* em termos instantâneos, vamos assumir o dobro da média como o número a ser usado para os cálculos, o que nos fornece o *ITCI* de 100 ms *para um terminal*.

Como estamos admitindo que até 10 terminais sejam conectados de uma só vez ao *TI*, neste momento, e ainda como a distribuição de atendimento aos terminais é uniforme (cada qual tem uma chance a cada ciclo de busca), podemos assumir o *ITCI* para 10 terminais como sendo:

$$ITCI_{10} = \frac{100}{10} = 10 \text{ ms}$$

Nota: Não estamos englobando no $ITCI_{10}$, para efeito de comparação com o *ITMT*, o tempo gasto para a transmissão do caractere do terminal pela linha de comunicação, pois a transmissão é simultânea com os outros processos.

Vamos, agora, para que possamos efetuar nossa avaliação, calcular o *ITMT*, que no máximo poderá ser igual ao *ITCI*; caso contrário a sobreposição de caracteres poderá ocorrer, isto é:

$$ITMT = ITCI_{10} = 10 \text{ ms}$$

O *ITMT* é função do Tempo Gasto (máximo) pelo *PROCESSADOR DE INTERRUPÇÕES (TGPI)* e o Tempo Médio de uma Micro-Instrução *PLTI (TMMI)*.

O *TGPI* é o maior tempo possível de ser consumido pelo *PROCESSADOR DE INTERRUPÇÕES* em qualquer de suas funções. Para estabelecê-lo, vamos então obter os tempos de cada função do *PROCESSADOR DE INTERRUPÇÕES*. Tais tempos foram calculados usando-se como base o ciclo do micro-processador INTEL 8008 (2,5μs) e o número de ciclos necessários a cada instrução *ASSEMBLER* do mesmo (*FALLER*[1]). A memória de cálculo de tais tempos está no (Apêndice 2):

a. Tempo gasto para consulta a *TABUS* e retorno ao *Interpretador PLTI*, em caso do programa virtual não estar com estado de pendência (nem entrada nem saída pendente):

(369 + 97) ciclos x 2,5μs = 116,5μs = 1,165 ms

b. Tempo gasto para consulta a *TABUS* e atendimento a programa virtual com entrada pendente e caractere presente na interface respectiva:

- 94 ciclos (a descontar de *a* porque desviou ao chegar no teste entrada pendente);

+497 ciclos (tempo de atendimento a entrada pendente);

+115 ciclos (somente quando houver fim de texto, isto é, recepção de (cr)).

+265 ciclos (mudança de estado do programa virtual);

783 ciclos x 2,5μs = 1957,5μs = 1,9575 ms

Tempo Total (*a* + *b*) = 1,165 + 1,9575 $\cong$ 3,2 ms

c. Tempo gasto para consulta a *TABUS* e atendimento a programa virtual com saída pendente e *buffer* de transmissão vazio:

- 67 ciclos (a descontar de *a* porque desviou ao chegar no teste saída pendente);

+328 ciclos (transmissão de caractere ≠ (Cr));

+ 90 ciclos (somente quando houver fim de texto);

+265 ciclos (mudança de estado do programa virtual);

616 ciclos x 2,5 μs = 1540 μs = 1,54 ms

Tempo Total (c + *a*) = 1,54 ms + 1,165 ms ≅ 2,7 ms

Observe-se que os tempos calculados em *b* e c englobam o tempo de *a*. Para fins de cálculo, vamos então tomar o tempo de 3,0 ms como sendo o valor de *TGPI*:

TGPI ≅ 3,0 ms (média aritmética entre *b* e c).

Para a obtenção do *TMMI*, vamos usar a tabela existente no (Apêndice 3), tabelas de tempos das micro-instruções *PLTI* e da frequência de ocorrência média das mesmas em programas *PLTI*.

Com base nesta tabela, temos que o tempo médio mais provável de uma micro-instrução *PLTI* é de 1 ms, isto com uma probabilidade de mais de 80%, pois que, em termos de frequência de existência de micro-instruções *PLTI* em um programa *PLTI* qualquer, mais de 80% das micro-instruções são de duração inferior a 1 ms, sendo portanto o tempo adotado de 1 ms para *TMMI* bastante aceitável.

Existem micro-instruções atípicas, tais como o *MOV*, com ocorrência de 0,49% em um programa *PLTI* que podem demorar quase 10 ms (um *MOV* de 200 *bytes*), mas em função da probabilidade de ocorrência muito baixa não podem ser consideradas como ocorrências relevantes.

Mesmo assim, o *SMT*, através do seu módulo *PROCESSADOR DE INTERRUPÇÕES*, testa, a cada terminal atendido, se houve sobreposição de caracteres (*overrun*) e, em caso positivo, alerta o usuário do terminal com um *bip*, além de só mostrar na tela os caracteres efetivamente recebidos (efeito de eco).

Temos então calculado o valor do *ITMT* como sendo a soma do *TGPI* com o *TMMI*:

ITMT = *TGPI* + *TMMI* =

= 3 ms + 1 ms = 4 ms

ITMT = 4 ms

Isto satisfaz plenamente à condição *ITMT* = $ITCI_{10}$, e até à condição de *ITMT* = $ITCI_{20}$: 4 ms = 5 ms ($ITCI_{20}$).

Como a influência do tempo provável de uma micro-instrução *PLTI* afeta o *ITMT* com 1 ms a cada ocorrência do mesmo *ITMT*, poderíamos então ajustar o número de micro-instrução *PLTI* a cada *interrupção* para até 6, pois:

$ITMT_5$ = 3 ms (constante TGPI) + 7 ms =

= 10 ms = 10 ms ($ITCI_{10}$) (intervalo de 7 micro-instruções *PLTI*).

Não recomendamos tal alteração, pois a folga existente de 6 ms entre o $ITMT_1$ e o $ITCI_{10}$ deve ser mantida como um fator de segurança contra programas atípicos à Tabela do (Apêndice 3), mesmo que o efeito de eco não torne a sobreposição de caracteres uma ocorrência muito grave.

Para que se tenha uma idéia da variação do desempenho de um programa que use o *SMT*, em função do número de micro-instruções *PLTI* executadas entre cada interrupção simulada, bem como da velocidade de digitação máxima aceitável para um dado número de terminais e micro-instruções *PLTI* por interrupção, mostramos abaixo uma tabela que apresenta tais informações para alguns casos. Para construção da Tabela, adotamos o tempo consumido pelo *PROCESSADOR DE INTERRUPÇÕES* como sendo o calculado no item *a*, pois é o caso mais geral:

Número de micro-instruções *PLTI* por interrupção	1	2	4	8	10
Degradação do programa, em termos de tempo (%)	≈50	≈33	≈25	≈12.5	≈10%

Número de Terminais na Rede		Velocidade de digitação máxima aceitável (car/min)				
	1	6000	3000	1500	750	600
	5	1200	600	300	150	120
	10	600	300	150	75	60

V.1.2. GASTO DE MEMÓRIA PRINCIPAL

Na DEFINIÇÃO DO SISTEMA (Capítulo II, item II.1) foi dito que a memória inicialmente ocupada pelo *SMT* era da ordem das outras rotinas de E/S.

Tomemos, então, por exemplo, para efeito comparativo, a memória ocupada pela rotina de E/S *DISCO* que trata o método de acesso a disco magnético do *TI*.

O tamanho de meória da rotina *DISCO* para acesso a uma unidade de disco, de cada vez que é chamada, é de 1250 *bytes* aproximadamente.

