UMA MAQUINA BASICA EDISON

Nelson Quilula Vasconcelos

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

Aprovada por:

Prof. Edil Severiano Tavares Fernandes

(presidente)

Prof. Cláudio Amorim

Prof. José Lucas Rangel Neto

Prof. Michael A. Stanton

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 1984

VASCONCELOS, NELSON QUILULA

Uma Máquina Básica Edison

(Rio de Janeiro) 1984.

viii, 240 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas, 1984)

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Sistemas Operacionais I. COPPE/UFRJ II. Título (série)

AGRADECIMENTOS

- Ao Professor Edil Severiano Tavares Fernandes, pela dedicação e apoio que tornaram possível o desenvolvimento deste trabalho.

- Aos demais membros da banca que muito me honraram com sua participação.

- Ao Professor Jean-Michel Nayrac que muito me ajudou a conhecer o computador MITRA-15.

- Aos demais Professores do programa de Engenharia de Sistemas da COPPE que com idéias e esclarecimentos muito contribuiram neste trabalho.

- Aos Funcionários do Programa de Engenharia de Sistemas da COPPE, em especial a Felipe Maia Galvão França, responsável pela solução de inúmeros problemas surgidos durante a realização das alteraçöes no computador MITRA-15.

- Aos professores do programa de Engenharia de Transportes, que viabilizaram o início das minhas atividades na COPPE, em especial os Professores Félix Antoine Mora-Camino e César das Neves, meus primeiros orientadores.

- Aos funcionários da CII - Thomson que de todas as formas procuraram colaborar, fornecendo as mais diversas informações que se fizeram necessárias para a realização das modificaçöes no computador MITRA-15.

. Aos colegas da COPPE que, estudando, discutindo e trabalhando, me forneceram muitas inspirações e idéias, em especial a Luiz Carlos Zancanella, que construiu o nosso compilador EDISON e uma das implementações de nossa máquina básica.

. Aos colegas do CEPEL pelo incentivo e apoio.

. A MÔnica Dias da Cunha pela ajuda e compreensão.

. Ao meu pai, que contribui de muitas formas na realização deste trabalho.

. A CAPES e ao CNPQ pelo suporte financeiro ao projeto no qual este trabalho está inserido.

Resumo da Tese Apresentada à COPPE/UFRJ como parte dos requisitos necessários à obtenção do grau de Mestre em Ciências (M.Sc.)

UMA MAQUINA BASICA EDISON

Nelson Quilula Vasconcelos

Novembro de 1984

Orientador: Edil Severiano Tavares Fernandes

Programa: Engenharia de Sistemas

Neste trabalho é apresentada uma máquina básica EDISON, composta por um núcleo e um interpretador. O interpretador implementa uma "máquina de pilha" cujo conjunto de instruções é também apresentado. O núcleo permite a realização das funções associadas à criação, destruição e sincronização de processos previstas na linguagem EDISON, bem como as atividades relacionadas ao escalonamento dos processos, além de realizar a inicialização da execução de um programa EDISON.

São ainda apresentadas algumas implementações desta máquina básica desenvolvidas em alguns dos equipamentos disponíveis no laboratório do Programa de Engenharia de Sistemas.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for degree of Master of Science (M.Sc.)

AN EDISON BASIC MACHINE

Nelson Quilula Vasconcelos

November, 1984

Chairman: Edil Severiano Tavares Fernandes

Department: Engenharia de Sistemas

This work describes an EDISON virtual machine that was structured in two great parts: The nucleus and the interpreter. The interpreter implements a stack machine whose instructions set is also presented. The nucleus carries out the tasks related both to the EDISON program initialization and to the process creation, destruction, synchronization and scheduling, necessary to run EDISON programs.

Some implementations of the virtual machine above carried out in the machines available in the laboratory of the "PROGRAMA DE ENGENHARIA DE SISTEMAS DA COPPE/UFRJ" are also shown.

INDICE

I. INTRODUÇAO

Os programas de computador mais freqüentemente encontrados descrevem, de uma forma seqüencial, os diversos passos elementares que a máquina deve seguir afim de realizar uma tarefa global. Em certas aplicações, porém, é necessário utilizar programas que descrevam um conjunto de tarefas (processos) que devem ser executados de forma paralela (i.e. simultaneamente) ou intercaladamente.

A execução de tais tarefas requer, via de regra, que certas informações sejam utilizadas por mais de uma delas. Assim é necessário dispor de mecanismos que permitam a comunicação entre as tarefas, possibilitando que elas cooperem na realização da função global especificada pelo programa.

E necessário, ainda, prever formas de sincronização entre as tarefas, uma vez que a execução de uma tarefa pode atingir um ponto (denominado ponto de sincronização) no qual o prosseguimento de sua execução requer que tenha havido, previamente, a execução de certas partes de outras tarefas.

Finalmente, deve-se considerar que a execução simultânea de diversas tarefas pode acarretar em competição por parte destas por recursos físicos (processadores, periféricos, etc) ou lógicos (seções críticas que envolvem o uso de tabelas globais, por exemplo).

Tradicionalmente, as aplicações que utilizam tarefas executadas simultaneamente englobam os sistemas operacionais e certas aplicações em tempo real. Atualmente, porém, considerando o custo decrescente do "hardware" e a crescente complexidade dos problemas a serem resolvidos utilizando métodos computacionais, tem-se tornado cada

vez mais conveniente descrever outros tipos de problemas sob a forma de tarefas executadas em paralelo, permitindo, assim, a solução de problemas cuja resolução seqüencial acarretaria em um tempo de resposta muito grande, que pode ser intolerável.

A linguagem EDISON, definida por Per Brinch Hansen (1), é particularmente conveniente para a descrição de tarefas paralelas (programação concorrente) porque, apesar de ser uma linguagem simples e compacta, dispõe de construções que facilitam a estruturação dos programas (utiliza o conceito de blocos e dispõe de variáveis estruturadas) e de poderosas e flexíveis ferramentas para a codificação de programas concorrentes que podem ser executados em ambientes que disponham de um ou mais processadores.

Um dos projetos do programa de Engenharia de Sistemas da COPPE/UFRJ consiste na implementação de um sistema EDISON. Para tanto foram definidas as seguintes atividades:

- Construção de um compilador EDISON
  (que gera um código intermediário);
- Construção de uma máquina básica EDISON
  (que deve interpretar o código supra-citado);
- Construção de um ambiente de programação EDISON
  (para facilitar a elaboração e execução de programas codificados em EDISON).

A construção de um primeiro compilador EDISON, codificado em ALGOL, foi um dos resultados de uma tese de mestrado desenvolvida por Zancanella (2), enquanto que neste trabalho é apresentada uma máquina básica EDISON e algumas implementações desta em diversos níveis.

II. A MAQUINA BASICA EDISON

II.1. CONSIDERAÇOES GERAIS

A idéia de construir uma máquina básica para uma linguagem está associada a conveniência de definir uma máquina virtual adequada à execução de programas codificados nesta linguagem. Esta máquina deve aceitar, como entrada, uma seqüência de instruções que permitam a execução, da forma mais conveniente possível, das ações expressas pelas sentenças que compõem os programas codificadas na linguagem considerada.

No caso de linguagens concorrentes é essencial, ainda, incluir na máquina virtual facilidades para criação e destruição de tarefas. Além disto, mecanismos destinados a permitir a comunicação, sincronização e escalonamento de tarefas devem também ser previstos.

Na proposta original da linguagem EDISON (1), o autor deixa em aberto a definição, e mesmo o emprego, de uma máquina básica para esta linguagem. Além disto, alguns parâmetros, que devem ser considerados na implementação da linguagem, foram também deixados em aberto.

No presente trabalho é apresentada uma proposta de máquina básica, que foi dividida em duas partes.

A primeira parte, destinada a implementar o conjunto de instruções escolhido, é denominada interpretador e utiliza uma "máquina de pilha". Assim, grande parte das instruções utilizadas referenciam e/ou armazenam resultados no topo de uma pilha ou em certas posições pré-definidas desta pilha.

A segunda parte da máquina básica destina-se a tratar dos problemas relacionados à concorrência e denomina-se núcleo. Ao núcleo

cabe implementar tanto as funções de criação e destruição de processos como também as regiões críticas condicionais empregadas pela linguagem, bem como realizar as atividades relacionadas com a multiplexação dos processadores, gerenciamento de memória e inicialização da execução dos programas EDISON.

Os parâmetros - não definidos pelo autor da proposta desta linguagem - que foram utilizados na presente máquina básica encontram-se no ANEXO I.

II.2. O INTERPRETADOR

II.2.1. CARACTERISTICAS GERAIS

Um programa EDISON é composto por um conjunto de rotinas ("procedures") que podem ser aninhadas. Cada uma destas rotinas pode utilizar parâmetros (passados por valor ou por referência, i.e. parâmetros do tipo "value" ou do tipo "var"), podendo ainda retornar um valor. Tais rotinas utilizam variáveis locais (i.e. variáveis inacessíveis às rotinas globais a ela) e variáveis globais (i.e. variáveis locais a rotinas nas quais ela estiver aninhada).

Estas rotinas podem realizar operações sobre quaisquer dos seus parâmetros ou variáveis (locais ou globais), podendo ainda, nos casos das rotinas com tipo ("typed procedures"), devolver um valor à rotina que a invocar.

Na máquina de pilha utilizada, cada variável (ou valor) tem uma posição correspondente na pilha. Assim, cada vez que uma rotina é invocada, são definidos a partir da posição atualmente ocupada pelo topo da pilha as seguintes regiões:

- Região correspondente ao valor retornado;
- Região das variáveis passadas como parâmetros;
- Região de armazenamento do contexto da rotina que realizou a chamada;
- Região das variáveis locais da rotina invocada.

A região correspondente ao valor retornado é alocada antes da realização da chamada à rotina. Esta região só é reservada no caso de rotinas com tipo.

A região das variáveis passadas como parâmetros é alocada sempre que a rotina fizer uso de parâmetros. Nesta região são copiados os valores dos parâmetros passados por valor e/ou os endereços das variáveis passadas como parâmetros do tipo "var". Esta região também é preenchida antes da realização da chamada à rotina.

A região de armazenamento do contexto é reservada para "salvar" o contexto da rotina que está realizando a chamada. Uma parte desta região é preenchida durante a execução da instrução que realiza a chamada à rotina, sendo que o restante desta região é preenchido durante a execução de uma instrução específica da máquina básica, que deve ser a primeira instrução executada pela rotina invocada.

Esta instrução é também responsável pela alocação da região das variáveis locais. Esta região destina-se a armazenar os valores das variáveis locais à rotina invocada.

Para permitir a localização das variáveis na pilha são utilizadas bases. Para cada nível de aninhamento de rotinas utiliza-se uma base. Estas bases ficam armazenadas em uma "tabela de bases" que, nas nossas implementações, fica residente na memória principal.

A base correspondente a cada rotina armazena o endereço de memória associado ao início da primeira região utilizada por esta rotina. O diagrama abaixo mostra um possível instantâneo da pilha e da tabela de bases para as rotinas aninhadas A, B e C.

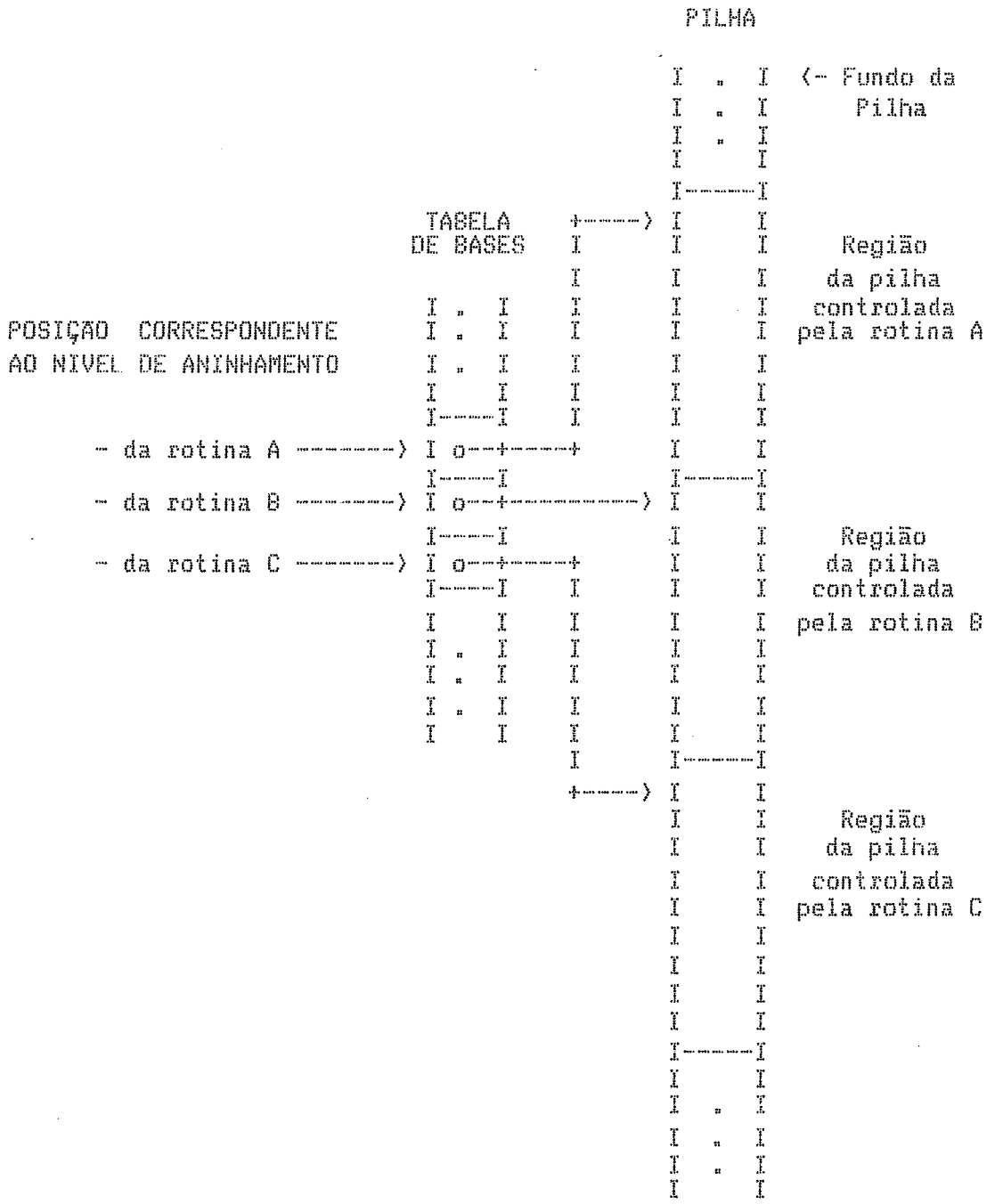

Figura II.1. Possível instantâneo da tabela de bases e da pilha.

Desta forma, uma variável local (ou parâmetro formal) de uma rotina pode ser localizada utilizando a base correspondente ao nível de aninhamento desta rotina e a distância desta variável (ou parâmetro) à esta base.

Como esta base é atualizada a cada chamada à rotina, sendo seu valor anterior "salvo" juntamente com o contexto da rotina que realizar a chamada, fica garantida a possibilidade de uso recursivo das rotinas.

Como cada processo utiliza uma tabela de bases diferente, fica garantida a possibilidade de re-entrância das rotinas.

O acesso às variáveis globais a uma rotina pode ser realizado utilizando um esquema semelhante ao empregado nas referências às variáveis locais, sendo porém necessário utilizar a base correspondente ao nível de aninhamento da rotina a qual a variável é local. Esta base pode ser obtida consultando a tabela de bases.