Calculemos agora o tamanho aproximado do *SMT* para uso com 1 programa virtual (1 terminal) apenas:

SMT propriamente dito (rotina escrita em *ASSEMBLER* que contém, além do *PROCESSADOR DE INTERRUPÇÕES*, a parte em *ASSEMBLER* que faz a comunicação *programa PLTI* e *SMT* - outros módulos em *PLTI*):

≅ 816 *bytes*;

SMTMOD (módulos do *SMT* escritos em *PLTI*):

≅3217 *bytes*;

TRATAF (rotina chamada pelo *SMT*, atualiza *TRAL* e *TAF* em uso no *NRES*):

≅ 30 *bytes*;

JUMP (rotina auxiliar usada pelo *SMT* para chamar rotina de leitura/gravação em disco da *ROM* (*DSKAUX*):

≅ 20 *bytes*;

TAF (tamanho de uma *TAF* construída pelo *CPROG* para cada terminal concatenado):

≅ 100 *bytes*;

TAMANHO TOTAL DO SMT:

Tm_{SMT} = 816 + 3217 + 30 + 20 + *n* x 100 =

Tm_{SMT} = 4182 + *n* x 100 *bytes*, onde *n* é o número de terminais concatenados.

Não foi considerado como parte do *SMT* o *Interpretador PLTI* (núcleo e rotinas), pois é o mesmo usado pelo programa *PLTI* usuário do *SMT*; nem a rotina *TDIRET*, pois, como dissemos no capítulo anterior, suas funções passaram a ser feitas pelo *CPROG*.

Assim, o tamanho da memória ocupada pelo *SMT* é aproximadamente 3,5 vezes maior que a rotina de E/S tomada como exemplo.

Dentre os diversos motivos para essa diferença, ressaltamos que cerca de 3/4 do *SMT* está escrito em *PLTI*, de um modo estruturado. É também relevante a sua maior complexidade em relação à rotina de E/S em disco, isto porque o *SMT* administra vários programas virtuais concorrentes; e o módulo *INICIALIZADOR*, apesar de ser usado somente uma vez pelo *SMT*, não pode ter seu espaço reaproveitado pelos módulos que restam e que permanecerão em uso constante, isto representando uma perda de aproximadamente 1080 *bytes*. Essa perda poderá ser eliminada no futuro, quando a evolução do *SOCO* tornar possível o *overlay* em rotinas de E/S e o *SMT* receber tal alteração.

Mesmo com o gasto de memória relativamente maior que o de rotinas de E/S convencionais, ainda ficará disponível para o programa *PLTI* de aplicação um total de 10 *Kbytes*. Sem dúvida ,

é um espaço mais do que suficiente para aplicações que usem o *SMT*, principalmente se contrabalançarmos as facilidades que estarão disponíveis. Poderemos, por exemplo, através de um programa escrito em *PLTI* (alto nível), utilizar uma rede de terminais com características diversas em termos de velocidade e formato de transmissão.

V.1.3. USO DE DISCO DE TRABALHO PELO SMT

Não foi feita nenhuma previsão inicial, por depender unicamente do tamanho da área de dados do programa de aplicação *PLTI*.

A fórmula para cálculo da área total necessária ao arquivo de trabalho do *SMT* (*SMTWRK*) é a seguinte:

$$T_{SMTWRK} = \lceil TAREA_{DADOS}/512 \rceil * (1 + n) \text{ setores,}$$

onde n é o número de terminais concatenados.

V.1.4. USO DE TERMINAIS LOCAIS x REMOTOS

Na DEFINIÇÃO DO MÓDULO *INICIALIZADOR* (Capítulo III, item *III.6.1.*) especificou-se a necessidade de se detetar a ligação do terminal com a interface respectiva no *TI* - se era via *modem* (remota) ou direta (local) - para requisição ou não de portadora.

Durante os testes do *SMT*, verificou-se que o *hardware* da interface permitia que, mesmo em ligações locais (*loop* de corrente) se requisitasse portadora e a resposta *Portadora presente* fosse obtida.

Deste modo, eliminou-se a necessidade de procedimentos diferentes para cada tipo de ligação, ficando o procedimento para ligação remota (*modem*) único para qualquer tipo de ligação,

simplificando sensivelmente a inicialização, principalmente pela ausência da necessidade de se descobrir o tipo da ligação (se remota ou local).

Concluímos, assim, que, apesar de alguns itens terem se desviado do inicialmente estimado, de um modo geral foram conseguidos resultados satisfatórios na implementação do *SMT*, podendo no futuro, com a evolução do *SOCO*, tais resultados serem aperfeiçoados.

V.2. EXTENSÕES SUGERIDAS/POSSÍVEIS

Devido ao modo de construção do *SMT*, modular, é possível se fazer extensões do mesmo, a qualquer tempo, alterando-se os módulos afetados sem que isso seja refletido nos outros módulos.

A seguir, vamos exemplificar tipos de extensões e os módulos que podem ser afetados.

V.2.1. IMPLEMENTAÇÃO DE NOVOS COMANDOS

Assim como hoje estão implementados quatro comandos para o *SMT* (*$ATIVE, $FIM, $TESTE, $FIMTESTE*), poderão ser implementados novos comandos. Essa implementação afeta exclusivamente o módulo *INTERPRETADOR DE COMANDOS*, bastando incluir a rotina que processará o novo comando no lugar adequado da estrutura *if-then-else*.

Alguns tipos de comandos sugeridos para extensões seriam:

a. Comandos de obtenção de informações, tipo *query*:

- número de usuários ligados ao programa real;
- relação de usuários e características dos terminais respectivos, ligados ao programa real;
- etc. .

b. Comandos de *message switching*:

- troca de mensagens entre terminais ligados ao programa real.

c. Outros comandos que atendam a necessidades específicas que venham a surgir.

Como alerta, apenas, lembramos que o acréscimo de novos

comandos, dependendo das funções que representem, aumentará a necessidade de memória principal do *SMT*, item que já está um pouco crítico na avaliação feita no início deste Capítulo.

V.2.2. IMPLEMENTAÇÃO DE NOVOS TERMINAIS DE E/S E/OU MUDANÇAS DE PROTOCOLO

O *SMT* foi implementado com o uso do protocolo de comunicações dos terminais EMBRACOMP disponíveis no *TI*, que é o protocolo *Teletype (TTY)*.

A mudança do protocolo ou a convivência de vários protocolos é possível sem grandes implicações, desde que se mantenha a padronização dos caracteres de controle do programa *PLTI* ou que se evolua para outro conjunto de caracteres, compatível para baixo com o existente.

Tal mudança afeta única e exclusivamente o *PROCESSADOR DE INTERRUPÇÕES*, a menos que se deseje alterar as mensagens emitidas pelo *SMT*, caso em que também os módulos *INICIALIZADOR*, *INTERPRETADOR DE COMANDOS* e *FINALIZADOR* serão afetados.

Para a simples mudança de protocolo *TTY* para outro qualquer, mantida a convenção de caracteres de controle hoje existente, a alteração afetará somente o módulo *PROCESSADOR DE INTERRUPÇÕES* nas rotinas de execução de entrada e de saída no terminal. Essas rotinas deverão passar a testar cada caractere transmitido contra os de controle (*Line-Feed*, *Carriage-Return* , *Form-Feed*) e fazer a tradução para o novo protocolo.

Caso se troque também a convenção dos caracteres de controle, além da simples troca de protocolo, lembramos novamente da necessidade *de alterar não só os módulos do SMT afetados acima referidos como todos os programas PLTI porventura já desenvolvidos, a menos que o conjunto dos caracteres hoje existente passe a ser um sub-conjunto do novo.*

Para a convivência de vários protocolos (p.ex.: *TTY*, *BSC*, *SDLC*), a alteração no *PROCESSADOR DE INTERRUPÇÕES* será mais substancial.