Partindo-se do pressuposto de que as variáveis locais a uma rotina são acessadas mais freqüentemente que as globais, optou-se por manter uma cópia da base local em um registrador da máquina virtual para evitar acessos à tabela de bases armazenada na memória principal quando forem realizadas operações que envolvam variáveis locais.

II.2.2. OS REGISTRADORES DO INTERPRETADOR

Para realizar as instruções utilizadas na máquina básica definida, o interpretador utiliza os seguintes registradores:

- PC, apontador de programa, indica o endereço da próxima instrução a ser executada;

- SP, apontador de pilha, armazena o endereço do topo da pilha;
- BASE, base local, armazena o valor da base correspondente à rotina que estiver sendo executada;
- BTBASE, endereço da tabela de bases, armazena o endereço da memória principal a partir do qual está armazenada a tabela de bases.

Além destes quatro registradores, prevê-se o uso de outros quatro destinados a estabelecer proteção de memória (STKBASE, STKLIM, CODEBASE, CODELIM) informando os limites dos espaços de endereçamento das regiões de memória principal utilizados para o armazenamento da pilha e do código executável.

Finalmente, deve-se observar que são necessários outros registradores para permitir o armazenamento dos operandos imediatos das diversas instruções empregadas, bem como resultados intermediários obtidos durante a execução destas instruções.

II.2.3. TIPOS DE ENDEREÇAMENTO UTILIZADOS

Considerando que as ações associadas a referências à variáveis, i.e. obtenção de valores de variáveis e atribuição de valores a variáveis, constituem uma grande parcela das ações realizadas durante a execução de um programa, procurou-se definir instruções que realizassem tais ações de uma forma mais eficiente possível.

Assim foram previstas diversas formas de endereçamento, de modo a realizar tais ações através da execução de um número reduzido de instruções.

Esta estratégia deverá contribuir para a diminuição dos tamanhos dos códigos associados aos programas, bem como para uma execução mais rápida destes, uma vez que o número de operações de busca e decodificação de instruções deverá também ser reduzido.

Durante a fase de especificação do repertório da máquina básica foi possível constatar que a utilização de oito tipos de endereçamentos permite a realização da maior parte das ações associadas a referências a variáveis empregando apenas uma instrução.

Os oito tipos de endereçamento foram definidos combinando-se três fases necessárias para a formação de endereços. Estas fases são consideradas seqüencialmente, i.e., a primeira fase fornece um endereço primário que é utilizado pela segunda fase. A segunda fase produz, então, um endereço secundário que é operado pela terceira fase de forma a construir o endereço efetivo.

A seguir são descritas cada uma destas fases.

a) Primeira fase - Construção do endereço primário

Esta fase é responsável pela adição de uma constante que acompanha a instrução sob a forma de operando imediato (denominado deslocamento) a uma base, de modo a produzir o endereço primário.

A base utilizada pode ser o conteúdo do registrador "base local" ou uma das entrada da tabela de bases conforme especificado por outra constante, que também acompanha a instrução sob a forma de operando imediato (denominado nível).

No primeiro caso temos um endereçamento local, que é empregado em acessos à variáveis locais ou aos parâmetros formais utili-

zados pela rotina que está sendo executada.

No segundo caso temos um endereçamento global, que é empregado em acessos às variáveis globais ou aos parâmetros formais empregados por rotinas nas quais a rotina que está sendo executada seja aninhada.

Deve-se observar que a constante denominada deslocamento (DSP) é utilizada em ambos os casos para especificar a distância entre o início da variável a ser referenciada e o início da região da pilha controlada pela rotina cuja base foi utilizada.

A constante denominada nível, que é utilizada no endereçamento global, permite especificar o nível de aninhamento da rotina à qual a variável (ou parâmetro formal) global é local.

b) Segunda fase - Construção do endereço secundário

O endereço primário pode ser utilizado diretamente como endereço secundário ou pode ser empregado como apontador para uma posição de memória na qual está armazenado o endereço secundário.

No primeiro caso temos um endereçamento direto, que é utilizado em referências a variáveis ou a parâmetros formais passados por valor, enquanto que no segundo caso temos um endereçamento indireto, que é utilizado em referências a parâmetros formais do tipo "var".

c) Terceira fase - Construção do endereço efetivo

O endereço secundário pode ser utilizado diretamente como endereço efetivo ou pode ser adicionado de um valor armazenado no topo da pilha.

No primeiro caso temos um endereçamento não indexado, no qual as variáveis ou os parâmetros formais são tratados como um todo, enquanto que no segundo caso temos um endereçamento indexado, que é utilizado para referenciar elementos de arranjos.

II.2.4. TIPOS DE INSTRUÇOES UTILIZADAS

Para implementar as ações expressas nos programas EDISON foram definidas onze categorias de instruções. As instruções que compõem as três primeiras categorias incluem campos que especificam o endereço de um operando (conforme foi visto no ítem II.2.3.), enquanto que as demais são, em sua maioria, instruções de zero endereços, que utilizam exclusivamente valores armazenados no topo da pilha e operandos imediatos.

A seguir são apresentadas as onze categorias de instruções, sendo que no ANEXO II há uma descrição individualizada das instruções empregadas.

a) Instruções para o cálculo de endereços de variáveis

Permitem armazenar o endereço de variáveis no topo da pilha e são utilizadas na passagem de parâmetros do tipo "var".

)) Instruções para obtenção de valores de variáveis

Permitem copiar o valor de uma variável para o topo da pilha. Tais instruções admitem três tipos de variações destinadas a tratar variáveis cujos tamanhos sejam um, dois ou um número arbitrário de "bytes".

c) Instruções para atribuição de valores a variáveis

Permitem copiar o valor armazenado no topo da pilha para uma variável. Tais instruções também admitem três variações destinadas a copiar valores que ocupem um, dois ou um número arbitrário de "bytes".

d) Instruções aritméticas ou lógicas

Permitem obter a soma, diferença, produto, resto da divisão inteira, conjunção ("OU" lógico) ou disjunção ("E" lógico) de dois operandos armazenados no topo da pilha. O resultado ficará também armazenado no topo da pilha. Esta categoria inclui, ainda, a instrução que faz o complemento lógico (negação) do valor booleano armazenado no topo da pilha.

c) Instruções para comparar valores simples

Estas instruções realizam operações relacionais que envolvem dois valores armazenados no topo da pilha (menor, menor ou igual, igual, maior, maior ou igual, diferente). O resultado é um valor booleano que fica também armazenado no topo da pilha.

f) Instruções para realização de operações com arranjos

Tais instruções permitem:

- Comparar dois arranjos armazenados no topo da pilha, produzindo um valor booleano que fica também armazenado no topo da pilha;
- Copiar um arranjo que acompanha a instrução sob a forma de operando imediato para o topo da pilha;
- Colocar um número arbitrário de "espaços" (brancos)

no topo da pilha;

- Converter um arranjo de palavras (p.ex. de valores inteiros) armazenados no topo da pilha em um arranjo de "bytes" (caracteres ou valores booleanos) que fica também armazenado no topo da pilha;
- Calcular a posição de um ítem em relação ao início de um arranjo. Para tanto, implementa uma função do tipo f(x) = A (x - B), onde x é um valor armazenado no topo da pilha (índice) e A (tamanho de cada elemento do arranjo) e B (limite superior do intervalo de índices válidos para o arranjo em questão) são constantes imediatas que acompanham a instrução. O valor calculado através desta função fica armazenado no topo da pilha, podendo ser utilizado posteriormente por uma instrução que empregue uma das formas de endereçamento indexado. A instrução de cálculo de posição relativa de elementos de arranjos pode ainda verificar se o valor do índice excede os limites do intervalo de índices válidos para o arranjo em questão. Para tanto, uma terceira constante imediata (que também acompanha a instrução) destinada a especificar o limite inferior deste intervalo é consultada. Se o valor calculado pela função for inválido a tarefa é abortada.

g) Instruções para realização de operações com conjuntos

Tais instruções permitem:

- Criar um conjunto vazio no topo da pilha;
- Incluir uma seqüência de elementos (armazenados no topo da pilha) em um conjunto (também armazenado na

pilha nas posições imediatamente abaixo das correspondentes aos elementos a incluir);

- Verificar se um elemento pertence a um conjunto armazenado no topo da pilha. O elemento em questão deve estar armazenado na pilha, na posição imediatamente abaixo do conjunto. O resultado booleano desta operação fica também armazenado no topo da pilha.
- Realizar a união, interseção e diferença de dois conjuntos armazenados no topo da pilha. O resultado desta operação ficará também armazenado no topo da pilha.

h) Instruções para atribuição de constantes e realização de desvios

Esta categoria inclui duas instruções destinadas a atribuir valores constantes ao topo da pilha e duas instruções que permitem a realização de desvios no fluxo de execução dos programas.

A primeira das instruções relacionadas ao uso de constantes permite copiar um valor incluído na instrução sob a forma de operando imediato para o topo da pilha, enquanto que a segunda destas instruções coloca no topo da pilha o resultado da soma do valor corrente do apontador de programa ("PC") com uma constante incluída na instrução.

Esta segunda forma deve ser utilizada quando uma rotina é passada como parâmetro para outra rotina, permitindo que o endereço da rotina passada como parâmetro seja convertido de um forma relativa ao "PC" para uma forma "absoluta".

As instruções de desvio utilizam também uma constante

imediata que será adicionada ao valor corrente de "PC". O resultado desta soma poderá ser atribuído incondicionalmente ao "PC" (no caso de desvio incondicional) ou esta atribuição ficará condicionada ao valor armazenado no topo da pilha (no caso de desvio condicional).

A condição testada no caso do desvio condicional é de que o valor armazenado no topo da pilha corresponda ao da constante booleana "false".

i) Instruções destinadas a chamar, iniciar e terminar rotinas

A primeira instrução deste grupo permite reservar na pilha uma área destinada ao armazenamento do valor retornado por uma rotina com tipo. Esta reserva é feita somando-se ao apontador do topo da pilha ("SP") um operando imediato que indica o tamanho (em "bytes") do valor que será retornado pela rotina que será invocada.

São previstas duas instruções destinadas a chamar rotinas.

Na primeira, o endereço da rotina é calculado de forma semelhante à utilizada nas instruções de desvio, i.e., uma constante que acompanha a instrução sob a forma de operando imediato é adicionada ao "PC" para obter o endereço da rotina (vide ítem h),

Na segunda destas instruções é utilizado um endereçamento indireto global, de forma que o endereço da rotina a ser invocada possa ser lido de uma variável (ou parâmetro) qualquer. Se esta variável tiver sido preenchida utilizando uma instrução de atribuição de constante relativa a "PC" (segundo tipo de instrução de atribuição de

constante - vide item h) é possível garantir uma total relocabilidade do código, uma vez que todas as posições de código serão sempre calculadas de uma forma relativa ao valor corrente do "PC".

Deve-se notar que esta segunda forma de chamada à rotinas deve ser empregada sempre que for invocada uma rotina que tenha sido passada como parâmetro.

Para iniciar a execução de uma rotina, foi prevista uma instrução específica. Os campos desta instrução indicam, através do uso de três constantes imediatas, o nível de aninhamento da rotina que está sendo iniciada, o tamanho total das regiões correspondentes ao valor retornado e às variáveis passadas como parâmetro, e o tamanho da região correspondente às variáveis locais à rotina.

A execução desta instrução permite que o valor corrente da entrada da tabela de bases correspondente ao nível de aninhamento da rotina seja "salvo" no topo da pilha juntamente com o valor deste nível de aninhamento. A seguir esta entrada é atualizada com o endereço onde começa a região da pilha controlada pela rotina.

Além disto, são também atualizados os valores da base local ("BASE") e do limite da região da pilha utilizável ("STKBASE") sendo seus valores anteriores salvos no topo da pilha.

Finalmente, ela realiza ainda a alocação das diversas regiões da pilha controladas pela rotina que está sendo iniciada.

A instrução destinada a indicar o término de rotinas ("RET") permite a liberação das regiões da pilha alocadas pela rotina que está sendo terminada e faz a restauração do contexto da rotina que realizou a chamada ("retorno").

Esta restauração compõe-se da recuperação dos valores anteriores à chamada da entrada da tabela de bases correspondente ao nível de aninhamento da rotina cuja execução está sendo terminada e dos valores dos registradores da base local ("BASE"), do limite inferior da região utilizável da pilha ("STKBASE") e do apontador de programa, que foram salvos durante a inicialização da rotina.

j) Instruções associadas ao gerenciamento de processos

Estas instruções são executadas pelo núcleo e destinam-se a permitir a criação, destruição e sincronização de processos.

l) Instruções para a realização de operações especiais

Implementam as "rotinas embutidas" (destinadas a realizar as operações de E/S) definidas na linguagem EDISON ("PLACE", "OBTAIN", "SENSE").

Deve-se notar que não está prevista uma instrução especial para implementar a "rotina embutida" "ADDR" uma vez que a ação expressa por esta rotina pode ser obtida com o uso de uma instrução para cálculo do endereço de uma variável (categoria a).

II.3. O NUCLEO

II.3.1. CARACTERISTICAS GERAIS

Durante a execução de um programa EDISON, diversas tarefas concorrentes (processos concorrentes) podem ser criadas e destruídas.

Em nossa máquina básica, decidiu-se delegar ao núcleo a

execução das atividades relacionadas com a criação e destruição de processos. Cabe também ao núcleo a sincronização e escalonamento dos processos, bem como o gerenciamento da memória do sistema, além da inicialização e término da execução dos programas EDISON.

Para realizar tais funções o núcleo utiliza listas circulares de estruturas denominadas descritores de processos.

Desta forma, a cada processo existente no sistema está associado um descritor de processos, sendo que a cada descritor de processo está associado uma tabela de bases.

O conjunto formado pelo descritor de processo e sua tabela de bases constitui o contexto do processo que, em nossas implementações, fica armazenado na memória principal em uma região que precede o fundo da pilha do processo em questão.

Nos descritores de processos ficam armazenados os valores correntes dos registradores da máquina virtual que executa o processo enquanto este processo não estiver alocado a um processador.

Estes descritores podem armazenar, ainda, um apontador para o descritor do processo que o criou (processo pai) bem como um contador destinado a armazenar o número de processos por ele criados (processos filhos).

Os descritores de processos contêm ainda dois apontadores ("NEXT") e ("LAST") que permitem organizá-los em listas circulares duplamente encadeadas.

Finalmente, são ainda armazenados nos descritores de processos informações sobre o estado do processo em relação aos

mecanismos de sincronização e de entrada e saída que serão discutidos posteriormente.

II.3.2. AS INSTRUÇOES EXECUTADAS PELO NUCLEO

O núcleo reconhece quatro instruções da máquina básica.

A execução destas instruções deve ser realizada com exclusão mútua. Desta forma, em sistemas que utilizem mais de um processador, deverá ser empregado um mecanismo que permita que cada processador informe aos demais que está executando uma instrução deste grupo.

Assim, antes de iniciar a execução de uma destas instruções, o processador deve consultar este mecanismo para verificar se algum dos outros processadores está também executando uma outra instrução do núcleo. Neste caso será necessário que o processador que pretende iniciar a execução da instrução considerada aguarde a conclusão da execução da outra instrução.

A primeira instrução deste grupo permite criar um processo, a segunda permite a conclusão normal de um processo, enquanto que as duas demais são utilizadas para delimitar regiões críticas.

A instrução de criação de processos admite duas variações. A primeira ("CRTFAT") inicializa a execução de um programa EDISON, criando o seu primeiro processo (processo pai). A segunda variação ("CRTSON") permite que um processo crie um processo filho.

Na primeira variação é utilizado um operando imediato destinado a informar o tamanho do código do programa cuja execução está sendo iniciada. A execução desta instrução gera um descritor para o processo que está sendo criado e aloca espaço para a sua tabela de

bases. Além disto, as variáveis do núcleo são também inicializadas.