Deverá ser incluído em *TABUS*, para cada elemento correspondente a cada terminal da lista de concatenações, um indicador do protocolo usado pelo mesmo; existe hoje em *TABUS* uma posição (1 *byte*) *reservada para uso futuro* que poderia ser usada para tal.

A cada vez que uma operação de E/S for *de fato* ser realizada por um programa virtual, isto é, for ser transmitido o próximo caractere de uma saída pendente ou for ser lido o próximo caractere de uma entrada pendente, a rotina respectiva deverá identificar qual protocolo está associado ao terminal e fazer as traduções necessárias antes de efetivar a operação de saída ou antes de finalizar a operação de entrada.

Apesar deste tipo de extensão/alteração representar aumento de memória, existe um fato mais importante que é o da alteração do *ITMT* (*Intervalo de Tempo Máximo entre dois atendimentos consecutivos a um mesmo Terminal*) para mais, afetando diretamente a eficiência do *SMT*. Evidentemente, tais considerações aplicam-se exclusivamente ao micro-processador INTEL 8008. A evolução para qualquer outra CPU implicará em reavaliação de todos os resultados obtidos pelo *SMT*.

V.2.3. IMPLEMENTAÇÃO DE NOVAS FUNÇÕES DE E/S PARA OS TERMINAIS

Esta é a extensão mais complexa de ser feita, pois altera a Tabela de Estados dos Programas Virtuais. Cada nova função incluída será um novo estado. Serão afetados os módulos *DISPATCHER*, *INTERPRETADOR DE COMANDOS*, *EXECUTOR DE E/S* e *PROCESSADOR DE INTERRUPÇÕES*.

Embora trabalhosa, tal implementação pode ser feita. Lembramos apenas que deve ser feito um estudo custo/benefício a fim de se verificar se outras alternativas existentes não seriam melhores para resolver o problema cuja solução seja implementar uma nova função de E/S; ou, se estas não existem, se o custo da alteração terá retorno que a justifique.

V.3. ANÁLISE FINAL DO TRABALHO

Como dissemos no início deste trabalho, o *SMT* é um método de acesso a terminais que libera o programador das preocupações com a administração da rede de terminais que vá utilizar, cabendo a ele encará-la como se fosse um único arquivo lógico associado a um arquivo físico do tipo terminal vídeo, onde é possível serem lidos registros digitados e serem escritas linhas de mensagens.

Assim, em troca da enorme facilidade concedida ao programador que usa terminais, desenvolveu-se uma rotina de E/S bem mais complexa que as comuns do *SOCO*, *SISTEMA MULTI-TERMINAL*, ao qual cabe resolver os problemas da administração da rede definida pelo usuário do *TI*. Em virtude de suas características especiais em relação às outras rotinas de E/S, o *SMT* deve ser encarado como um método de acesso cuja implementação é bem mais difícil que a de uma simples rotina de E/S de uma impressora.

A partir de agora é possível, assim, desenvolver-se no *TI*, sob o *Sistema Operacional em Disco (SOCO)*, aplicações escritas em linguagem de alto nível (*PLTI*) para uma rede de terminais configurada no momento de execução dos programas de aplicação.

Doze meses foram decorridos desde a tênue idéia do que seria o trabalho de elaboração do *SMT* até a sua implementação final com a consequente disponibilidade para os usuários do *TI* (julho de 1977/julho de 1978). Tendo o trabalho sido desenvolvido basicamente por uma pessoa em tempo parcial - fora o tempo gasto pelo orientador, principalmente no início, quando chegou a ser de uma tarde por semana durante três meses - acreditamos ser um intervalo de tempo bastante aceitável para uma elaboração de tal complexidade.

Os resultados obtidos pelo *SMT*, comparados com os estimados durante a sua definição, à exceção da memória principal necessária (vide item V.1.), foram bastante satisfatórios, senão muito bons, o que demonstra o sucesso do trabalho.

Cremos ter colaborado para o desenvolvimento do PROJETO *TERMINAL INTELIGENTE* DO NCE/UFRJ com um trabalho substancial , que será bastante útil a todo usuário do *TI* no momento e no futuro.

Estaremos sempre à disposição de quem necessitar de nosso apoio, tanto com relação ao uso do *SMT* quanto à sua manutenção.

V I . B I B L I O G R A F I A

1. *FALLER, NEWTON* - Terminal Inteligente: Manual do Usuário Rio de Janeiro 1975. 79 p. (NCE/UFRJ).

2. Terminal Inteligente: Manual do Usuário do Sistema Operacional de Simulação (SOS) Rio de Janeiro 1975. 185 p. (NCE/UFRJ).

3. *VIDA CURA, JOSÉ CARLOS* - Sistema de E/S no Terminal Inteligente Rio de Janeiro 1976. 13 p. (NCE/UFRJ).

4. *CHAGAS RODRIGUES, GUILHERME* - Sistema Operacional para o Terminal Inteligente Rio de Janeiro 1976. 22 p. (NCE/UFRJ).

5. *MORAES MELLO, PAULO CESAR* - Uma Linguagem de Alto Nível para o Micro-Computador PLTI Rio de Janeiro 1976. 12 p. (NCE/UFRJ).

6. *DIJKSTRA, EDSGER W.* - A Class of Allocation Strategies Inducing Bounded Delays Only Eindhoven, Spring Joint Computer Conference 1972. pg. 933-936.

VII. APÊNDICES

VII.1. APÊNDICE 1

ALTERAÇÕES PROVISÓRIAS DO SMT PARA FUNCIONAR COM NÚCLEO NÃO RESIDENTE

SOS - V1M00 * MONTADOR * DATA 09/ 8/78

```
.XREF
.TSIM
***
*    ROTINA  DE  E/S  PARA  ACESSO A TERMINAIS - SISTEMA MULTITERMINAL
*
*                              *   S  M  T   *
*
*    O SMT TEM POR OBJETIVO MONITORAR  O USO DE PROGRAMAS VIRTUAIS
*          POR VARIOS TERMINAIS  ASSOCIADOS  A UM  PROGRAMA REAL, ONDE
*          PROGRAMA VIRTUAL E A COPIA DO PROGRAMA  REAL  EM USO POR
*          UM TERMINAL.
*
*          ********************
*          *ESTRUTURA DO SMT*
*          ********************
*           ********       *******       ************************************
*           *NUCLEO* <---> *INTES* ---> *SMTSIS ---> EXEC E/S --> INICIA *
*           ********       *******       *  ROTINA       |<-----------|    *
*             ||  ||           |         * PRINCIPAL |-->|<---------|       *
*             ||  ||           |         *     |       |DISPAT        |      *
*             ||  ||           |         *     |       |    |--> INTERP.CON. *
*             ||  ||           |<--------|<----------|                       *
*             ||  ||                     *     |       |<---|----FINALIZADOR *
*             ||  ||<-----------------------'<----------'            |       *
*             ||  |                        ***************************|*******
*             ||  |---------------------->*PRCINT                     |      *
*             ||                           *   |                       |      *
*             ||                           *   |                       |      *
*             ||<--------------------------|                           |      *
*             |                            ***************************|*******
*             |-------------------------------------------------------->|
*
*
*
*
***
*
          ENTRY SMT
          EXT   SMTMOD
TRATAF    EQU   /3F72
DECHL     EQU   /08
INCHL     EQU   /28
STEND     EQU   /10
LUZES     EQU   /30
```

```
          INDEX     EQU    /20
          LDEND     EQU    /18
          PTNUC     EQU    /79
          * SALVA REGISTRADORES USADOS PELA ROT E/S
          SMT       LAB                 *
                    PUSH                * SALVA REG B
                    LAH
                    LBL
                    LRH    REGCAL
                    LMC                 * SALVA REG C
0028                RST    INCHL        * HL = HL + 1 ;
0010                RST    STEND        * SALVA REGS DE
0028                RST    INCHL        * HL = HL + 2 ;
                    LMA                 * SALVA REG  H
0028                RST    INCHL        * HL = HL + 1;
          *
          *  ALTERA JMPINT (NA VOLTA DO SMTNOD) COLOCANDO HL + 2 NA FORMA LH
          *          EM JMPINT + 1.  ISTO E FEITO UMA VEZ E A SEGUIR O JMP ABAIXO
          *          E ALTERADO PARA LMB, POP, LBA . O CONTROLE RETORNA AO POP.
          *
          JMPALT    JMP    TABUS        *  DESVIA P/ ROTINA ALTERA JMPINT
                    CAL    TRATAF       * CHAMA TRATAF - AJUALIZ TRAL E TAF ATUAIS=NUC
```

```
***
*    PASSA PARAMETROS NO STACK E CHAMA ROTINA PLTI (SMIMOD) COM OS
*         MODULOS:
*                        -EXECUTOR  DE E/S,
*                        -DISPATCHER,
*                        -INTERPRETADOR DE COMANDOS,
*                        -FINALIZADOR,
*                        -INICIALIZADOR.
*
*        PARAMETROS:
*                        - RETMOD,
*                        - END$TABUS,
*                        - END$PRCINI,
*                        - END$FINALD,
*                        - FUNCAO.
*
*                             FUNCAO :    0 = PEDIDO NORMAL DE EXECUCAO DE
*                                             OP DE E/S.
*                                         1 = EXECUTAR MODULO FINALIZADOR.
*
*
*
*
*    A  ROTINA  RETORNA  UM CODIGO INDICATIVO DA ACAO A  SER TOMADA:
*
*          RETMOD          0 =  RETORNO NORMAL PARA A INTES
*
*                          1 =  DAR JMP P/INTES . PROG VIRTUAL QUE ESTAVA
*                               ATIVO DEU EXIT.
*
*                          2 =  REEXECUTAR OPER E/S. PROG VIRTUAL COMECANDO
*
*                      OUTROS=  CHAMAR LUZES - ERRO GRAVE
*
*****
```

```
*
*  PREPARA  CHAMADA NORMAL  DO SMTMOD
*
NORMAL   XRA              *
         LRH   FUNCAO     *
         LMA
*
*     CHAMA      SMTMOD
*
SMTCAL   XRA              * INDICA PARA A ROTINA SMTMOD -PLTI- QUE
         PUSH             *    FOI CHAMADA POR PROGRAMA ASM.
         PUSH             *
         LRH   RETMOD-1   *    CAMPO RETMOD  CONTERA  0
         LAL              *                          PARTE   BAIXA
         PUSH             *    CODIGO  DE RETORNO  DO SMTMOD
         LAH              *                          PARTE ALTA
         PUSH             *
         LRH   TABUS           * PEGA ENDERECO DE TABUS
         LAL
         PUSH                  * POE PARTE BAIXA NO STACK
         LAH                   * POE PARTE ALTA NO STACK
         PUSH
         LRH   PRCINT          * PEGA ENDERECO DO PRCINT
         LAL
         PUSH                  * POE PARTE BAIXA NO STACK
         LAH                   * POE PARTE ALTA  NO STACK
         PUSH
         LRH   FINALD          * PEGA ENDERECO DO FINALIZADOR
         LAL
         PUSH                  * POE PARTE BAIXA NO STACK
         LAH                   * POE PARTE ALTA  NO STACK
         PUSH
*
*  AS INSTRUCOES SEGUINTES, FICARAO TEMPRARIAMENTE ATE O NUCLEO DO
*                           INTERPRETADOR PLTI SE TORNAR RESIDENTE.
*
         LRH   PTNUC      *       PEGA PONTEIRO PARA NUCLEO
0018     KST   LDEND      *    OBTEM ENDERECO DO NUCLEO DO INTERPRETADOR
*
         LAL              *
         PUSH             * POE PARTE BAIXA NO STACK
         LAH              *
         PUSH             *POE PARTE ALTA NO STACK
         LRH   FUNCAO     *    PEGA FUNCAO SOLICITADA
```

```
                LAM             *
                PUSH            *     POE PARTE BAIXA NO STACK
                XRA             *
                PUSH            *     POE PARTE ALTA NO STACK
          *
          *   RESTAURA  REGISTRADORES  SALVOS   E  CHAMA  SMTMOD
          *
          *
          *    ALTERA NUCLEO DO INTERP. PLTI P/ INIBIR TRACE PLTI.
          *
                LRH     PTNUC     * OBTEM
0018            RST     LDEND     *         ENDNUC
                LAI     /15       *                +
0020            RST     INDEX     *                  21
                LAI     /A8       *    ALTERA O READ P/
                LMA               *                        XPA.
          *
                LRH     REGCAL
                LCM
0028            RST     INCHL
                LDM
0028            RST     INCHL
                LEM
0028            RST     INCHL
                LAM
0028            RST     INCHL
                LRM
                LHA
                LLB
          *
          *    TEMPORARIO ATE NUCLEO INTERP. PLTI SE TORNAR RESIDENTE. ENTAO
          *                    CAL SMPLTI SE TORNARA CAL SMTMOD.
          *
                CAL     SMPLTI    *
          *
          *    RESTAURA NUCLEO DO INTERP. PLTI PARA REINICIAR TRACE PLTI.
          *
                PUSH              *    SALVA RA.
                LRH     PTNUC     * PEGA ENDNUC + 21.
0018            RST     LDEND     *
                LAI     /15       *
0020            RST     INDEX     *
                LAI     /41       *    ALTERA O XRA PARA READ.
                POP               *    RESTAURA RA.
```

```
*         &B = CODIGO DE RETORNO DA SUB-ROTINA SMTMOD: RETMOD.
*  SE CODIGO = 0  ENTAO  DA RET PARA INTES
*
          DCB
          INB
          JFZ    NOTZE
          RET                 *  RETURN  PARA  A  INTES
*
* SE  CODIGO = 1  DA  JMP P/INTES  - PROG VIRT ATIVO ANT DEU EXIT.
*
NOTZE     DCB
*
*  A PROX INST E JTZ INTES+CALLL, GERADA NA 1A. VEZ QUE O
*                 SMTSIS E CHAMADO, ATRAVES DO CONTEUDO DOS REGS H,L .
*
JMPINT    JTZ    *-*          *     &A = STATUS
*
* SE  CODIGO = 2  REEXECUTA A OPER DE E/S.
*
          DCB
          JFZ    ERRSMT       *
          LRH    REGCAL       *   PEGA END&SAVEAREA.
          LMC                 *   SALVA RC.
0028      RST    INCHL        *   HL = HL + 1;
0010      PST    STEND        *   SALVA REGS DE.
          JMP    NORMAL       *  REEXECUTA OPERACAO DE  E/S
ERRSMT    LAI    /A4          *  SE CODIGO <> 0,1, OU 2  ENTAO ERRO GRAVE
0030      RST    LUZES        *
          LAB                 *
          ADI    /02          *   MOSTRA CODIGO DE ERRO.
0030      RST    LUZES        *
          HLT                 *
```

```
*
*     TEMPORARIO ATE NUCLEO PLTI SE TORNAR RESIDENTE.
*
*          CHAMA SMTMOD  (ROTINA EM PLTI).
*
SMPLTI    LRH    SMTMOD     *
          LAI    /0A        *   PULA INICIO DO PROGRAMA PLTI.
3020      RST    INDEX      *
JMPNUC    JMP    *-*        * JMP P/ NUCLEO INT. PLTI CONSTRUIDO NA
*                                 PRIMEIRA VEZ QUE O SMT E CHAMADO.
```

```
*
*  FINALIZADOR  -  ROTINA ASM PREPARA E EXECUTA CHAMADA SMTMOD.
*
FINALD    LAI    /01
          LRH    FUNCAO
          LMA
          JMP    SMTCAL
*
```

```
         *
         * DEFINICAO DAS CONSTANTES  E  AREAS  *  PRCINT  E  SMTSIS  *
         *
0007     REGCAL   DS    7
         FUNCAO   DC    H'0'
         TABEST   DC    (/04,/13) * INATIVOEP --> INATIVOFE

                  DC    (/13,/00) * INATIVOFE

                  DC    (/21,/09) * ATIVOOK

                  DC    (/38,/21) * ATIVOSP   --> ATIVOOK

                  DC    (/44,/53) * ATIVOEP   --> ATIVOFE

                  DC    (/53,/00) * ATIVOFE

                  DC    (/64,/73) * INATIVOTEP--> INATIVOTFE

                  DC    (/73,/00) * INATIVOTFE

                  DC    (/88,/64) * INATIVOTSP--> INATIVOTEP

                  DC    (/98,/04) * INATIVOSP --> INATIVOEP

                  DC    (/A8,/21) * INATIVOMP --> ATIVOOK

                  DC    /20       * FIM DA TABEST
         *
         *
0007     SALVAR   DS    7
0001     RETMOD   DS    1
0001     CHARLI   DS    1
```