Nas nossas implementações, o conjunto formado pelo descritor e a tabela de bases (contexto do processo) do primeiro processo criado é armazenado na região de memória imediatamente a seguir à utilizada para armazenar o código do programa, sendo que toda a memória remanescente é alocada para a pilha do processo criado.

A segunda variação da instrução de criação de processos utiliza dois operandos imediatos.

O valor do primeiro destes operandos é adicionado ao apontador de programa do processo que a utilizar (processo pai), indicando, de uma forma relativa, o endereço da próxima instrução que será executada pelo processo pai.

Este mecanismo é necessário porque as instruções imediatamente seguintes à instrução de criação de processos serão executadas pelo processo que está sendo criado (processo filho).

O segundo operando imediato indica o tamanho da região de memória a ser alocado à pilha do processo que está sendo criado.

Na execução desta instrução o núcleo prepara o descritor para o processo filho e o armazena no topo da pilha do processo pai. A seguir é realizada uma cópia da tabela de bases do processo pai para a região de memória imediatamente adjacente à utilizada para armazenar o descritor do processo filho, produzindo a tabela de bases deste novo processo.

A alocação de memória para o processo filho é feita reservando uma área de memória, de tamanho igual ao valor especificado pelo segundo operando imediato incluído na instrução, na pilha do pro-

cesso pai. Isto é feito adicionando o valor desta constante ao apontador de pilha do processo pai.

A criação do processo é concluída pela incrementação do contador de processos filhos do processo pai, seguido do encadeamento do descritor do processo criado à lista circular de descritores de processos.

Ao fim da criação do processo, o núcleo examina a próxima instrução a ser executada pelo processo pai. Se esta for uma outra instrução de criação de processo, ela será imediatamente executada. Caso contrário o processo pai ficará bloqueado, sendo que um processo da lista circular será selecionado para execução.

A execução de uma instrução de término de processo é realizada verificando se o processo que a está executando foi o primeiro processo criado, sendo que neste caso a execução do programa é terminada. Caso contrário, o contador de processos filhos do processo pai é decrementado. Se o novo valor deste contador for zero, este processo pai será reativado e incluído na lista circular. A seguir será selecionado para execução um processo da lista circular.

Na linguagem EDISON é possível declarar regiões críticas condicionais através da construção "when". As instruções que compõem estas regiões críticas devem ser executadas de uma forma não interruptível, de modo a garantir a exclusão mútua no uso de certos recursos compartilhados.

Ao final da execução destas instruçõoes é necessário verificar se alguma das condições que guardam a região crítica foi atendida. Caso positivo a execução do "when" é concluída. Caso contrário

as condições deverão ser reavaliadas posteriormente.

Para permitir a realização destas regiões críticas a máquina básica utiliza uma instrução que indica o início de uma região crítica e outra que assinala o final desta região.

A execução da instrução destinada a indicar o início de uma região crítica condicional causa a movimentação da constante booleana "true" para o topo da pilha do processo, seguida pelo armazenamento dos valores dos registradores do interpretador no descritor do processo que está sendo executado. Este descritor é também marcado, de forma a assinalar que o processo a ele correspondente está em uma região crítica. A seguir um processo da lista circular é selecionado para execução.

A execução da instrução destinada a indicar o fim de uma região crítica é iniciada pelo teste do valor armazenado no topo da pilha do processo que a invocou.

Caso este valor corresponda à constante booleana "false", o núcleo conclui que uma das condições que guarda a região crítica foi atendida. Neste caso, os valores dos registradores do interpretador são armazenados no descritor do processo que está sendo executado, sendo que este processo é marcado como não estando em região crítica.

A seguir, independentemente do resultado de teste mencionado no parágrafo anterior, um processo da lista circular será selecionado para execução.

II.3.3. ALGUMAS ESTRATEGIAS DE ESCALONAMENTO

Neste trabalho optou-se por não utilizar qualquer forma de temporização para apoiar o esquema de escalonamento.

Esta opção foi feita tendo em vista a conveniência de possibilitar o uso da máquina básica desenvolvida em equipamentos de baixo custo, que não disponham de facilidades para utilização de mecanismos de temporização por "hardware".

Desta forma o problema de escalonamento ficou reduzido à estratégia a ser adotada na seleção do próximo processo a receber o controle de um processador que fique disponível.

O núcleo considera que um processador está disponível após o término da execução de qualquer uma das instruções do núcleo ou quando ocorrer o término anormal de um processo.

Na discussão das possíveis formas de seleção do processo que deve receber o controle de um processador é conveniente considerar isoladamente as situações que envolvem sistemas dotados de apenas um processador daquelas em que estejam envolvidos sistemas multiprocessadores.

No primeiro caso será sempre possível selecionar para execução um único processo dentre aqueles que estejam em região crítica, ao contrário do segundo caso, no qual tal seleção nem sempre será possível porque apenas um dos processadores deve executar um processo que esteja em região crítica de cada vez.

Como todas as implementações levadas a cabo no decorrer deste trabalho envolveram sistemas dotados de apenas um processador, especial ênfase foi dada às considerações sobre as alternativas de

políticas de escalonamento permissíveis neste caso.

A primeira possível alternativa seria realizar a escolha do próximo processo a ser selecionado percorrendo simplesmente a lista circular de processos, de forma a selecionar o primeiro processo nela encontrado, sem levar em consideração o fato deste processo estar ou não em região crítica.

O principal problema associado a esta alternativa está ligado ao fato de não ser utilizada qualquer forma de temporização no escalonamento dos processos.

Como o número de instruções realizadas em regiões críticas tende a ser pequeno comparado com o número de instruções realizadas fora destas regiões, ter-se-ia uma penalização excessiva dos processos que fizessem uso de regiões críticas freqüentemente.

Aliado a este fato, haveria dificuldade de utilizar um esquema de priorização dos processos no qual as prioridades fossem atribuídas de uma forma estática. Isto aconteceria porque caso a lista circular fosse percorrida de uma forma que levasse em conta as prioridades dos processos, sendo estas prioridades fixadas, ter-se-ia uma situação na qual os processos menos prioritários somente seriam selecionados para execução após o final da execução dos processos mais prioritários (se tal fosse possível).

Desta forma, o uso de prioridades, que é indispensável em certas situações (aplicações que envolvam atuação em tempo real, por exemplo) obrigaria o uso de um esquema que incluisse uma atribuição dinâmica de prioridades que, além de complicar o mecanismo de escalonamento, tenderia a causar um aumento na quantidade de tempo do processa-

dor necessário para determinar o processo a ser ativado ("overhead").

A segunda alternativa considerada foi a de agrupar os processos em região crítica em uma lista circular independente da lista utilizada para os demais processos. Desta forma é possível utilizar um esquema que permita que um processo da lista principal somente seja selecionado após ter sido realizada uma volta completa na lista dos processos em região crítica, implementando uma política do tipo "foreground" e "background".

Esta alternativa implica em uma comutação de contextos de processos mais freqüente que a anterior, mas permite o uso de um esquema de prioridades fixadas, além de não penalizar os processos que utilizem freqüentemente as regiões críticas.

Nas duas primeiras implementações desta máquina básica não foi considerado o problema do uso de prioridades e a alternativa adotada foi a primeira, enquanto que nas demais a alternativa adotada foi a segunda, sendo permitido portanto o uso de prioridades estáticas.

Em sistemas dotados de vários processadores poder-se-ia considerar uma alternativa de política de escalonamento que envolvesse a seleção, sempre que possível de um processo que estivesse em região crítica (i.e., sempre que um processador ficar disponível e nenhum dos demais processadores estiver executando processo em região crítica, será selecionado para execução um processo que esteja em região crítica).

Esta alternativa, de implementação bastante simples, poderia ter resultados convenientes porque seria dado aos processos em região crítica a maior prioridade possível causando uma sobrecarga ao sistema relativamente pequena (limitada pelo inverso do número de processadores).

III. AS IMPLEMENTAÇOES

Foram realizadas quatro diferentes implementações da máquina básica descrita neste trabalho. As duas primeiras foram levadas a cabo através de programas codificados em linguagens de alto nível, enquanto que as demais envolveram o uso de linguagens de baixo nível.

Estas diferentes implementações foram feitas considerando as dificuldades inerentes à programação nestes diferentes níveis e os resultados, em termos de eficiência de utilização dos recursos de "hardware" disponíveis.

Assim, as implementações em linguagens de alto nível não apresentaram maiores dificuldades de programação, estando associadas, porém, aos maiores tempos de execução.

As implementações levadas a cabo utilizando linguagens de baixo nível ("assembly" e micro-programa), por outro lado, apresentaram dificuldades de programação maiores, conduzindo, porém a tempos de execução de programas muito menores.

III.1. AS IMPLEMENTAÇOES EM LINGUAGENS DE ALTO NIVEL

A primeira implementação da máquina básica foi levada a cabo utilizando a linguagem ALGOL para a máquina B-6700 (3).

Esta implementação foi construída ao mesmo tempo que as especificações da máquina básica, sendo utilizada para consolidar estas especificações.

Esta estratégia foi adotada tendo em vista a alta legibilidade dos programas codificados em ALGOL, a par da conveniência de

produzir uma documentação da máquina básica definida sob a forma de um programa.

Em termos de eficiência da utilização dos recursos de "hardware", porém, esta implementação é extremamente inadequada. Na realidade é virtualmente impossível utilizar esta implementação na execução de programas EDISON reais em função da baixa eficiência no uso do processador, aliado ao fato de que a máquina utilizada trabalha em regime de multiusuário.

Todavia, o programa que constitui esta implementação foi de grande valia no desenvolvimento de todas as demais implementações, sendo sua listagem apresentada no ANEXO III.

A segunda implementação foi desenvolvida na linguagem LPS para o equipamento COBRA-300, constituindo a primeira versão realmente utilizável da máquina básica.

A realização desta implementação foi levada a cabo por Zancanella (1), sendo empregada na validação do compilador EDISON desenvolvido. Esta implementação permite a execução de pequenos programas, cujo tamanho é limitado pelo tamanho da memória utilizável no equipamento disponível (42 kbytes), aliado ao tamanho relativamente grande do programa construído (8 kbytes) para levar a cabo esta implementação.

III.2. AS IMPLEMENTAÇÕES EM LINGUAGENS DE BAIXO NIVEL

As duas implementações em linguagens de baixo nível foram construídas para o computador MITRA-15 da CII-THOMSON.

A primeira destas implementações utiliza um programa

codificado em uma linguagem "assembly" disponível no computador MITRA-15 (4) enquanto que a segunda emprega um micro-programa para esta máquina.

A implementação em "assembly" foi construída basicamente a partir de uma "tradução" do programa utilizado na implementação em ALGOL para a linguagem aceita pelo "assembler" da máquina empregada.

Esta "tradução" embora lenta, trabalhosa e sujeita a inevitáveis erros de codificação (que por vezes tornam-se difíceis de isolar) não apresenta nenhuma particularidade digna de nota.

Esta implementação, a exemplo da implementação em ALGOL, inclui testes de validação realizados previamente à execução de cada uma das operações especificadas nas instruções para facilitar a depuração dos programas EDISON.

Assim, antes da execução de cada instrução são testados os seguintes ítens:

- Validade do conteúdo do apontador de programas;
- Possíveis situações de "overflow" ou "underflow" da pilha como consequência da execução da instrução;
- Códigos de instrução inexistente ou operandos imediatos com valores incompatíveis;
- "Overflow" aritmético e índices inválidos;
- Criação de processos filhos por processos filhos;
- Tentativas de aninhamento de regiões críticas;
- Inconsistências nas filas de processos.

Apesar destes exaustivos testes, a eficiência desta implementação é bastante razoável, sendo que são executadas cerca de mil instruções da máquina básica por segundo (no ANEXO VII pode ser encon-

trado o tempo de execução de cada uma das instruções que compõem o repertório da máquina básica nesta implementação).

O programa construído para realizar esta implementação foi dividido em quatro módulos de compilação e ocupa cerca de 4,5 kbytes de memória.

A implementação a nível de micro-código não realiza os testes de validação como nas demais implementações em virtude da limitação do tamanho da memória de micro-programa (1k palavras), aliada às dificuldades inerentes à micro-programação da máquina utilizada (p.ex. dificuldade de utilização de micro-subrotinas) que serão detalhadamente discutidas nos próximos capítulos.

Esta implementação, cuja estratégia é discutida em detalhes no capítulo seis permite utilizar com grande eficiência os recursos de "hardware" disponíveis, resultando na possibilidade de execução de cerca de cem mil instruções da máquina básica por segundo, como pode ser inferido a partir dos tempos de execução de cada uma destas instruções, que estão relacionados no ANEXO VII.

IV. A MICRO-MAQUINA DO COMPUTADOR MITRA-15

IV.1. CONSIDERAÇOES GERAIS

Uma descrição completa desta micro-máquina pode ser encontrada na publicação denominada "ORDINATEUR MITRA 15 - ORGANES CENTRAUX" de autoria de Brewee, Leconte e Thierion (5). Como esta publicação tem uma circulação bastante restrita, é apresentado, a seguir, um resumo das principais características desta micro-máquina para permitir a compreensão dos capítulos seguintes deste trabalho.

Além desta descrição, um resumo das principais alterações realizadas no "hardware" desta micro-máquina afim de permitir modificação dinâmica do micro-programa, é também apresentado.

IV.2. A ARQUITETURA DA MICRO-MAQUINA

A figura IV.1. apresenta um esquema geral das principais partes da micro-máquina e suas interconexões. A seguir estão descritos cada um dos blocos apresentados nesta figura.

a) O Relógio e a Base de Tempo ("RLG")

A base de tempo fornece um sinal de relógio (Mi) a cada micro-instrução. Normalmente o período deste sinal é de 300 nS. Este período pode ser dilatado quando for realizada alguma operação de referência à memória principal ou a algum periférico. Nestes casos a chegada deste sinal é retardada até que a operação considerada esteja completada.