VII.2. APÊNDICE 2

CÁLCULO DOS TEMPOS APROXIMADOS DE CADA FUNÇÃO DO PROCESSADOR DE INTERRUPÇÕES

SOS - V1M06 * MONTADOR * DATA 09/08/78

```
                *
                *   PROCESSADOR  DE  INTERRUPCOES  -  SISTEMA  MULTITERMINAL  -  SMT  -
                *
                PRCINT  EQU   *
0003            SAVNUC  DS    3                    CONSULTA A "TABUS"
                *
                *   SALVA REGISTRADORES  USADOS
                *
                      6 PUSH              *  SALVA REG. A
                      5 LAB
                      6 PUSH              *  SALVA REG. B
                      5 LAC
                      6 PUSH              *  SALVA REG. C
                      5 LAD               *
                      6 PUSH              *  SALVA REG. D
                      5 LAE               *
                      6 PUSH              *  SALVA REG. E
                      5 LAH               *
                      6 PUSH              *  SALVA REG. H
                      5 LAL               *
                      6 PUSH              *  SALVA REG. L
                ***
                *   FORMATO DE TABUS :
                *
                *    DCL 1 ELEM$TABUS BASED &HL,
                *
                *   DESLOC                          NOME DO CAMPO
                *
                *     /00                        2 END$FIS$LINK$TABUS          ADDR,
                *     /02                        2 PC$PROG$VIRTUAL,            ADDR,
                *     /04                        2 PT$AREA$DADOS$DISCO         ADDR,
                *     /06                        2 RETORNO$OP$ES               BYTE,
                *     /07                        2 PT$TAF$PROG$VIRTUAL         ADDR,
                *     /09                        2 END$AREA$ENT$SAI            ADDR,
                *     /0B                        2 ESTADO$PROG$VIRTUAL         BYTE,
                *     /0C                        2 PT$PROX$ELEM$TABUS          ADDR;
                *
                ***
                      8+8 LRH   ATABUS    *
0018                  40  RST   LDEND
                      8   LAI   /0C       *  PEGA ELEM DE TABUS SEGUINTE AO
0020                  27  RST   INDEX      *    AO  ULTIMO  PROCESSADO
0018                  40  RST   LDEND      *
                BUSCA 5+5 MDH             *
```

503 - V1M00 * MONTADUR * DATA 09/08/78

```
                8  LAI   /0B        *  BUSCA
0020           27  RST   INDEX
                8  LAM              *        PROG. VIRTUAL
                8  NDI   /04        * (BIT 2 LIGADO)
               11  JFZ   ENTPED     *                  COM ENTRADA OU
                8  LAM
                8  NDI   /08        * (BIT 3 LIGADO)          SAIDA PENDENTE.
          94   11  JFZ   SAIPED
              818  LRH   ATABUS     *
0010          40  RST   STEND      *
      369  67   11  JMP   VOLTA      *
           *
           * PROCESSA OPERACAO DE E/S
           *
ENTPED      8  LBI   /01                    ENTRADA PENDENTE.
           11  JMP   COMUM
           *
SAIPED      8  LBI   /02                    SAIDA PENDENTE.
           *
COMUM     818  LRH   ATABUS
0010       40  RST   STEND      *  GUARDA END DO ELEM TABUS ATENDIDO
          515  MHD
                8  LAM              * PEGA ENDERECO LINK
0000  SELEC    6  SEL   0
                8  STATUS                  COMUM A ENTRADA E SAIDA PENDENTE.
                6  PUSH             *  SALVA STATUS NO STACK
                5  DCB
               11  JFZ   SAIDA      * ROTINA  SAIDA PENDENTE
               11  JMP   ENTRA      * ROTINA  ENTRADA PENDENTE
           *
           *
           *
           * ROTINA PROCESSA ENTRADA PENDENTE
           *
ENTRA       8  POP                  *
            5  LBA                  *          SALVA STATUS EM &B
            8  NDI   /0E
               11  JTZ   SEMERR
           *
           * ERRO NA INTERFACE - AVISA OPERADOR DO TERMINAL C/ BIP.
           *
                  MHD               *     &HL   =   ENDLNK
                  LAM               *    SELECIONA O LINK.
0000              SEL   0           *
```

SOS - V1.00 * MONTADOR * DATA 09/08/78

ENTRADA PENDENTE

```
0004                CNTR  4          *   RESET DO LINK.
                    LAI   /07        *
                    WRITE            *   DA  BIP NO TERMINAL.
                    STATUS           *
                    LBA              *   CONTINUA A EXECUTAR A OPER. DE ENTRADA.
                *
            SEMERR  LAB              *  PEGA  STATUS SALVO EM &B
                    NDI   /01        * TESTA SE  CHAR  PRESENTE
                    JTZ   VOLTA
                    READ             * SIM, LE.
                *
                *   ROTINA PARA FAZER APAGAMENTO DE CARACTERES JA DIGITADOS (RUB/OUT).
                *
                    CPI   /7F        *   TEST1 SE PEDIDO DE APAGAMENTO.
                    JNE   ECCO       *   NAO, DESVIA PARA DAR ECO.
                    MHD              *
                    LAI   /07        *
0020                RST   INDEX      *
0018                RST   LDEND      *
                    LAI   /0A        *
0020                RST   INDEX      *
                    LAM              *
                    CPI   /00        *   TESTA SE EXISTE TEXTO P/ APGAR.
                    JE    VOLTA      *   NAO, RETORNA AO NUCLEO DO INT. PLTI.
                    LAI   /5C        *
                    WRITE            *
            ESP01   STATUS           *
                    NDI   /80        *
                    JTZ   ESP01      *
                    LAM              *
                    SUI   /01        *
                    LMA              *
                    ADI   /01        *
0020                RST   INDEX      *
                    LAM              *
                    WRITE            *   MOSTRA CARACTERE APAGADO.
            ESP02   STATUS           *      (ENTRE BARRAS INVERTIDAS)
                    NDI   /80        *
                    JTZ   ESP02      *
                    LAI   /5C        *
                    WRITE            *
                    JMP   VOLTA      *   APAGAMENTO REALIZADO, VOLTA AO NUCLEO .
            ECCO    WRITE            * FAZ ECO DO CARACTER LIDO.
                    LRH   CHARLI
```

```
                    7  LMA                 * SALVA CHAR-LIDO.
                  575  MHD                 *  PEGA  END$TABUS
                    8  LAI     /07
0020                27  RST     INDEX       *  OBTEM END$TAF
                    8  LBM
0028                15  RST     INCHL
                    8  LCM
                  575  MHB                 *  POE  END$TAF  EM &HL
                    8  LAI     /0A
0020                27  RST     INDEX       *  PEG  PT$ULT$BYTE  NA TAF
                    8  LAM
                    8  ADI     /01         *  INC  DE  1.
                    8  CPI     /51         *   TESTA SE REG > 80
                   11  JI      TAMOK       *   NAO, COLOCA CHAR LIDO DA TAF
                   11  JMP     IGNORA      * SIM, IGNORA ULTIMO CHAR COLOCADO NA TAF
            *
TAMOK       9  LMA
            *
0020  IGNORA 27  RST     INDEX       *  OBTEM POS. DO  CHAR  NA AREA$RECEP$TRANSM
                  575  MBH                 *
                  818  LRH     CHARLI      *  PEGA  CHAR  LIDO.
                    8  LAM                 *
                  575  MHB                 *
                    9  LMA                 *  POE CHAR-LIDO NA TAF.
          497       8  CPI     /0D         *   COMPARA COM O CR
                   11  JFZ     VOLTA       *  SE  NAO CR VOLTA INTERPRETADOR.
ACABIO      8  STATUS                      * VERIFICA SE PRONTO P/ TRANSMITIR.
                    8  NDI     /80         *
                   11  JTZ     ACABIO      *   SE NAO, ESPERA I/O ACABAR.
                    8  LAI     /0A         *  DEVOLVE LINE-FEED COMO RESPOSTA DE
                    6  WRITE               *          DE TEXTO LIDO OK
                  575  MHD                 *     &HL   =  PT$TABUS
                    8  LAI     /0B         *  PEGA ESTADO DO PROG VIRTUAL
0020                27  RST     INDEX       *
                  575  MDH                 *  SALVA  $HL EM $DE
                    8  LBM                 *  VAI PARA ROTINA COMUM DE MUDAR ESTADO DO
          115       11  JMP     MUDEST      *  PROG VIRTUAL  FIM DE TEXTO (ENTRADA)
            *
```

```
                *
                * ROTINA PROCESSA SAIDA PENDENTE
                *
        SAIDA    8 POP              *  PEGA STATUS SALVO NA STACK
                 8 NDI    /80       * JA ACABOU ?
                11 JTZ    VOLTA     * SE NAO VOLTA AO INTERPRETADOR.
                 6 STATI             *
                 8 NDI    /60       * TESTA SE LINK OK;
                11 JFZ    FISOK     * SE SIM, TRANSMITE CARACTERE.
                *
                *  MODEM  OU  LOOP NAO LIGADOS  - ERRO NA INTERFACE
                *
                   LAI    /23       *   INDICA NaS LUZES LINK C/ ERRO.
0030               RST    LUZES     *
                   JMP    VOLTA
                *
                *
                *
        FISOK  515 MHD               *  PEGA   END$TABUS
                 8 LAI    /07
0020            11 RST    INDEX     *  OBTEM  END$TAF
                 8 LBM
0028            15 RST    INCHL
                 8 LCM
                515 MHB              *  POE END$TAF EM  $HL
                 8 LAI    /0A       *    PEGA PT$ULT$BYTE
0020            11 RST    INDEX     *
                 8 LAM               *    INC  DE  1
                 8 ADI    /01       *
                 9 LMA               *    GUARDA NA  TAF
0020            11 RST    INDEX     *    FAZ  &HL = NOVO CARACTER A TRANSMITIR
                 8 LAM               *
                 6 WRITE             *    TRANSMITE CHARACTER
                 8 CPI    /0D       *   TESTA SE E CR.
       328      11 JFZ    VOLTA     *    SE NAO CR  VOLTA  AO INTERPRETADOR
0028            15 RST    INCHL     * VERIFICA SE PROX CHAR TBEM E CR
                 8 LAM               *   Fim-de-Texto (SAIDA)
                 8 CPI    /0D       *  TESTA SE CARACTER SEGUINTE TBEM E CR.
                11 JFZ    VOLTA     *   SE NAO EXISTEM 2 CR'S SEGUIDOS NAO E EOT
                515 MHD              *  OBTEM NO &HL PT$PROX$ELEM$TABUS
                 8 LAI    /0B       *    PEGA  ESTADO DO PROG$VIRTUAL EM TABUS
0020            11 RST    INDEX     *
                 9 LBM               *
       90       515 MDH             *    GUARDA EM &DE END$ESTADO$PROG$VIRTUAL
```

```
                *
                *  ROTINA  OBTEM  PROXIMO ESTADO  DO PROGRAMA VIRTUAL
                *
MUDEST  8,18    LRH     TABEST
        8       LCI     /20     __ MUDANCA DE ESTADO DO PROGRAMA __
PEGAET  8       LAM                    VIRTUAL
        5       CPB
        11      JE      ACHOU
        8       LAI     /02
0028    15      RST     INCHL
        8       LAM
        8       XRI     /20
        11      JFZ     PEGAET
        8       LAM             *   ERRO     GRAVE
        8       ORI     /80     *          ESTADO
0030    13      RST     LUZES   *                  INEXISTENTE
                HLT             * ERRO DO SMT - COLOCOU PROG EM ESTADO INEX.
                *
                *  OBTEM  ESTADO  SEGUINTE
                *
0028    ACHOU   15      RST     INCHL   *
        8       LAM             * OBTEM ESTADO DA TABELA
        5,15    MHD             * OBTEM END$ESTADO$PROG$VIRTUAL
        9       LMA             * MUDA  ESTADO$PROG$VIRTUAL
        5       LAL             *   SUBTRAI 4 DE HL.
        8       SUI     /04     *
        5       LLA             *
        5       LAH             *
        8       SBI     /00     *
        5       LHA             *
0018    40      RST     LDEND   *  PEGA END$TAF$PROG$VIRTUAL
        8       LAI     /0A     *       PEGA END$PT$ULT$BYTE
0020    11      RST     INDEX   *
        5       XRA             *
        9       LMA             *   COLOCA ZEROS EM PT$ULT$BYTE
        *265
```

```
*
*  SAIDA GERAL DO PROCESSADOR DE INTERRUPCOES - SMT
*
VOLTA   8   POP             *
        5   LLA             *  RESTAURA REG L
        8   POP             *
        5   LHA             *  RESTAURA REG H
        8   POP             *
        5   LEA             *  RESTAURA REG E
        8   POP             *
        5   LDA             *  RESTAURA REG D
        8   POP             *
        5   LCA             *  RESTAURA REG C
        8   POP             *
        5   LBA             *  RESTAURA REG B
        8   POP             *  RESTAURA REG A
JMPNC3 11   JMP   *-*       *  RETORNA AO INTERPRETADOR.
* 97
*  O JMP ACIMA E CONSTRUIDO EM ROTINA EXISTENTE NO FIM DO PROGRAMA
*        E QUE E CHAMADA NA 1A. VEZ QUE O SMT E CHAMADO.
*        (TEMPORARIO ATE O NUCLEO DO INT. PLTI SE TORNAR RESIDENTE).
*
```