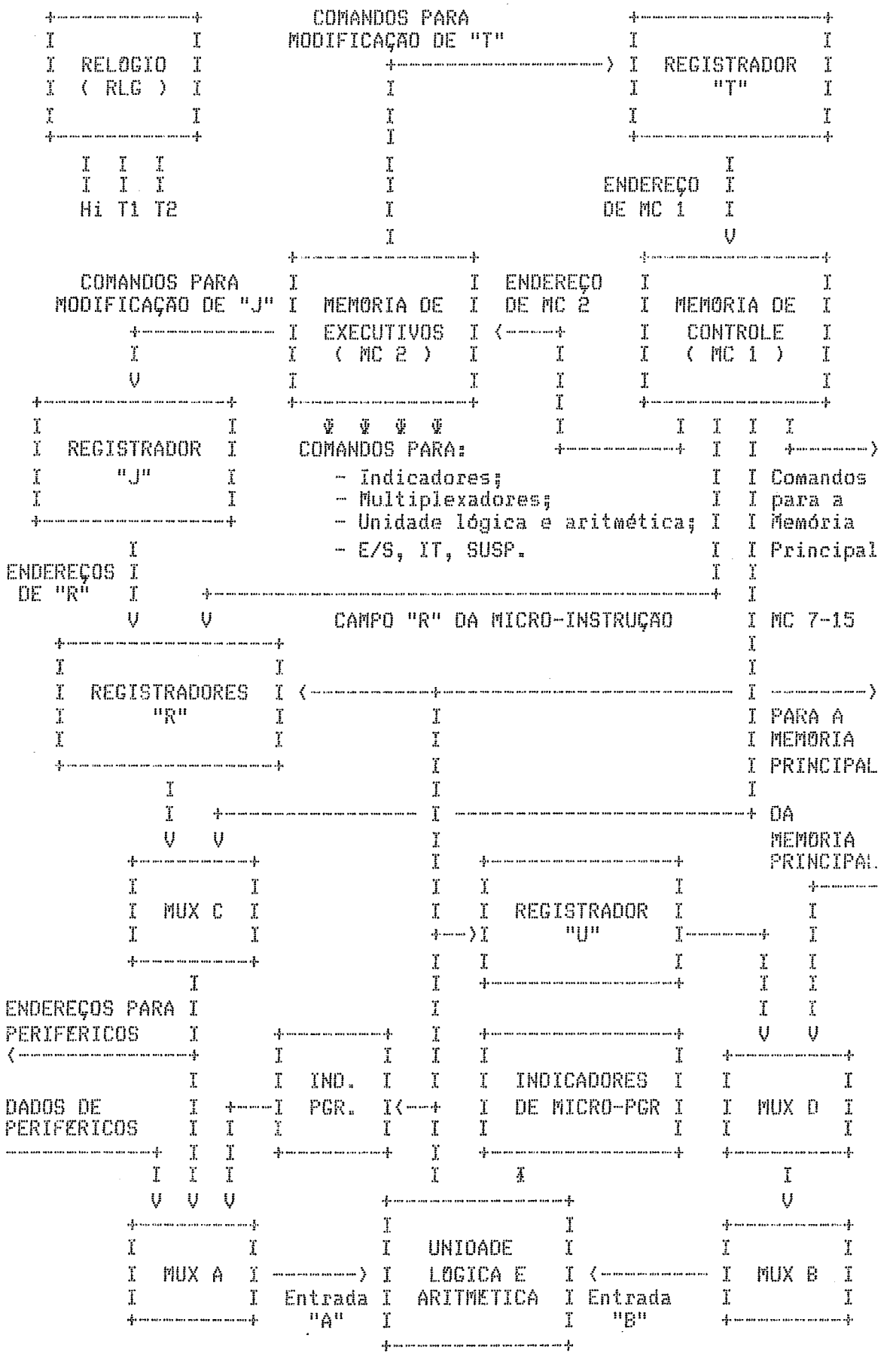

Figura IV.1. Micro-máquina do Computador MITRA-15

O período de Hi é sub-dividido em quatro fases. A primeira (T1) corresponde ao tempo de seleção da próxima micro-instrução (através de T_y MC 1 e MC 2) e dura em média 200 nS. A segunda fase (θ1) é um eventual tempo de espera pela chegada de uma informação externa à micro-máquina. A terceira fase (T2) corresponde ao tempo de execução das operações especificadas pela micro-instrução e dura em média 100 nS. A quarta fase (θ2) corresponde a um eventual tempo de espera, utilizado entre duas micro-instruções consecutivas que referenciem a memória principal.

Deve-se observar que a base de tempo gera apenas os sinais T1 e T2, sendo que as fases θ1 e θ2 não precisam ser explicitamente informadas ao restante da micro-máquina.

b) A Memória de Controle (MC1)

Esta memória destina-se a armazenar o micro-programa, sendo constituída de 1024 palavras de 16 bits. Na micro-máquina original, esta memória é do tipo "PROM", sendo que, para a realização do presente trabalho, foi adicionada uma outra memória de igual capacidade, porém do tipo "RAM".

Cada palavra desta memória corresponde a uma micro-instrução e está dividida em quatro campos (vide figura IV.2.).

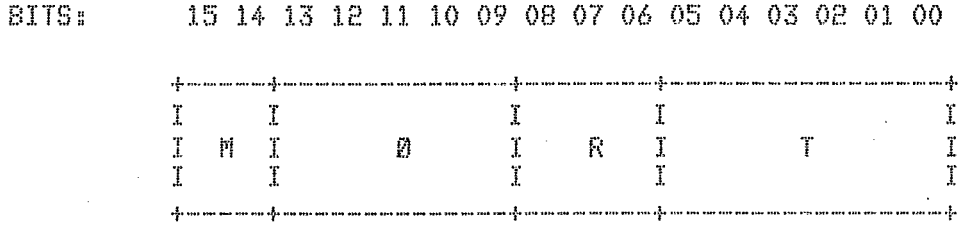

Figura IV.2. Campos da palavra da micro-memória

O primeiro campo (M), formado pelos dois bits mais significativos, destina-se a informar o tipo de operação de memória a ser realizado (0 = escrita, 1 = leitura, 2 = reescrita, 3 = nenhuma operação).

O segundo campo (Ø), formado pelos cinco bits imediatamente adjacentes ao primeiro campo, especifica o tipo de micro-instrução. Este campo é sub-dividido em dois sub-campos. O primeiro, que corresponde aos três bits mais significativos, informa a classe da micro-instrução (0 = classe ØR, 1 a 6 = classe Ø, 7 = classe ØS). O segundo sub-campo, associado aos dois bits menos significativos, permite informar a sub-classe da micro-instrução.

O terceiro campo (R), formado pelos três bits imediatamente adjacentes ao segundo campo, é utilizado para informar o número de um registrador ou de um indicador cujo conteúdo será consultado ou modificado pela execução da micro-instrução. No caso das micro-instruções da classe ØR este campo fornece uma especificação complementar ao tipo de micro-instrução.

O quarto campo (T), constituído pelos seis bits menos significativos da palavra, tem um significado variável de acordo com o tipo de micro-instrução. Assim, nas micro-instruções das classes Ø e ØR ele pode armazenar um parâmetro utilizado para o cálculo do endereço da próxima micro-instrução ou uma constante que será carregada no registrador "U". Nas micro-instruções da classe ØS, seu bit mais significativo, concatenado com seus dois bits menos significativos, determinam um parâmetro utilizado para o cálculo do endereço da próxima micro-instrução. Os demais bits fornecem uma especificação complementar ao tipo de micro-instrução.

c) A Memória de executivos (MC 2)

Esta memória destina-se a armazenar uma forma de nano-programa (denominado "executivos"), sendo constituida de 96 palavras de 48 bits. Esta memória é do tipo "ROM" e de suas 96 palavras, 88 são acessíveis ao micro-programa, enquanto que as 8 demais são acessíveis apenas através da manipulação das chaves do painel do computador e destinam-se a fornecer facilidades para a manutenção.

Cada palavra desta memória contém as ordens necessárias para controlar os recursos da unidade central de processamento durante a execução de uma micro-instrução. A figura IV.3. apresenta o significado de cada um dos seus campos.

As 96 palavras desta memória estão distribuídas em três grupos de 32.

O primeiro grupo armazena os 8 executivos de suporte à manutenção (endereços de 0 a 3 e de 28 a 31) e os 24 executivos referenciados pelas micro-instruções da classe Ø (endereços de 4 a 27), sendo endereçado pelo campo Ø das micro-instruções.

O segundo grupo armazena os 32 executivos referenciados pelas micro-instruções da classe ØR, sendo endereçado pelo resultado da concatenação dos dois bits menos significativos do campo Ø com os três bits que compõem o campo "R" das micro-instruções.

O terceiro grupo armazena os 32 executivos referenciados pelas micro-instruções da classe ØS, sendo endereçado pela concatenação dos dois bits menos significativos do campo Ø com três bits do campo "T" das micro-instruções (vide discussão dos campos da memória de controle no ítem b).

POSIÇÃO (BIT)	TAMANHO (BIT)	NOME	SIGNIFICADO
00 - 03	4	NSGDC	Comando do "byte" direito da ULA
04 - 07	4	NSGGC	Comando do "byte" esq. da ULA
08 - 08	1	NSGLA	Modo de funcionamento da ULA
09 - 11	3	NBTOC	Comando do bit 0 do Mutiplex B
12 - 14	3	NBTC	Comando dos bits de 1 a 14 do Multiplex B
15 - 17	3	NBTFC	Comando do bit 15 do multiplex B
18 - 19	2	NALC	Comando do multiplexador A
20 - 20	1	NGACO	Comando do multiplexador C
21 - 21	1	NMUCO	Comando do multiplexador O
22 - 23	2	NCINC	Comando do "vai um"
24 - 25	2	NRUC	Comando do registrador "U"
26 - 26	1	NRRC	Comando dos registradores "R"
27 - 28	2	NRJC	Comando das modificações de "J"
29 - 33	5	NRTC	Comando das modificações de "T"
34 - 34	1	NPARAM	Zeramento automático de "T"
35 - 37	3	NIDC	Comando dos indicadores I1 e I2
38 - 38	1	NMACO	Comando do indicador MA
39 - 39	1	NMSCO	Comando dos indicadores MS e PM
40 - 42	3	NIBC	Comando do indicador B
43 - 43	1	NSOC	Comando do indicador SO (PO)
44 - 45	2	NESC	Comando de Entrada e Saída
46 - 46	1	NTITC	Comando das interrupções
47 - 47	1	NSUCO	Comando das suspensões

Figura IV.3. Campos da palavra da memória de executivos

Desta forma a execução de uma micro-instrução é iniciada pela seleção do executivo nela especificado durante a fase T1 do ciclo do relógio. Durante a fase T2 deste ciclo, os conteúdos dos diversos campos do executivo selecionado são transferidos para as linhas de controle da micro-máquina permitindo que as operações nele especificadas sejam levadas a cabo.

d) O apontador de Micro-programa (Registrador "T")

Este registrador é composto por dez "flip-flops" que permitem endereçar as 1024 palavras da memória de controle (MC 1). Estes "flip-flops" estão organizados em dois grupos de cinco. O grupo correspondente aos cinco bits mais significativos do registrador "T" define um campo denominado "campo de página" enquanto que o outro grupo define o "campo de palavra".

Durante o ciclo T2 do relógio o valor armazenado no registrador "T" sofre uma modificação de modo a endereçar a próxima micro-instrução a ser executada.

São utilizados seis diferentes tipos de modificações do valor deste registrador, sendo que para uma dada micro-instrução o tipo de modificação do valor do registrador "T" é especificado pelo campo NRTC (de cinco bits) do executivo por ela invocado. A seguir estão apresentados estes seis tipos de modificação do valor do registrador "T".

- Primeiro tipo de modificação de "T" (NRTC igual a 0)

Este tipo de modificação admite duas variações de acordo com a classe a qual pertence a micro-instrução que está sendo executada.

A primeira variação, denominada modificação normal de "T", é selecionada nos casos de micro-instruções das classes Ø e ØR e permite a atribuição do valor armazenado nos cinco bits menos significativos do campo "T" da micro-instrução que está sendo executada ao campo página ou ao campo palavra do registrador "T".

Desta forma, se o bit mais significativo do campo "T" da micro-instrução corrente contiver o valor zero, será realizada a atribuição ao campo palavra do registrador "T". Caso contrário, a atribuição será realizada ao campo página.

A segunda variação, denominada modificação restrita de "T", é selecionada nos casos de micro-instruções da classe ØS. O mecanismo utilizado neste tipo de modificação de "T" é semelhante ao empregado no caso de modificação normal de "T", diferindo apenas pelo fato de que somente os dois bits menos significativos do campo "T" são utilizados na atribuição (que pode ser realizada sobre os dois bits menos significativos do campo página ou do campo palavra do registrador "T, de acordo com o valor do bit mais significativo do campo "T" da micro-instrução, de forma similar à modificação normal de "T"). Esta restrição é necessária em virtude da utilização dos três demais bits do campo "T" da micro-instrução como especificação complementar do tipo de executivo a ser referenciado.

- Segundo tipo de modificação de "T"
(NRTC igual a 1 ou 2)

Este tipo de modificação do valor do registrador "T" permite realizar desvios condicionados ao valor de um dos oito indicadores da micro-máquina. As micro-instruções que utilizam este tipo de

modificação de "T" são sempre da classe Ø e o campo R destas micro-instruções é utilizado para especificar qual dos indicadores deve ser considerado.

Este tipo de modificação admite duas variações. A primeira (NRTC = 1) faz uma modificação normal de "T" caso o valor do indicador especificado seja zero enquanto que a segunda (NRTC = 2) faz este tipo de modificação caso o valor do indicador seja um. Em ambas as variações, o não atendimento da condição testada implica na realização de uma modificação de "T" do terceiro tipo.

- Terceiro tipo de modificação de "T"
  (NRTC igual a 3, 7, 11 ou 15)

Este tipo de modificação, denominada modificação implícita de "T", realiza uma simples incrementação do valor armazenado nos quatro bits menos significativos do registrador "T" (pertencentes ao campo palavra).

- Quarto tipo de modificação de "T" (NRTC igual a 4)

Este tipo de modificação do registrador "T", denominado recópia de "T", não realiza modificação alguma no valor armazenado neste registrador e destina-se primariamente a implementar um "HALT".

- Quinto tipo de modificação de "T" (NRTC igual a 12)

Este tipo de modificação do registrador "T", denominado retorno pela pilha, atribui um valor armazenado no topo de uma pilha existente na micro-máquina ao registrador "T". Este tipo de modificação destina-se primariamente a permitir o término de uma rotina de trata-

mento de suspensão (i.e., interrupções no nível de micro-código), realizando o retorno do controle da micro-máquina ao micro-programa interrompido.

- Sexto tipo de modificação de "T"

Este tipo de modificação, denominado modificação de "T" com mascaramento, permite atribuir a um conjunto de até seis bits do registrador "T" valores obtidos de um outro registrador da micro-máquina ou da saída da "ULA".

O campo formado pelos sete bits menos significativos do registador "T" é afetado por este tipo de modificação. O bit menos significativo é sempre complementado. A modificação (ou não) dos seis bits restantes é controlada por uma "máscara" localizada no campo "T" da micro-instrução corrente. A figura IV.4. apresenta um exemplo de uma modificação deste tipo.

Este tipo de modificação do registrador "T" é particularmente conveniente no caso de decodificação das instruções do nível convencional.

O registrador "T" é dotado de uma capacidade de zeramento automático que é habilitada sempre que o campo NPARAM do executivo for um (independentemente do valor do campo NRTC) e o resultado da modificação de "T" for um valor idêntico ao anteriormente armazenado neste registrador.

Esta facilidade pode ser empregada no fim da execução da seqüência de micro-instruções correspondente à realização de uma instrução do nível convencional, quando é necessário retornar ao início do

micro-programa para começar a interpretar a próxima instrução do nível convencional.

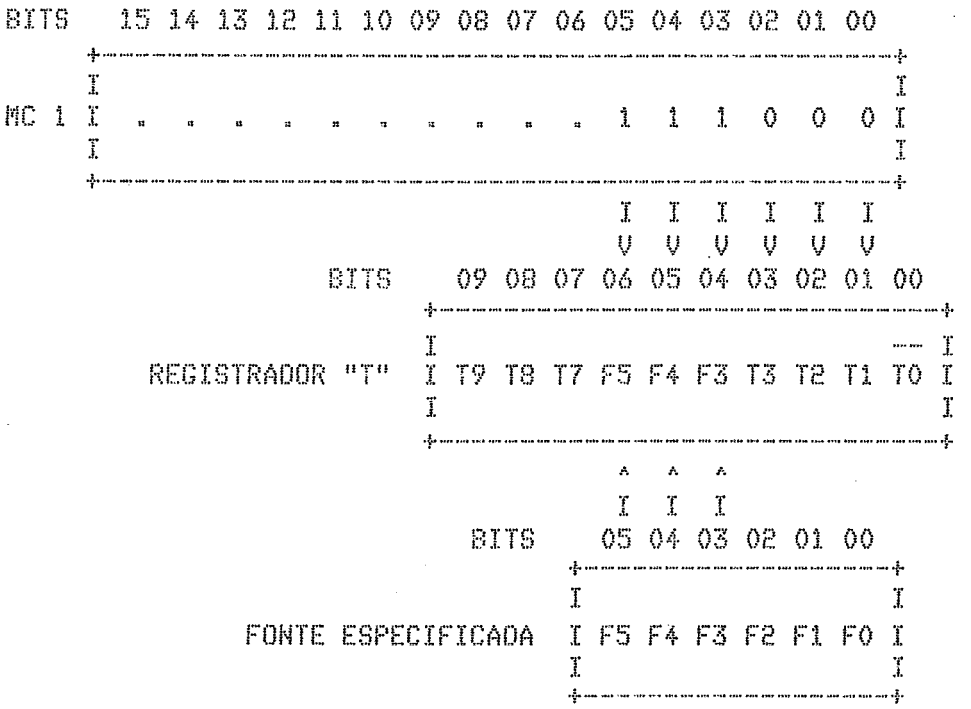

Figura IV.4. Exemplo de modificação de "T" com mascaramento

e) O Banco de Registradores ("R")

Este banco é composto por uma memória do tipo "RAM" organizada em grupos de oito palavras de 16 bits. E previsto o uso de um número variável de grupos (compreendido entre dois e trinta e dois), sendo que na configuração utilizada estão disponíveis oito grupos totalizando assim 64 palavras.