505 - V1N00 * MONTADOR * DATA 09/-8/78

```
                ENTRY  INFACE
        INFACE  POP
                POP             * VELOCIDADE
                LDA             *
                POP
                POP
                LEA             * FORMATO
                POP
                POP             *  POE NO REG A O ENDLNK
                LBA             *
0000    SELECT  SEL    0
                STAT1            *    PEGA STATUS - 2A. PALAVRA
                NDI    /01       *   TESTA SE INTERFACE LIGADA
                JFZ    LIGADO    * SE SIM, VAI PROSSEGUIR INICIALIZACAO
                LAI    /A2       *
0030            RST    LUZES     * SE NAO, INDICA NAS LUZES INTERFACE NAO
                LAB              *                                  CONECTADO
                ORI    /80       *    MOSTRA ENDLNK C/ ERRO.
0030            RST    LUZES     *
                LAB              *
                JMP    SELECT    *  VOLTA A TESTAR INTERFACE
0004    LIGADO  CNTR   4         *
                LAE
0002            CNTR   2         * POE NA INTERFACE O FORMATO DA TRANSMISSAO
                LAD
0003            CNTR   3         * POE NA INTERFACE A VELOCIDADE DA TRANSMISSAO
                LAE
        *
        * TRANSMISSAO POR MODEM  -  ESPERAR QUE BITS DO  STATUS FIQUE = 1.
        *
0005    PEDMOD  CNTR   5
                LCI    /64       *  ESPERA ATE 100 VEZES A PORTADORA SER OK.
        VESTAT  STAT1            *  TESTA  SE  BITS = 1
                LRA
                NDI    /20       *  SE SIM MODEM OK -
                RFZ              *
                DCC              * SE MODEM NAO OK TESTA SE SAO 100 VEZES
                JFZ    VESTAT    * SE NAO, VOLTA A TESTAR MODEM.
                LAI    /A3       *  SE JA TESTOU 100 VEZES E NAO OK
0030            RST    LUZES     *     MOSTRA NAS LUZES ERRO DE PERIFERICO
                LAB              *
0000            SEL    0         *     SELECIONA LINK OUTRA VEZ
                JMP    PEDMOD    *  VOLTA A TESTAR MODEM.
```

```
*    TABELA   DE   USUARIOS   -   TABUS   -
ATABUS   DC    D(TABUS)
TABUS    EQU   *          *
         LMB              *   GUARDA REG L QUE NAO HAVIA SIDO SALVO.
         INB              *   SOMA 1   EM  L (&B)
         JFC   SEGUE      *   SE NAO VAI 1,ENTAO NAO SOMA 1 EM H (&A).
         ADI   /01        *   SE VAI 1 ,SOMA 1 EM H (&A).
SEGUE    LRH   JMPINT+1   *    POE  EM  JMPINT
         LMB              *     &B  =  &L   END DO RETORNO P/ INTES
         LRH   JMPINT+2   *
         LMA              *   &A = &H          NO FORMATO LH
*
*      TEMPORARIO :  ATE NUCLEO DO INT. PLTI SE TORNAR RESIDENTE.
*
         LRH   PTNUC      *
0018     RST   LDEND      *   CONSTROI JMP PARA NUCLEO
         LAH              *              DO INTERPRETADOR PLTI.
         LBL              *
         LRH   JMPNUC+1   *
         LMB              *
0028     RST   INCHL      *
         LMA              *
         LHA              *   PEGA END$NUCLEO + 3.
         LLB              *
         LAI   /03        *
0020     RST   INDEX      *
         LAH              *
         LBL              *
         LRH   JMPNC3+1   *
         LMB              *   CRIA JMP P/ NUCLEO + 3.
0028     RST   INCHL      *
         LMA              *
*
*
*
         LRH   JMPALT     *  RECONSTROI INICIO DO SNTSIS :
         LAI   /F9        *     LMB
         LMA              *
         LRH   JMPALT+1   *
         LAI   /4D        *    POP
         LMA              *
         LRH   JMPALT+2   *
         LAI   /C8        *          LBA
         LMA              *
```

```
SOS - V1M00                    * MONTADOR *                    DATA 09/08/78


                    JMP    JMPALT+1   *   VOLTA PARA EXECUTAR A PARTIR DE POP.
    0092            DS     146
                    DC     *-*
          *

                    END                *              FIM    DO   SMTSIS

BYTES     MONTAGEM BEM SUCEDIDA      0  ADVERTENCIA(S)     0   ERROS(S)

                                       OPCOES EM EFEITO   XREF   LIST   TSIM

S  CRUZADAS

 0598

 0251

 0059 0167 0169 0171 0173 0217 0407 0463 0476 0494 0505 0619 0619
 0234 0311 0315 0375 0378 0391 0405 0411 0420 0435 0461 0467 0471 0482 0517 0614

 0630 0633


 0187 0309 0312 0376 0515 0664

 0454 0500 0560 0563 0584
```

VII.3. APÊNDICE 3

TABELA DE TEMPOS E FREQUÊNCIA DAS MICRO-INSTRUÇÕES PLTI E TEMPO MÉDIO PROVÁVEL DE UMA MICRO-INSTRUÇÃO PLTI

ORDEM	CÓDIGO (HEX)	MNEMÔNICO	Nº DE CICLOS	Nº DE CICLOS COM NÚCLEO	TEMPO APROX. EM μS (TI)	FREQUÊNCIA (USO) (TI)	FREQUÊNCIA ACUMULADA
1	02	LIA	78	286	715,0	18,77	18,77
2	01	LIB	55	263	657,5	17,30	35,07
3	23	GOT	56	264	660,0	9,17	44,24
4	04	LA	159	367	917,5	8,41	52,65
5	03	LB	136	344	860,0	6,56	59,21
6	08	SA	115	323	807,5	5,46	64,67
7	22	IF	69/83	277/291	692,5/727,5	5,02	69,69
8	07	SB	77	285	712,5	4,82	74,51
9	0F	ADD	121	329	822,5	3,39	77,90
10	18	EQ	113	321	802,5	2,50	80,40
11	10	SUB	193	401	1002,5	1,76	82,16
12	3E	ADDB	181	389	972,5	1,62	83,78
13	17	AND	81	289	722,5	1,05	84,83
14	1B	GT	111	319	797,5	0,98	85,81
15	0C	INDA	227	435	1087,5	0,90	86,71
16	19	NE	105	313	782,5	0,87	87,58
17	35	RET-PLTI	47	255	637,5	0,84	88,42
18	35	RET-ASM	267	475	1187,5	0,84	89,26
19	33	ENT	402+B1*102+	610+...	1525+B1*255+ +B2*542,5	0,82	90,08
20	3C	LXSA	352	560	1400,0	0,78	90,86
21	3D	LAS	306	514	1285,0	0,78	91,64
22	05	LXB	182	390	975,0	0,74	92,38
23	0B	INDB	144	352	880,0	0,70	93,08
24	15	OR	81	289	722,5	0,69	93,77
25	29	LIAC	92	300	750,0	0,57	94,34
26	24	MOV	161+194*Tam	369+...	922,5+(485* *Tam)	0,49	94,83
27	3B	LXSB	309	517	1292,5	0,44	95,27
28	21	END	181	389	972,5	0,40	95,67
29	1A	LT	101	309	772,5	0,38	96,05
30	06	LXA	197	405	1012,5	0,31	96,36
31	32	ASM	847	1055	2637,5	0,25	96,61
32	26	EXIT	-	-	-	0,24	96,85
33	14	NOT	65	273	682,5	0,24	97,09
34	20	DOTA	369	577	1442,5	0,23	97,32
35	09	SEB	101	309	772,5	0,18	97,50
36	28	SLL	97+65*n	305+...	762,5+162,5n	0,18	97,68
37	27	SLR	"	"	"	0,15	97,83
38	11	MULD	4892	5100	12750,0	0,14	97,97
39	0E	NOP	11	219	547,5	0,13	98,10
40	12	DIVD	10824	11032	27580,0	0,11	98,21
41	1E	MINU	72	280	700,0	0,11	98,32
42	1F	DOTB	347	555	1387,5	0,11	98,43
43	0A	SEA	141	349	872,5	0,10	98,53
44	25	CASE	427	635	1587,5	0,09	98,62
45	1C	LE	111	319	797,5	0,08	98,70
46	13	MOD	10824	11032	27580,0	0,08	98,78

ORDEM	CÓDIGO (HEX)	MNEMÔNICO	Nº DE CICLOS	Nº DE CICLOS COM NÚCLEO	TEMPO APROX. EM μS (TI)	FREQUÊNCIA (USO) (FI)	FREQUÊNCIA ACUMULADA
47	0D	GOTC	90+Bl*49	298+...	745+Bl*122,5	0,08	98,86
48	38	SR	420+Bl*552	628+...	1570+Bl*1380	0,05	98,91
49	1D	GE	106	314	785,0	0,04	98,95
50	36	RETV	138	346	865,0	0,02	98,97
51	16	XOR	81	289	722,5	0,02	98,99
52	2B	READ	499+E/S	707+E/S	1767,5+E/S	-	
53	2C	WRITE	"	"	"	-	
54	2D	REWR	"	"	"	-	
55	2E	SEEK	"	"	"	-	
56	2F	REWD	489+E/S	697+E/S	1742,5+E/S	-	
57	30	ENDF	"	"	"	-	
58	31	INV	-	-	-	-	
59	34	ABO	56	264	660,0	-	
60	37	NOP	11	219	547,5	-	
61	39	SET	172	380	950,0	-	
62	3A	RSET	232	440	1100,0	-	

a. Núcleo do *Interpretador PLTI* gasta 208 ciclos de processador por micro-instrução.

b. Ciclo do INTEL 8008 = 2,5μs.

TEMPO MÉDIO PROVÁVEL DE UMA MICRO-INSTRUÇÃO *PLTI*:

$$\frac{\sum_{i=1}^{51} TI_i \times FI_i}{100} \cong 895,98\mu s$$

Para montagem da Tabela acima, pelo Prof. GUILHERME CHAGAS RODRIGUES, foram adotadas as seguintes premissas:

1. ADD, SUB, MUL, DIV, MOD - Sem *overflow*.
2. ADD, SUB - *Carry* ⊕ sinal soma = 1 com 50% probabilidade.
3. EQ, NE, LT, GT, LE, GE - 50% probabilidade *FALSE*.
4. DOTB, DOTA - Considerado o caso incremento > 0, não escape.
   Tempos da INTDO:
   0 com escape = 174 + 8 Pux + 3 Po
   0 sem escape = 177 + 4 Pux + 3 Po + 2 Pox
   0 com escape = 190 + 8 Puse + 3 Po
   0 sem escape = 187 + 4 Pux + 3 Po + 2 Pox
5. DIVD - Resultado com 50% *bits* "1".
6. DIVD - Se quociente < 0,+ 50; se resto < 0,+ 94; se divisor < 0,+ 44.
7. MULD - Multiplicando com 50% *bits* "1".
8. MULD - Despreza *overflow*; se resultado < 0,+ 81; cada fator < 0 + 44.
9. MOV - Despreza tamanho = 0.
10. CASE - Despreza índice = 0, < 0, > tamanho.
11. SLR, SLL - Despreza deslocamento < 0.
12. INTES - *STATUS* sempre = 0.
13. SR - 50% parâmetros *byte* (diferença = 5 para cada *address*.)
14. SET - Sem esperar validade, RSET sem erro.

VII.4. APÊNDICE 4

PARTES DO SISTEMA OPERACIONAL EM DISCO DO TI (SOCO) ALTERADAS/UTILIZADAS PELO SMT

a. NÚCLEO RESIDENTE

Pos 35-37 Hex - Desvio para *entry-point* do programa: utilizada pelo módulo *INICIALIZADOR* para obter endereço da área de dados e seu tamanho.

Pos 38-55 Hex - Rotina de *EXIT*: alterada pelo *INICIALIZADOR* para desviar para o módulo *FINALIZADOR* quando chamada; restaurada pelo módulo *FINALIZADOR*.

Pos 70-73 Hex - Endereço da *TRAL* e *TAF* em uso: atualizadas pela rotina *TRATAF* a cada vez que o *SMT* é chamado; utilizadas pelo *INICIALIZADOR* para acessar *TRAL* e *TAF* do arquivo que usa o *SMT*.

Pos 74-75 Hex - Endereço da *savearea* do *Interpretador PLTI*: utilizada pelo *EXECUTOR DE E/S*, *INICIALIZADOR* e *DISPATCHER* para obter PC da última operação de E/S, REGs C e DE , etc. .

Pos 78-79 Hex - Endereço do núcleo do *Interpretador PLTI*: utilizado pelo *INICIALIZADOR* para alterar o núcleo de modo a ativar o *PROCESSADOR DE INTERRUPÇÕES*, colocando instrução *JMP* para o mesmo, restaurada ao final do *SMT* pelo *FINALIZADOR*; utilizado ainda pela parte *ASSEMBLER* do *SMT* para inibir o *TRACE PLTI* da parte do *SMT* em *PLTI*.

b. NÚCLEO DO INTERPRETADOR PLTI

Alterado pela parte do *SMT* em *ASSEMBLER* intermediária entre a micro-rotina *INTES* e a parte do *SMT* em *PLTI*, para não dar *TRACE* desta rotina *PLTI*. Isto é feito pela alteração da instrução *ASSEMBLER READ* (do núcleo) para *XRA* na chamada do *SMTMOD* (rotina *PLTI*) e vice-versa no retorno.

As três primeiras posições são copiadas pelo *INICIALIZADOR* para o *PROCESSADOR DE INTERRUPÇÕES* nas posições correspondentes, e nelas é construído um *JMP* (Hex 44) para o *PROCESSADOR DE INTERRUPÇÕES*. Não é restaurada esta parte, pois, ao fim

do *SMT*, é destruída pelo *CPROG*.

c. *INTERPRETADOR PLTI*

Micro-rotina *INTES* - O *SMT* utiliza o conteúdo do par de Registradores HL na primeira vez que recebe o controle da micro-rotina *INTES*, visto o mesmo conter o endereço de retorno - 1 - para a referida micro-rotina. Este endereço é usado para construir um *JMP* para retorno à *INTES* em caso de não haver *CAL* pendente, como quando um programa virtual termina, pois neste caso o *SMT* recebe controle via *EXIT* e não via *CAL* da micro - -rotina.