Esta memória é endereçada pelo registrador "J", que é constituído de oito bits. Os cinco bits mais significativos são utilizados para selecionar um dos trinta e dois possíveis grupos.

A seleção de um registrador dentro de um grupo é feita utilizando o resultado da conjunção ("OU" lógico) dos três bits menos

significativos do registrador "J" com os três bits que compõem o campo "R" da micro-instrução.

As micro-instruções que referenciam executivos nos quais o campo NRRC tenha valor um movimentam o conteúdo da saída da unidade lógica-aritmética para o registrador selecionado de acordo com o esquema de endereçamento indicado no parágrafo anterior.

f) O Registrador de Endereçamento
do banco de registradores ("J")

Conforme já foi mencionado, este registrador contribui na formação do endereço do registrador do banco de registradores afetado pela ação de uma micro-instrução.

O valor armazenado neste registrador pode sofrer uma modificação ao final da execução de uma micro-instrução.

Existem quatro tipos de modificações deste registrador. O campo NRJC (de dois bits) da palavra da memória de executivos seleciona o tipo de modificação a ser utilizado ao fim de uma dada micro-instrução.

O primeiro tipo de modificação (que é utilizado na maior parte dos executivos) corresponde a uma modificação "dummy" que não altera o valor deste registrador.

O segundo tipo de modificação de "J" permite realizar uma movimentação dos oito bits menos significativos da saída da unidade lógica-aritmética para o registrador "J".

O terceiro tipo de modificação de "J", denominado retorno

pela pilha, transfere o valor armazenado no topo de uma pilha existente na micro-máquina para o registrador "J". Este tipo de modificação é usado no término de rotinas de tratamento de suspensões para restaurar o valor do registrador "J" anterior ao início do tratamento da suspensão.

O quarto tipo de modificação de "J" movimenta zeros para os oito bits do registrador "J".

g) A Unidade Lógica-aritmética ("ULA")

A unidade lógica-aritmética empregada nesta micro-máquina opera sobre dois valores de 16 bits (presentes nas saídas dos multiplexadores "A" e "B") e produz um resultado (também de 16 bits) que pode ser utilizado em diferentes partes da micro-máquina (vide figura IV.1.).

Além destas interconexões principais existe ainda uma linha de entrada de "vai-um" e uma correspondente linha de saída, de forma a permitir a realização de operações que envolvam várias palavras.

A operação a ser realizada pela "ULA" é especificada por três campos da palavra da memória de executivos.

O primeiro destes campos, NSGLA (de um bit), permite especificar o funcionamento em modo lógico ou em modo aritmético.

Os dois demais campos, NSGDC e NSGGC (de quatro bits cada), permitem especificar o tipo de operação a ser realizada respectivamente sobre os oito bits menos significativos dos dados processados pela "ULA" e sobre os oito bits mais significativos destes dados, de acordo com a tabela apresentada na figura IV.5.

Valor de NSGDC ou de NSGCC	Valor presente na saída da "ULA"	
	Modo Aritmético	Modo Lógico
0	A − 1 + Vai	$\overline{A}$
1	(A . B) − 1 + Vai	$\overline{(A . B)}$
2	$\overline{(A . B)}$ − 1 + Vai	$\overline{A \cup B}$
3	−1 + Vai	1
4	A + $\overline{(A \cup B)}$ + Vai	$\overline{(A \cup B)}$
5	(A . B) + $\overline{(A \cup B)}$ + Vai	$\overline{B}$
6	A − B − 1 + Vai	$\overline{(A \oplus B)}$
7	$\overline{(A \cup B)}$ + Vai	$\overline{A \cup B}$
8	A + $\overline{(A \cup B)}$ + Vai	$\overline{A . B}$
9	A + B + Vai	A ⊕ B
10	$\overline{(A . B)}$ + $\overline{(A \cup B)}$ + Vai	B
11	(A ∪ B) + Vai	A ∪ B
12	A + A + Vai	0
13	$\overline{(A . B)}$ + A + Vai	$\overline{A . B}$
14	(A . B) + A + Vai	A . B
15	A + Vai	A

Obs. "Vai" representa a entrada de "vai um" da "ULA"

Figura IV.5. Tabela de operações da "ULA"

h) O Registrador "U"

Este registrador é utilizado para armazenar os valores intermediários obtidos em computações que envolvam mais de uma operação da "ULA".

A modificação do conteúdo deste registrador é controlada pelo campo NRUC (de dois bits) da palavra da memória de executivos.

Se o valor deste campo for zero, o conteúdo do registrador "U" não é afetado pela saída da "ULA"; se este valor for um, o conteúdo do registrador "U" também não é afetado pela saída da "ULA" mas os seus oito bits mais significativos serão zerados.

Se o valor de NRUC for dois, o valor da saída da "ULA" é copiado em "U"; se o valor deste campo for três, os oito bits menos significativos da saída da "ULA" serão copiados nos oito bits menos significativos do registrador "U" ao mesmo tempo em que os oito bits mais significativos do registrador receberão zeros.

i) O Multiplexador "A" ("MUX A")

Este multiplexador, que é controlado pelo campo NALC (de dois bits) da palavra da memória de executivos, tem quatro entradas, sendo sua saída conectada a entrada "A" da "ULA".

A primeira destas entradas é conectada à saída do multiplexador "C", a segunda é também conectada à saída deste multiplexador, sendo que os dois "bytes" são permutados (i.e. o "byte" mais significativo da saída é conectado ao "byte" menos significativo da entrada e o "byte" menos significativo da saída é conectado ao "byte" mais significativo da entrada).

A terceira entrada do "MUX A" é oriunda de um barramento utilizado para transferir dados provenientes de periféricos enquanto que a quarta entrada é conectada à saída de um registrador de cinco bits que é destinado ao armazenamento dos indicadores do nível convencional.

j) O Multiplexador "B" ("MUX B")

O multiplexador "B" é composto por três multiplexadores de oito entradas que são conectados a entrada "B" da "ULA".

O primeiro destes multiplexadores é controlado pelo campo NBTFC (de três bits) da palavra da memória de executivos e atua sobre o bit menos significativo desta entrada.

O terceiro destes multiplexadores é controlado pelo campo NBTOC (de três bits) da palavra da memória de executivos e atua sobre o bit mais significativo desta entrada.

Finalmente, o segundo destes multiplexadores é controlado pelo campo NBTC (de três bits) da palavra de executivos e atua sobre os 14 demais bits da entrada "B" da "ULA".

Todos os oito conjuntos de entrada dos três multiplexadores que compõem o "MUX B" são conectados à saída do multiplexador "D" e à saída do indicador "B" permitindo o deslocamento do valor presente em "D" de um bit para a esquerda ou para a direita, bem como a permutação dos dois "bytes" que formam esta saída, como pode ser observado na tabela apresentada na figura IV.6.

Valores de NBTFC, NBTC ou NBTOC	Saída do multiplexador do bit menos significativo	Saída do multiplexador dos 14 bits intermediários	Saída do multiplexador do bit mais significativo
0	D 0	D (1 - 14)	D 15
1	D 8	P (1 - 14)	D 7
2	D 1	D (2 - 15)	0
3	0	D (0 - 13)	D 14
4	B	0	B
5	C 15	0	D 15
6	1	0	D 0
7	1	0	1

OBSERVAÇÕES: B representa a saída do indicador "B";

C representa a saída do multiplexador "C";

D representa a saída do multiplexador "D";

Os índices colocados abaixo das letras que indicam as saídas dos multiplexadores especificam os bits desta saída que serão utilizados (O bit 0 é o menos significativo da saída e o bit 15 é o mais significativo).

Figura IV.6. Tabela das ações do multiplexador "B"

l) O multiplexador "C" ("MUX C")

O "MUX C" é controlado pelo campo NGACO (de um bit) da palavra da memória de executivos e possui apenas duas entradas.

Sua saída é conectada a uma das entradas do multiplexador "A". Uma de suas entradas é conectada à saída do banco de registradores "R" enquanto a outra é conectada às nove linhas correspondentes aos bits menos significativos da palavra da memória de controle. Nesta segunda entrada as sete linhas correspondentes aos bits mais significativos são zeradas.

1) O multiplexador "D" ("MUX D")

Este multiplexador é controlado pelo campo NMUCO (de um bit) da palavra da memória de executivos e possui apenas duas entradas.

Sua saída é conectada à entrada do multiplexador "B". Uma de suas entradas é conectada à saída do registrador "U" e a outra é ligada à memória principal.

n) Os indicadores de programa ("IND. PGR.")

A micro-máquina dispõe de cinco indicadores que podem ser usados no nível convencional. Dois destes indicadores não podem ser testados pelo micro-programa e têm uso fixo (PM - indicador de proteção de memória e MA - indicador de máscara das interrupções) enquanto que os demais tem seu significado determinado pelo micro-programa e podem ser testados por este.

Estes três últimos indicadores são utilizados no micro-programa original para indicar o funcionamento em modo mestre (MS); o

"vai um" no final de uma instrução aritmética do nível convencional ou o carregamento do valor zero em um dos registradores acessíveis no nível convencional (I1); o "overflow" detetado no final de uma instrução aritmética ou o carregamento de um valor negativo em um dos registradores acessíveis ao nível convencional (I2).

O indicador de proteção de memória com valor zero em conjunto com o posicionamento da chave de proteção de memória existente no painel do computador permite ativar o mecanismo de proteção de memória.

Este mecanismo impede que palavras da memória principal marcadas como protegidas sofram alterações. A tentativa de violação desta restrição causará um desvio na seqüência normal de execução do micro-programa, que reportará o erro e causará o término da execução do programa de nível convencional responsável pela tentativa de violação.

Quando o indicador de máscara das interrupções for igual a um, o micro-programa não tomará conhecimento da existência de interrupções pendentes (exceto no caso das interrupções por falta de alimentação e retorno de alimentação que não são mascaráveis). Quando o valor deste indicador for zero, o micro-programa tomará conhecimento das interrupções pendentes e as tratará de acordo com seu nível de prioridade.

A modificação do conteúdo dos indicadores de proteção de memória e de modo mestre é controlada pelo campo MMSCO (de um bit) da palavra da memória de executivos. Desta forma os executivos que tiverem este campo com o valor um atualizarão estes indicadores com os valores das linhas de peso 4 (MS) e de peso 16 (PM) da saída da "ULA".

O posicionamento do indicador de máscara das interrupções

é controlado pelo campo NMACO (de um bit) da palavra da memória de executivos. Desta forma os executivos que tiverem este campo com valor um causarão a atualização deste indicador a partir do valor da linha de peso 8 da saída da "ULA".

O posicionamento dos indicadores I1 e I2 é controlado pelo campo NIBC (de três bits) da palavra da memória de executivos de acordo com a tabela apresentada na figura IV.7.

NIDC	I1	I2
0 - 3	Nenhuma alteração	Nenhuma alteração
4	Posicionado a um caso a saída da "ULA" seja zero	Posicionado a um caso a saída da "ULA" seja positiva
5	Posicionado de acordo com o "vai um" da "ULA"	Posicionado a um caso haja "overflow" aritmetico
6	Recebe o valor da linha de peso dois da saída da "ULA"	Recebe o valor da linha de peso um da saída da "ULA"
7	Recebe o valor do indicador de micro-programa SZ	Recebe o valor do indicador de micro-programa B

Figura IV.7. Modificações nos indicadores I1 e I2

o) Os indicadores de micro-programa

A micro-máquina utiliza cinco indicadores disponíveis ao micro-programador. Eles são posicionados de acordo com resultados obti-

dos ao longo da execução das micro-instruções e podem ser testados por estas.

O primeiro deles indica a disponibilidade (ou não) da placa responsável pela execução das operações de multiplicação e divisão por "hardware" (EX).

O segundo indica se o resultado de uma operação da "ULA" foi negativo (SN) enquanto que o terceiro destina-se a indicar se o resultado de uma operação da "ULA" foi zero ("SZ").

O quarto indicador deste grupo informa se o endereço utilizado em uma operação de acesso a memória é ímpar (SO).

O quinto indicador de micro-programa (indicador B) tem uso múltiplo e seu conteúdo é alterado conforme o valor do campo NIBC (de três bits) da palavra da memória de executivos (vide tabela apresentada na figura IV.8.).

p) Interface com os Mecanismos de Entrada e Saída

Esta interface é controlada pelo campo NESC (de dois bits) da palavra da memória de executivos. Se o valor deste campo for zero, nenhuma operação de entrada ou saída é realizada; se for um, é realizada uma operação de leitura de um dado de um periférico. Se o campo contiver o valor dois, a operação realizada é a escrita de um dado em um periférico enquanto que um valor três neste campo causará uma aceitação de pedido de suspensão.

Em todas as operações de entrada e saída o endereço do periférico afetado é especificado pelos quatro bits menos significativos da saída do multiplexador "C", enquanto que os dois bits imediatamente

adjacentes a estes informam o tipo de comando a ser executado pelo periférico.

NIBC	MODIFICAÇÃO DE B
0	Nenhuma modificação
1	Recebe o "vai um" da "ULA"
2	Recebe o valor de I1
3	Recebe o valor de I2
4	Bit mais significativo da saída do multiplexador "D"
5	Bit menos significativo da saída do multiplexador "D"
6	Recebe o valor zero
7	Recebe o valor um

Figura IV.8. Modificações do indicador B

No caso das operações de leitura, os dados fornecidos pelo periférico ficam disponíveis em uma das entradas do multiplexador "A" enquanto que no caso de operações de escrita, os dados utilizados são tomados da saída da "ULA".

q) Interface com os mecanismos de suspensão e interrupção

Os dois últimos bits da palavra da memória de executivos autorizam ou inibem a tomada de conhecimento de interrupções ou suspensões durante a execução da micro-instrução.

É importante observar que este mecanismo é diferente do implementado pelo indicador MA, uma vez que o campo NTITC vai impedir ou permitir (conforme seu valor seja zero ou um) que a execução de uma micro-instrução referenciando o executivo cause a tomada de conhecimento da presença de uma interrupção.

O indicador MA posicionado em um, por outro lado, impedirá que qualquer interrupção possa ser levada em conta, independentemente dos valores dos campos NTITC dos executivos referenciados pelas micro-instruções que vierem a ser executadas, até que o valor deste indicador seja posicionado a zero. Este mecanismo corresponde, portanto, a uma forma de controle residual (6).

O campo NSUCO (de um bit) da palavra da memória de executivos tem, sobre as suspensões, um efeito inverso ao que o campo NTITC tem sobre as interrupções, uma vez que o valor um deste campo de um executivo obrigará a micro-máquina a ignorar qualquer pedido de suspensão durante a execução das micro-instruções que vierem a referenciar este executivo.

r) Interface com a Memória Principal

Como já foi visto, a interface com a memória principal é controlada diretamente por um campo específico da palavra da memória de controle (campo M da palavra de MC 1), permitindo que o tipo de

operação de memória a ser realizada durante a execução de uma micro-instrução independa do tipo de executivo referenciado.

Como a memória principal do MITRA-15 é implementada em núcleos de ferrite, as operações de leitura são destrutivas, i.e. as operações de leitura causam a perda do valor armazenado na posição lida.

Esta característica obriga a que todas as operações de leitura de uma posição da memória principal sejam seguidas de uma operação de reescrita para restaurar o valor previamente lido.

As operações de escrita, por outro lado, requerem a realização de uma prévia operação de leitura da posição de memória a ser modificada a fim de destruir seu valor anterior.

Estas características obrigam a realização de todas as operações de memória em duas etapas.

A primeira destas etapas é sempre uma operação de leitura da posição a ser referenciada, sendo comandada pelo valor um no campo M da palavra de "MC 1".

A segunda etapa será uma reescrita (comandada pelo valor dois no campo M da palavra de "MC 1") ou uma escrita (comandada pelo valor zero no campo M da palavra de "MC 1") na memória principal.

Nas micro-instruções com o campo M igual a três não é realizada nenhuma operação com a memória principal.

Na realização das duas etapas das operações da memória principal são utilizados dois registradores.

O primeiro destes registradores ("S") armazena o ende-

reço da posição de memória a ser utilizada. A entrada de "S" é conectada à saída da "ULA", enquanto que sua saída está ligada ao barramento de endereçamento da memória principal. Como a atualização deste registrador fica habilitada no início da etapa de leitura da memória principal, o valor presente na saída da "ULA" neste momento corresponde ao endereço da posição da memória principal a ser utilizada.

O segundo registrador ("M") armazena o valor lido ou o valor que será escrito na memória principal tendo, para tanto, um acesso bidirecional. Desta forma, este registrador pode receber o valor presente na saída da "ULA" (o que ocorre no início da execução da etapa de escrita na memória) ou o valor lido da memória princpal (o que ocorre durante a execução da etapa de leitura da memória principal).

O valor armazenado em "M" pode ser transferido para a memória principal (o que ocorre tanto na etapa de escrita na memória, quanto na etapa de reescrita) ou pode ser utilizado pela micro-máquina, uma vez que fica disponível em uma das entradas do multiplexador "D".

Deve-se observar que o multiplexador "D" seleciona a entrada correspondente à saída do registrador "M" no final das operações de leitura da memória principal, independentemente do valor do campo da palavra da memória de executivos destinado a controlar este multiplexador (NMUCO), permitindo, assim, o uso imediato do valor lido da memória principal.

Como cada etapa do ciclo de acesso à memória principal tem uma duração maior que o ciclo do relógio, é utilizado um mecanismo que sincroniza a micro-máquina com a memória principal.

Este mecanismo é implementado pelo próprio relógio da micro-máquina (ver ítem a) que aumenta a duração de seu período (in-

cluindo as fases 01 e 02) sempre que o micro-programa comandar operações de memória em micro-instruções consecutivas.

IV.3. MODIFICAÇOES REALIZADAS NO "HARDWARE"

Estas modificações foram levadas a cabo através da construção de uma placa de circuito impresso na qual foi colocada a nova memória de controle, juntamente com uma lógica que permite tanto o carregamento dinÂmico de micro-programas (que são escritos nesta nova memória de controle) quanto a comutação dinÂmica do micro-programa original para micro-programas alternativos armazenados na memória de controle adicionada (e vice-versa).

A nova memória de controle é constituída de 16 pastilhas de memória do tipo "RAM" (de tecnologia "HIIMOS"), sendo que cada pastilha armazena 1024 palavras de um bit.

Estas pastilhas tÊm linhas de entradas e saídas de dados independentes, sendo que as saídas são do tipo coletor aberto.

Como as pastilhas que compõem a memória de controle original também utilizam saídas do tipo coletor aberto, foi possível conectar as saídas das pastilhas que compõem a nova memória de controle diretamente em paralelo com a saída da memória de controle original, sendo relativamente simples controlar a ativação da memória de controle original ou o novo conjunto de memória de controle.

Para permitir a escrita na nova memória de controle, conectou-se as entradas de dados desta memória ao barramento de dados da memória principal do MITRA-15.

Além disto, as dez linhas de endereçamento desta memória

foram conectadas à saída de um multiplexador de duas entradas, construído utilizando pastilhas de tecnologia TTL.

Uma das entradas deste multiplexador foi conectada à saída do registrador "T", enquanto que a outra entrada foi conectada às linhas correspondentes aos dez bits menos significativos do barramento de endereçamento da memória principal do MITRA-15.

Este esquema permite tanto a leitura de micro-instruções da nova memória (quando o multiplexador de endereçamento selecionar a saída do registrador "T"), como o carregamento de micro-programas alternativos (quando o multiplexador de endereços selecionar o barramento de endereçamento da memória principal do MITRA-15).

Para controlar as ações da nova memória, foi incluído na placa um registrador de nove bits cuja entrada foi conectada às linhas correspondentes aos nove bits menos significativos do barramento de dados da memória principal do MITRA-15.

A escrita neste registrador fica habilitada quando as quinze linhas do barramento de endereçamento da memória principal do MITRA-15 tiverem um valor equivalente ao estabelecido por um conjunto de chaves instalados na placa construída e for comandada uma operação de memória.

Desta forma, é possível alterar o valor armazenado neste registrador sob o comando de um programa, através da realização de uma operação de escrita sobre uma posição pré-definida da memória principal.

O bit menos significativo do registrador de controle é utilizado para controlar o carregamento de micro-programas na memória de controle adicionada. Assim, quando o valor deste bit for um o multi-

plexador de endereçamento da memória de controle adicionada selecionará a entrada correspondente ao barramento da memória principal do MITRA-15.

Além disto, quando o valor deste bit for um e as linhas do barramento de endereçamento da memória principal correspondentes aos cinco bits mais significativos tiverem um valor igual ao armazenado nos cinco bits mais significativos do registrador de controle, durante uma operação de acesso à memória principal, a escrita sobre a nova memória de controle é habilitada, permitindo o carregamento de um micro-programa nesta memória.

Os três demais bits deste registrador são utilizados para selecionar a memória de controle a ser utilizada pela micro-máquina. Assim, a micro-memória original será utilizada sempre que uma das seguintes condições for atendida:

- O segundo bit do registrador de controle contiver o valor zero;
- O terceiro bit do registrador de controle contiver o valor zero e o indicador de modo mestre/escravo (MS) estiver na posição correspondente a modo mestre;
- O quarto bit do registrador de controle contiver o valor zero e a micro-máquina estiver tratando uma suspensão.

Este esquema foi concebido de modo a tornar possível a transferência do controle para o micro-programa original durante as operações de entrada e saída.

Isto foi feito porque considerou-se conveniente manter os

"handlers" de entrada e saída originalmente utilizados pelo computador MITRA-15. Como uma parte destes "handlers" foi implementada a nível de micro-código (correspondendo às rotinas de tratamento das suspensões), é necessário dispor de facilidades de retorno automático do controle ao micro-programa original durante a execução das micro-rotinas de tratamento das suspensões, o que é possível quando o quarto bit do registrador de controle armazena o valor zero.

Como a parte dos "handlers" implementada em nível convencional é executada sempre em modo mestre, decidiu-se utilizar o mecanismo implementado com o uso do terceiro bit do registrador de controle, que permite que o controle da micro-máquina retorne ao micro-programa original sempre que o indicador de modo (MS) estiver na posição correspondente a modo mestre.

Deve-se notar que o uso do registrador de controle permite, também, inibir o funcionamento dos mecanismos de comutação automática do controle da micro-máquina supra-citados.

Esta estratégia foi adotada tendo em vista possíveis futuras atividades de pesquisa que envolvessem a construção de novos "handlers" para o computador MITRA-15.

Finalmente, deve-se observar que durante a inicialização da micro-máquina ("RESET") todos os bits do registrador de controle são posicionados com valor zero de forma a permitir que as operações relacionadas à inicialização da micro-máquina sejam realizadas sob controle do micro-programa original.

V. O "SOFTWARE" DE SUPORTE A MICRO-PROGRAMAÇÃO

V.1. CONSIDERAÇÕES GERAIS

Como já foi mencionado, o projeto original do computador MITRA-15 não previa a modificação do micro-programa utilizado, tendo sido necessário, conseqüentemente, a realização de modificações no "hardware" de modo a permitir a alteração dinâmica do micro-programa utilizado (vide capítulo IV).

Como a alteração do micro-programa não estava prevista, o fabricante não forneceu o "software" de suporte para o desenvolvimento de novos micro-programas para esta máquina, tendo sido necessário desenvolvê-lo.

No início das atividades relacionadas à micro-programação do computador MITRA-15 constatou-se a necessidade do desenvolvimento de duas peças de "software" principais.

A primeira, denominada conjunto de geração de micro-código, permite que programas codificados em uma linguagem simbólica sejam transformados nos padrões de bits executáveis pela micro-máquina do computador MITRA-15.

A segunda, denominada de conjunto de suporte à depuração de micro-programas, permite que os micro-programas gerados pela primeira sejam carregados na memória de controle e executados de uma forma conveniente aos propósitos de depuração.

Esta segunda peça de "software" é também responsável pela interface do novo micro-programa com os mecanismos de entrada e saída.

A seguir são apresentadas estas duas peças de "software".

V.2. O CONJUNTO DE GERAÇÃO DE MICRO-CODIGO

Este conjunto é composto por quatro programas e cinco arquivos, sendo que cada um destes programas gera um relatório específico.

A inter-relação desses programas e arquivos pode ser acompanhada no diagrama apresentado na figura V.1.

A seguir são descritos os quatro programas utilizados.

a) O "Micro-assembler" ("MICASB")

Este programa, codificado em FORTRAN, traduz um micro-programa codificado em uma linguagem simbólica (cuja sintaxe está descrita no ANEXO IV) para um código intermediário.

Esta tradução é feita em dois passos, uma vez que são permitidas referÊncias a rótulos ainda não definidos.

O primeiro passo lÊ o micro-programa codificado na linguagem simbólica (do arquivo "MICPGR") e gera um código intermediário (que fica armazenado no arquivo "CODINT"), além de construir uma tabela de símbolos (que fica armazenada no arquivo "TABSIM").

O arquivo "MICPGR" é constituído por até cinco mil registros de oitenta "bytes" e não utiliza campos fixos.

O arquivo "TABSIM" é constituído por 4001 registros de 32 "bytes" organizados em blocos de oito registros (256 "bytes"). Cada registro armazena um rótulo, sendo o acesso a estes registros realizados utilizando uma função de "hash" do nome do rótulo.

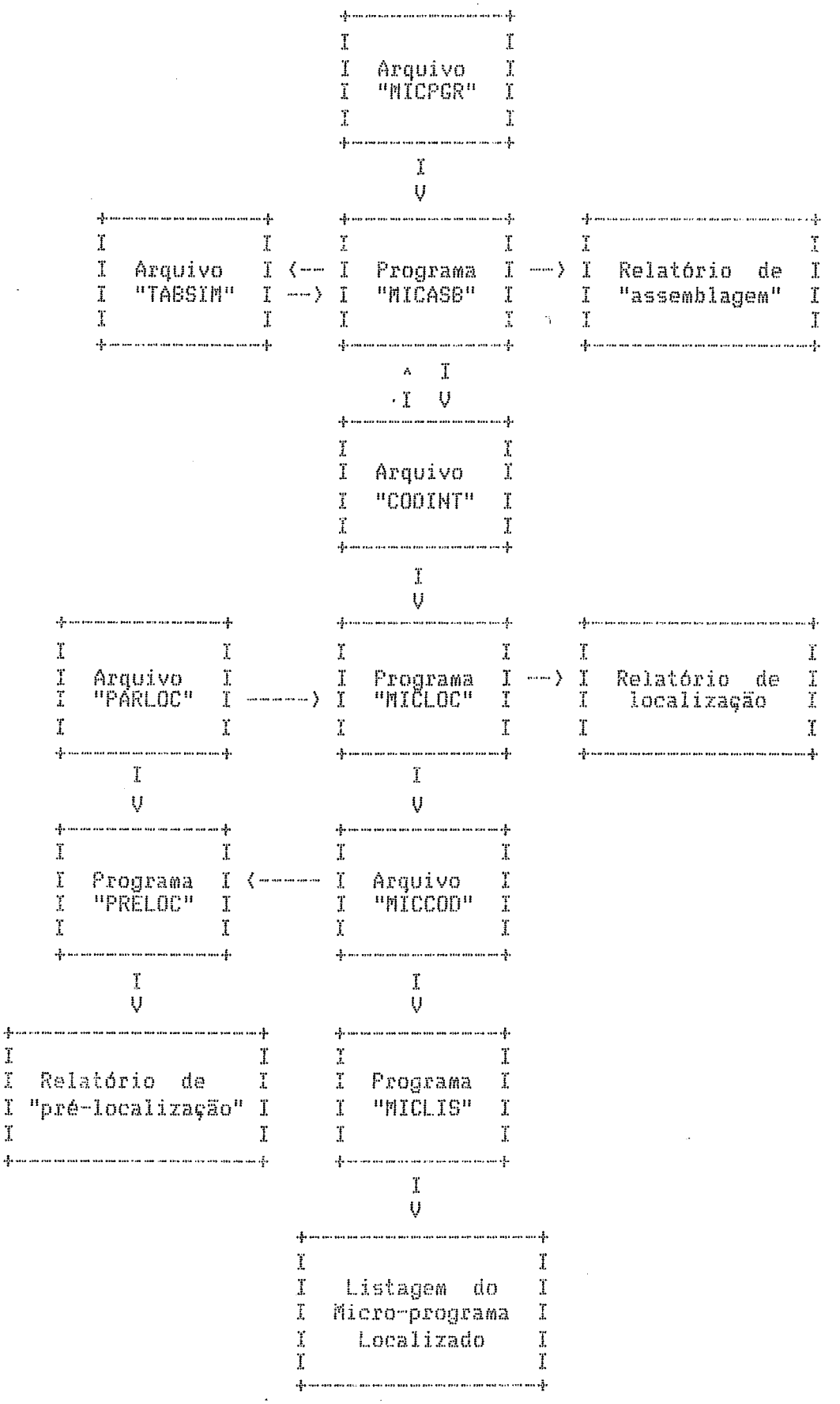

Figura V.1. O conjunto de geração de micro-código

O arquivo "CODINT" é formado por até 1024 registros de tamanho variável cujo formato ao final do primeiro passo está apresentado na figura V.2.

NOME DO CAMPO	POSIÇÃO ("BYTES")	CONTEÚDO
LINHA	0 - 1	Número da linha onde começa a micro-instrução
CODIGO	2 - 3	Código da micro-instrução (com parâmetros zerados)
PARAM	4 - 5	Valor do parâmetro ou apontador para o símbolo + 4096
SUCMAS	6 - 7	Número da micro-instrução sucessora, valor da máscara ou apontador para símbolo + 4096
NLIN	8 - 8	Número de linhas ocupadas pela micro-instrução
NMULT	9 - 9	Número de sucessoras múltiplas (de 0 a 64, vide modificação de "T" com mascaramento em IV.1.)
MULT	10 - 138	Número de micro-instrução sucessora múltipla ou apontador para símbolo + 4096 (ocorre NMULT vezes)

Figura V.2. Formato dos registros do arquivo "CODINT" ao final do primeiro passo.

No segundo passo é dada uma forma final ao código intermediário (que fica ainda armazenado no arquivo "CODINT", no formato apresentado na figura V.3.), ao mesmo tempo em que são resolvidas as

referências aos rótulos utilizados antes de sua definição.

NOME DO CAMPO	POSIÇÃO ("BYTES")	CONTEÚDO
LINHA	0 - 1	Número da linha onde começa a micro-instrução
CODIGO	2 - 3	Código da micro-instrução com parâmetro para modificação normal de "T" zerado
SUCNOR	4 - 5	Número da micro-instrução sucessora normal
MASC	6 - 7	Máscara utilizada nas modificações de "T" com mascaramento
TIPO	8 - 8	Tipo de indicação de sucessora empregado pela instrução
NMULT	9 - 9	Número de sucessoras múltiplas
MULT	10 - 138	Número de micro-instrução sucessora múltipla (ocorre NMULT vezes)

Figura V.3. Formato dos registros do arquivo "CODINT" ao final do segundo passo

Este programa gera um relatório composto por três partes, sendo que quaisquer destas partes (ou todas) podem ser omitidas (através do uso de parâmetros de execução do programa).

A primeira parte é gerada pelo primeiro passo, sendo constituída de uma listagem do micro-programa sob a forma simbólica, na qual são apontados eventuais erros sintáticos, sendo que as linhas que

contém erros são listadas independentemente do parâmetro de execução utilizado.

A segunda parte do relatório é gerada ao final da execução do primeiro passo, sendo constituída de uma listagem da tabela de símbolos, onde são relacionados todos os rótulos definidos ao longo do micro-programa, acompanhados dos seus valores.

A terceira parte do relatório é gerada pelo segundo passo, sendo constituída de uma listagem do micro-código gerado, em um formato legível. Nesta listagem são também apontados eventuais erros semânticos, sendo que as micro-instruções nas quais forem detetados tais erros são listadas independentemente do parâmetro de execução utilizado.

Os problemas associados ao seqüenciamento da execução das micro-instruções não são resolvidos por este programa. Desta forma o arquivo "CODINT" armazena as micro-instruções com os campos reservados para os parâmetros das modificações normais e restritas de "T" zerados.

Como pode ser visto na figura V.3, os registros deste arquivo contém também campos destinados a especificar os números das micro-instruções sucessoras, juntamente com outras informações destinadas a facilitar a identificação da micro-instrução (número da linha na qual ela aparece no micro-programa em formato simbólico) e a simplificar a resolução do problema de seqüenciamento (tipo de sucessora).

b) O "Micro-localizador" ("MICLOC")

Este programa, codificado em "assembly", resolve o problema de seqüenciamento da execução do micro-programa.

Compete a ele encontrar uma ordenação para a implantação

das micro-instruções que compõem o micro-programa na memória de controle de tal forma que a execução deste micro-programa seja realizada na ordem desejada.

Durante a definição deste programa foi constatada a impossibilidade da análise de todas as possíveis ordens de implantação das micro-intruções na memória de controle, pois seria necessário considerar 1024! possibilidades, no pior caso.

Desta forma foi necessário conceber um algoritmo que eliminasse rapidamente um grande nÓmero de permutações inviáveis e considerasse primeiramente as famílias de possíveis soluções que tivessem maiores possibilidades de conter uma solução viável.

Na definição deste algoritmo foram levados em consideração a questão da localidade bem como a tendÊncia dos programadores a construirem micro-programas nas quais, na maior partes dos casos, as micro-instruções que devem ser executadas seqÜencialmente aparecem em ordem consecutiva (porque na maior parte das linguagens de programação a execução do programa segue a seqÜÊncia na qual o programa foi escrito, o que tenderia a acostumar os programadores a utilizarem tal forma de apresentação).

Assim, o algoritmo utilizado pelo micro-localizador realiza uma varredura das micro-instruções que compõem o micro-programa na ordem na qual estas micro-instruções aparecem no micro-programa.

Quando uma micro-instrução está sendo varrida, o micro-localizador tenta obter endereços de implantação para esta micro-instrução e para todas as suas sucessoras ainda não tratadas (i.e., que apareçam depois da micro-instrução em questão e que não sejam sucessoras de outras que apareceram antes) de forma a permitir a execução destas

micro-instruções na seqüência especificada.

Ao calcular estes endereços de implantação, este programa leva em consideração as posições da micro-memória que já estão ocupadas, bem como as possibilidades de variação dos parâmetros para modificação normal ou restrita de "T" (caso o executivo referenciado pela micro-instrução em questão utilize um destes tipos de modificação de "T").

Além disto, este programa verifica ainda se as sucessoras da micro-instrução que está sendo tratada utilizam modificações implícitas de "T" ou modificações de "T" com mascaramento. Nestes casos o micro-localizador verifica também se estas micro-instruções sucessoras podem atingir suas sucessoras a partir do endereço de implantação que lhes tenham sido atribuídos. Isto é feito porque, no caso das micro-instruções que utilizem estas formas de modificação de "T", o endereço de implantação da sucessora fica perfeitamente determinado a partir da fixação do endereço da micro-instrução em questão.

Ao atingir uma micro-instrução que, pelos critérios acima estabelecidos, não pode ser implantada em nenhuma das posições da micro-memória ainda não utilizadas, o algoritmo passa a procurar a última micro-instrução varrida cujo endereço de implantação possa ser alterado, ou que referencie executivo que utilize modificação normal ou restrita de "T" cujo parâmetro possa ser alterado.

Para que uma micro-instrução possa ter seu endereço de implantação alterado é necessário que nenhuma de suas predecessoras apareça no micro-programa em posição anterior a dela e que o seu endereço de implantação atual não corresponda ao da última posição da memória de controle.

Para que seja possível alterar o parâmetro para modificação normal ou restrita de uma micro-instrução que referencie um executivo que faça uso de um destes tipos de modificações de "T", é necessário que o valor atual deste parâmetro seja menor que seu valor máximo (31 no caso de modificação normal de "T" ou 7 no caso de modificação restrita de "T").

Tão logo o micro-localizador consiga determinar qual micro-instrução atende um dos requisitos supra-mencionados, o endereço de implantação, ou o parâmetro para modificação normal de "T" desta micro-instrução é incrementado e o micro-localizador volta a realizar a varredura a partir desta micro-instrução.

O relatório gerado pelo programa micro-localizador é composto por mensagens que indicam quais micro-instruções foram varridas pelo menos uma vez. Desta forma, cada vez que a varredura avança de uma micro-instrução, uma mensagem é impressa, permitindo que o micro-programador saiba, a qualquer momento, o estágio no qual se encontra o processo de localização das micro-instruções. Estas mensagens incluem a hora em que foram impressas, de forma a permitir um acompanhamento do desempenho do algoritmo.

O algoritmo empregado pelo micro-localizador não resolve todos os problemas de seqüenciamento da execução das micro-instruções do micro-programa responsável pela implementação da máquina básica EDISON.

Assim, foi desenvolvido um esquema que permite ao micro-programador interagir com o processo de localização (através do programa pré-localizador) de forma a resolver os problemas de seqüenciamento remanescentes.

c) O "Pré-localizador" ("PRELOC")

O esquema desenvolvido para a resolução dos problemas de seqüenciamento que o micro-localizador não é capaz de solucionar incluiu o uso de um recurso que permite interromper a execução deste programa no momento em que sua execução atinja um ponto a partir do qual o algoritmo não pode prosseguir.

Este ponto é detetado (pelo micro-programador) quando, durante um intervalo de tempo muito grande (maior que uma hora), o micro-localizador não imprime nenhuma mensagem indicativa de primeira varredura de uma micro-instrução.

Neste ponto, o micro-programador deve acionar uma das chaves do painel do computador MITRA-15 para comandar o armazenamento do resultado parcial atingido (no arquivo "MICCOD") de tal forma que seja possível realizar, manualmente, pequenas alterações na solução que está sendo considerada, de forma a superar o problema encontrado.

Uma vez realizada esta intervenção, é possível re-iniciar a execução do programa micro-localizador a partir do ponto em que foi interrompido.

O programa pré-localizador foi concebido de forma a facilitar a intervenção do micro-programador sobre a solução do problema de seqüenciamento tratado pelo programa micro-localizador.

Assim, quando o micro-localizador atinge o ponto a partir do qual não consegue prosseguir, o micro-programador pode interromper a execução deste programa e analisar os resultados já obtidos (que podem ser detalhadamente listados utilizando o programa micro-listador) de

forma a detetar as causas da dificuldade encontrada.

Uma vez determinada a dificuldade, é possível (e na maior parte dos casos, muito simples) proceder pequenas alterações na solução que está sendo analisada pelo micro-localizador que permitam o prosseguimento da execução deste programa.

O programa pré-localizador permite que o micro-programador informe ao programa micro-localizador que posições atingidas por determinadas micro-instruções devem ser utilizadas como ponto de partida.

É possível também informar que algumas micro-instruções já atingiram uma posição conveniente, e que o micro-localizador deverá considerá-las como fixadas (que poderão, inclusive, ser diferentes das posições previamente calculadas pelo micro-localizador).

Por outro lado, é também possível informar que a posição atribuída pelo micro-localizador a uma micro-instrução é inconveniente, sendo portanto necessária a realização de uma busca de uma nova posição para a micro-instrução em questão.

Todas as informações mencionadas nos parágrafos anteriores podem ser passadas através de um diálogo entre o micro-programador e o programa pré-localizador (realizado através de um teletipo). O programa pré-localizador então realiza uma análise das informações recebidas e as transfere ao programa micro-localizador através do arquivo "PARLOC".

Este arquivo é composto por 1024 registros de quatro "bytes" (duas palavras), sendo que cada um destes registros corresponde a uma micro-instrução. Os primeiros dois "bytes" destes registros

armazenam a posição a ser ocupada pela micro-instrução, enquanto que os outros dois "bytes" destinam-se a armazenar o número da sua primeira micro-instrução predecessora.

Registros que tiverem posições com valores negativos são considerados livres (micro-instruções não localizadas ou com posições não aproveitadas). Quando a posição for nula ou positiva, o micro-localizador aproveitará esta posição como ponto de partida na próxima execução, sendo que, caso o número da instrução predecessora seja zero, o micro-localizador considerará a micro-instrução em questão fixada.

O programa pré-localizador pode emitir, ainda, (a pedido, através do diálogo que é mantido com o micro-programador) um relatório que indica o estado em que se encontram cada uma das micro-instruções (estes estados podem ser NÃO LOCALIZADA, LOCALIZADA, APROVEITADA ou FIXADA).

d) O "Micro-listador" ("MICLIS")

Este programa, codificado em FORTRAN, destina-se a fornecer um relatório completo do estado da localização do micro-programa, além de reproduzir o formato simbólico original das micro-instruções.

Para tanto, este programa lê o arquivo "MICCOD" (gerado pelo programa micro-localizador) que é composto por quatro registros de 1024 palavras (2048 "bytes") onde cada uma destas palavras corresponde a uma micro-instrução.

O primeiro registro armazena os códigos das micro-instruções (já localizadas), na ordem em que estas micro-instruções devem ser implantadas na memória de controle.

No segundo registro, cada palavra corresponde a uma micro-instrução, na ordem em que estas aparecem no micro-programa em formato simbólico, e indica a posição ocupada pela micro-instrução na memória de controle.

O terceiro registro armazena as linhas correspondentes a cada uma das micro-instruções e está ordenado da mesma forma que o segundo.

O quarto registro, que também está ordenado da mesma forma que o segundo, armazena os números das primeiras micro-instruções predecessoras de cada uma das micro-instruções que compõem o micro-programa.

A partir destas informações o programa micro-listador monta uma tabela que indica o número (e linha) da micro-instrução que ocupa cada uma das posições da memória de controle. Esta tabela constitui a primeira parte do relatório gerado por este programa.

A segunda parte do relatório é uma reconstituição do micro-programa em formato simbólico, acompanhado dos endereços de implantação de cada uma das micro-instruções e de suas sucessoras, além da apresentação (em binário) do código da micro-instrução.

Este relatório foi útil tanto na análise das intervenções a serem executadas durante o processo de localização do micro-programa, como para validar o conjunto de geração de micro-código.

V.3. O CONJUNTO DE DEPURAÇÃO DE MICRO-PROGRAMAS

Este conjunto é formado por apenas um programa, codificado em "assembly", denominado "MICMON".

Este programa carrega o micro-programa na memória de controle, carrega um arquivo de código que contenha instruções executáveis pela máquina básica EDISON e a seguir executa normalmente, passo a passo ou com pontos de parada ("breakpoints"), o programa EDISON carregado.

São fornecidas, ainda, facilidades para examinar e modificar quaisquer posições da memória principal (sendo que o endereço da posição a examinar ou modificar pode ser especificado de forma absoluta, em relação ao início da região de trabalho deste programa, ou em relação ao início da área ocupada pelo código do programa EDISON).

De forma semelhante, é possível examinar e modificar os valores dos registradores da máquina virtual, o contexto de um processo, ou variáveis quaisquer de um processo (especificadas pelo nível de aninhamento estático da rotina as quais são locais e deslocamento, vide item II.3.2.).

Finalmente este programa é responsável também pela realização das operações de entrada e saída especificadas em programas EDISON (vide item VI.5.).

VI. A MAQUINA BASICA MICRO-PROGRAMADA

VI.1. O MAPEAMENTO DA MAQUINA BASICA NA MICRO-MAQUINA

Dos 64 registradores que compõem o banco de registradores da micro-máquina (vide item IV.2.), 56 são reservados para os "handlers" de entrada e saída que não foram alterados. Assim, apenas 8 registradores foram utilizados pelo micro-programa que realiza a máquina básica.

O primeiro destes registradores foi alocado para armazenar o apontador de programa ("PC") e o segundo foi empregado para armazenar o endereço da tabela de bases ("BTBASE").

O terceiro aponta para uma área de comunicação com o programa (codificado na linguagem "assembly" original do computador MITRA-15) responsável pelas operações de entrada e saída especificadas nos programas EDISON (programa "MICMON" - vide itens V.2. e VI.5.) e denomina-se "MONITORBASE".

Os dois registradores seguintes (quarto e quinto) foram utilizados para armazenar resultados parciais obtidos ao longo da execução das instruções da máquina básica, bem como para armazenar os operandos imediatos ("DSP" e "LV") destas instruções.

O sexto registrador é o apontador de pilha ("SP") e o sétimo registrador armazena o código da instrução que está sendo executada ("IR").

O óltimo registrador armazena o operando imediato "SZ" (tamanho de cadeia) das intruções.

Deve-se observar que não foi possível armazenar o valor

do registrador "BASE" (base local) na memória local da micro-máquina.

Na realidade, quando foi constatada a impossibilidade de alojar todos os registradores do interpretador mapeados nos registradores da micro-máquina, resolveu-se deslocar este registrador para a memória principal porque ele era o menos freqüentemente acessado.

Com o mapeamento deste registrador na memória principal, parte dos ganhos obtidos pelo uso da modalidade local de endereçamento foram perdidos (vide ítem II.3.2.). A perda não foi, porém, completa porque as instruções que empregam a modalidade de endereçamento local não utilizam o operando imediato correspondente ao nível, economizando, assim, o ciclo de acesso à memória necessário para a leitura do operando imediato.

Finalmente, cabe assinalar que não foi possível utilizar nenhum esquema de proteção de memória em virtude das limitações do tamanho da memória de controle e do número de registradores disponíveis, bem como pelas dificuldades no uso de micro-rotinas na arquitetura empregada.

VI.2. A ESTRUTURA DO MICRO-PROGRAMA

O micro-programa construído (cuja listagem encontra-se no ANEXO VI) pode ser dividido em onze partes principais:

a) busca e pré-decodificação da instrução;

b) interpretação das instruções de cálculo de endereços de variáveis e consulta e modificação de valores de variáveis;

c) interpretação das instruções aritméticas e lógicas;

d) interpretação das instruções de comparação;

e) interpretação das instruções relacionadas a arranjos;

f) interpretação das instruções relacionadas a conjuntos;

g) interpretação das instruções relacionadas a constantes e desvios;

h) interpretação das instruções relacionadas a rotinas;

i) interpretação das instruções do núcleo;

j) escalonamento dos processos;

l) interpretação das instruções destinadas a implementar as operações especiais.

A primeira destas partes - busca e pré-decodificação da instrução - é responsável pela leitura do código da próxima instrução a ser executada seguida do desvio do fluxo de controle para uma das demais partes (com exceção da parte j - escalonamento de processos - que só é utilizada durante a execução de instruções do núcleo).

As partes de b a h correspondem às diversas categorias de instruções executadas pelo interpretador, enquanto que as partes i e j correspondem às tarefas do núcleo.

A última parte é responsável pela interpretação das operações especiais e pela interface com os mecanismos de entrada e saída que serão posteriormente discutidos (vide ítem VI.5.).

VI.3. A ARVORE DE DECODIFICAÇAO DAS INSTRUÇOES

Os códigos de operações das instruções que compõem a máquina básica ocupam os sete bits menos significativos do "byte" mais significativo das palavras da memória principal do MITRA-15.

Como a máquina básica definida utiliza apenas 99 instruções das 128 combinações possíveis, algumas instruções podem ser refe-

renciadas utilizando mais de uma combinação. Além disto, algumas instruções que não estavam previstas no conjunto originalmente adotado para a máquina básica EDISON, foram também introduzidas afim de simplificar a estrutura de decodificação das instruções. No ANEXO VII estão relacionadas todas as instruções utilizadas nesta implementação, juntamente com os valores dos códigos de operação a elas correspondentes e os tempos necessários para executá-las.

Os sete bits que compõem o código das instruções foram divididos em três campos.

O primeiro campo, denominado TIPO, corresponde ao bit mais significativo (peso 64), indica se a instrução utiliza ou não o esquema de endereçamento discutido no ítem II.2.3. O valor zero neste bit indica que este esquema será utilizado (instruções das categorias a, b e c discutidas no ítem II.2.4.).

O segundo campo, denominado CATEGORIA, corresponde aos três bits imediatamente adjacentes ao primeiro campo (bits de peso 32, 16 e 8) e informa a categoria a qual pertence a instrução.

O último campo, denominado ENDER ou OPERAÇÃO (de acordo com o valor do campo TIPO), corresponde aos três bits menos significativos do código da instrução. No caso das instruções que utilizam um endereço (campo TIPO igual a zero), este campo informa o tipo de endereçamento utilizado, enquanto que nas demais instruções, este campo informa a operação específica que deverá ser realizada.

A figura VI.1. apresenta um resumo do significado dos campos do código da operação enquanto que a figura VI.2. apresenta o significado dos bits que formam o campo ENDER.

Nome do Campo	Posição (bit)	Significado
TIPO	0	O valor zero indica o emprego do esquema de endereçamento discutido no item II.2.3.
CATEGORIA	1 - 3	Informa a categoria a qual pertence a instrução
ENDER. ou OPERAÇAO	4 - 6	Informa o tipo de endereçamento utilizado ou a operação específica a ser realizada

Figura VI.1. Significado dos campos do código de operação

BIT	Significado
0	Tipo de base empregada (0 = local, 1 = global)
1	Indireção (0 = endereçamento direto, 1 = endereçamento indireto)
2	Indexação (0 = endereçamento não indexado 1 = endereçamento indexado)

Figura VI.2. Significado dos bits que formam o campo ENDER

A interpretação das instruções é iniciada pela incrementação do apontador de programa ("PC") simultaneamente à leitura do código da instrução a ser executada.

Tão logo o código desta instrução esteja disponível, é realizado um desvio condicionado pelo valor do campo TIPO. Se o valor deste campo for zero, será construído o endereço do operando a ser utilizado. Caso contrário, o interpretador conclui que está sendo executada uma instrução de zero endereços.

No primeiro caso, o micro-programa lê o operando imediato correspondente ao deslocamento ("DSP") da variável referenciada e a seguir realiza um desvio condicionado pelo valor do bit menos significativo do campo ENDER, para determinar que tipo de base deve ser considerada na formação do endereço da variável.

Se o valor deste bit for zero, a base é a local, enquanto que o valor um deste bit indica a utilização de uma base global. Neste segundo caso, é lido um operando imediato (que indica o nível de aninhamento da rotina a qual a variável é local). À seguir, é lida entrada da tabela de bases correspondente a este nível de aninhamento.

Uma vez obtida a base, esta é adicionada ao deslocamento ("DSP") de modo a obter o endereço primário, ao mesmo tempo que é realizado um desvio condicionado pelo valor do segundo bit do campo ENDER, para determinar se o endereçamento é direto ou indireto.

Desta forma, caso o valor deste bit seja um, o endereçamento é considerado indireto. Neste caso, o valor do endereço secundário (vide ítem II.2.3) é lido da posição de memória apontada pelo endereço primário.

À seguir é realizado um desvio condicionado pelo valor dos dois bits menos significativos do campo CATEGORIA. O significado das diversas combinações destes bits pode ser acompanhado na figura VI.3.

Valor do bit de peso 2 do campo CATEGORIA	Valor do bit de peso 1 do campo CATEGORIA	Tipo de operação realizada
0	0	Cálculo de endereço de variável
0	1	Consulta ou atribuição de valor a variável que ocupe um "byte"
1	0	Consulta ou atribuição de valor a variável que ocupe dois "bytes"
1	1	Consulta ou atribuição de valor a variável que ocupe um número arbitrário de "bytes"

Figura VI.3. Significado das combinações de valores dos dois bits mais significativos do campo CATEGORIA nas instruções com o campo TIPO com valor um.

Assim, se o valor destes dois bits for zero, a operação a ser realizada será somente o cálculo do endereço de uma variável.

Como o endereço secundário já está disponível, é suficiente verificar se o modo de endereçamento empregado utiliza indexação.

Para isto é feito um desvio condicionado pelo valor do bit mais significativo do campo ENDER. Se o valor deste bit for um, é realizada a leitura do valor armazenado no topo da pilha, que é, a seguir, adicionado ao valor do endereço secundário de forma a obter o endereço efetivo. Caso o valor deste bit seja zero, o endereço efetivo será o próprio endereço secundário. Em ambos os casos, o endereço efeti-

vo é colocado no topo da pilha, sendo que neste ponto é concluída a interpretação da instrução.

Na segunda combinação de valores dos dois bits mais significativos do campo CATEGORIA, a operação a ser realizada envolverá uma variável cujo tamanho corresponda a um "byte".

Para realizar tais operações, é colocado o valor um no registrador reservado ao armazenamento do tamanho das cadeias ("SZ") e feito um desvio no fluxo de controle do micro-programa de forma a permitir que uma seqüência de micro-instruções que é responsável pelo tratamento de cadeias seja executada.

Na terceira combinação de valores dos dois bits mais significativos do campo CATEGORIA, a operação a ser realizada envolverá uma variável cujo tamanho corresponda a dois "bytes" (uma palavra).

Como estas operações são muito freqüentes (uma vez que as variáveis inteiras e de tipo enumeração ocupam uma palavra), optou-se por realizá-las através da execução de uma seqüência de micro-instruções específicas, ao invés de inicialmente colocar o valor dois no registrador que armazena o tamanho da cadeia ("SZ") e, em seguida, desviar para a seqüência de micro-instruções responsável pelo tratamento de cadeias.

A seqüência de micro-instruções especificamente destinada a tratar as operações que envolvem variáveis cujos tamanhos correspondem a dois "bytes" é iniciada por um desvio condicionado pelo valor do bit menos significativo do campo CATEGORIA em conjunto com o valor do bit mais significativo do campo ENDER.

Isto é feito porque o bit mais significativo do campo

ENDER indica se o endereçamento é indexado ou não, enquanto que o bit menos significativo do campo CATEGORIA indica se a operação a ser realizada corresponde a uma consulta ou a uma atribuição a uma variável.

Estes dois bits são decodificados simultâneamente porque a posição na pilha da qual deverá ser lido o índice empregado na construção do endereço efetivo varia de acordo com o tipo de operação a ser executada (nos casos de atribuição de valores a variáveis, o índice deve ser lido da posição da pilha abaixo da qual será lido o valor a ser atribuído à variável).

Assim, as quatro combinações possíveis dos valores destes bits são tratadas por quatro diferentes seqüências de micro-instruções. No final de cada uma destas seqüências, há um retorno para o início da do micro-programa para permitir a interpretação da próxima instrução.

Na última combinação de valores dos dois bits mais significativos do campo CATEGORIA, a operação é de consulta ou atribuição de valor a uma variável que ocupe um número arbitrário de "bytes".

Neste caso, o micro-programa realiza a leitura do operando imediato destinado a indicar o tamanho da variável a ser considerada, sendo este valor armazenado no registrador "SZ".

A seguir o fluxo de controle é desviado para a seqüência de micro-instruções responsável pelo tratamento de cadeias.

Esta seqüência é iniciada por um desvio condicionado pelo valor do bit mais significativo do campo ENDER em conjunto com o valor do bit menos significativo do campo CATEGORIA.

Como já foi visto, estes bits destinam-se a indicar o

tipo de operação a ser realizada (consulta ou atribuição de valor) e se o endereçamento é indexado ou não, sendo decodificados simultaneamente porque a posição da pilha da qual deve ser lido o índice varia de acordo com o tipo de operação (uma vez que no caso das atribuições de valor a variáveis, o índice deve ser lido de uma posição da pilha abaixo das posições que armazenam o valor a ser atribuído à variável).

Assim, se a instrução corresponde a uma consulta com endereçamento não indexado, o endereço efetivo da variável é o próprio endereço secundário.

Desta forma, o fluxo de controle do micro-programa pode ser diretamente desviado para a execução de um "loop", controlado pelo registrador "SZ" (que é decrementado a cada iteração), que realiza a cópia dos "bytes" que compõem a variável para o topo da pilha. Quando o valor do registrador "SZ" for nulo, o micro-programa verifica se o valor de "SP" é ímpar, adicionando uma unidade a este registrador, neste caso (este apontador deve armazenar sempre um endereço de palavra, que no MITRA-15 é um número par). Neste ponto a interpretação da instrução é concluída.

No caso de instruções de consulta a variveis com endereçamento indexado, é realizada a leitura do valor do índice (do topo da pilha), sendo este índice adicionado ao endereço secundário de forma a gerar o endereço efetivo. A seguir, o fluxo de controle é desviado para o "loop" mencionado no parágrafo anterior.

No caso das instruções de atribuição de valores à variáveis com endereçamento não indexado, o apontador de pilha ("SP") é decrementado de um valor igual ao tamanho da variável (armazenado no rgistrador "SZ"). Se este tamanho for ímpar, mais uma unidade é

subtraída do apontador de pilha de modo a garantir que este apontador manterá um endereço de palavra.

A seguir, o fluxo de controle é desviado para a execução de um "loop" (controlado pelo registrador "SZ", que é decrementado a cada iteração) que realiza a cópia da cadeia armazenada a partir da posição apontada pelo apontador de pilha ("SP") para a variável considerada. Quando o valor do registrador "SZ" atinge zero está concluída a execução da instrução.

No caso de instruções de atribuição à variáveis com endereçamento indexado, o apontador de pilha ("SP") é também decrementado de um valor igual ao tamanho da variável (sendo também subtraída mais uma unidade deste apontador no caso de variáveis com tamanhos ímpares). A seguir, o índice é retirado do topo da pilha e adicionado ao endereço secundário de forma a construir o endereço efetivo da variável.

A seguir, ocorre um desvio para o "loop" mencionado na implementação das atribuições a variáveis com endereçamento não indexado. Neste caso, porém, a cadeia que será copiada é aquela apontada pelo apontador do topo da pilha adicionado de duas unidades (para que o índice não seja copiado para a variável).

Como já foi visto anteriormente, quando o bit correspondente ao campo TIPO tem o valor um, o interpretador considera que está sendo executada uma das instruções de zero endereços.

Nestes casos é realizado um desvio condicionado pelo valor do campo CATEGORIA para uma seqüência específica de micro-instruções. A tabela apresentada na figura VI.4. associa os possíveis valores deste campo com as partes do micro-programa que são invocadas.

Valor do campo CATEGORIA	Parte do micro-programa invocado
0	Interpretação das instruções aritméticas e lógicas
1	Interpretação das instruções de comparação
2	Interpretação das instruções relacionadas a arranjos
3	Interpretação das instruções relacionadas a conjuntos
4	Interpretação das instruções relacionadas a constantes e desvios
5	Interpretação das instruções relacionadas a rotinas
6	Interpretação das instruções do núcleo
7	Interpretação das instruções especiais

Figura VI.4. Parte do micro-programa invocado de acordo com o valor do campo CATEGORIA

A seguir estão descritos os passos que compõem a decodificação das instruções levadas a cabo por cada uma das partes relacionadas na figura VI.4.

a) A interpretação das instruções aritméticas e lógicas

A interpretação destas instruções é iniciada pela leitura dos dois operandos que serão utilizados por estas instruções. Estes operandos encontram-se no topo da pilha e são deslocados para os dois registradores de trabalho.

A seguir, o micro-programa comanda o início da primeira fase da operação de escrita na memória (na posição correspondente ao topo da pilha) do resultado da operação e realiza um desvio no fluxo de controle condicionado pelo valor do campo OPERAÇÃO.

A tabela apresentada na figura VI.5. indica o tipo de operação correspondente a cada um dos possíveis valores do campo OPERAÇÃO.

As seqüências de micro-instruções que interpretam estas instruções terminam com a própria conclusão da execução da instrução, sendo que as seqüências responsáveis pela realização da soma, diferença, conjunção ("ou" lógico) e dijunção ("e" lógico) são compostas por apenas uma micro-instrução.

A seqüência responsável pela realização da negação do valor booleano armazenado no topo da pilha, por utilizar apenas um operando, é iniciada pela movimentação do valor previamente lido do topo da pilha como segundo operando de volta para o topo da pilha.

b) Interpretação das instruções de comparação

Da mesma forma que no caso das instruções aritméticas e lógicas, a parte do micro-programa que realiza as instruções de comparação é iniciada pela leitura dos dois operandos (do topo da pilha),

seguido de uma preparação para o armazenamento do resultado da operação (no topo da pilha).

Valor do campo OPERAÇÃO	Operação realizada
0	Soma
1	Diferença
2	Multiplicação
3	Divisão
4	Resto da divisão inteira
5	Negação
6	Conjunção ("OU" lógico)
7	Dijunção ("E" lógico)

Figura VI.5. Operações possíveis no caso de interpretação de instruções aritméticas ou lógicas

A seguir o micro-programa realiza um desvio condicionado pelo resultado da comparação dos dois valores lidos.

Se estes dois valores forem iguais é realizado um deslo-

camento de um bit para a esquerda do valor do código da instrução. Se o segundo valor lido da pilha for menor que o primeiro são realizados dois deslocamentos de um bit para a esquerda do valor do código da instrução.

A seguir, independentemente do resultado da comparação, é realizado um desvio no fluxo de controle do micro-programa condicionado pelo valor do bit mais significativo do campo OPERAÇAO.

Se o valor deste bit for zero, o valor booleano "falso" é colocado no topo da pilha. Caso contrário, o valor booleano "true" é colocado no topo da pilha.

Como o bit do código da instrução efetivamente testado é modificado de acordo com o resultado da comparação (em virtude dos eventuais deslocamentos realizados), este esquema permite testar todas as combinações de comparações previstas no conjunto de instruções da máquina básica EDISON.

A figura VI.6. apresenta as condições necessárias para que o resultado da comparação corresponda ao valor booleano "true" para cada um dos possíveis valores do campo OPERAÇAO.

c) Interpretação das instruções relacionadas a arranjos

A interpretação das instruções desta categoria é iniciada pela leitura do operando imediato "SZ" (tamanho da cadeia) que é colocado no registrador destinado a armazená-lo (vide item VI.1.).

A seguir o fluxo de controle do micro-programa sofre um desvio condicionado pelo valor dos dois bits mais significativos do campo OPERAÇAO.

Valor do campo OPERAÇÃO	Condição necessária para a obtenção do resultado "true" para a comparação
0	O resultado é sempre "false"
1	Primeiro operando da comparação menor que o segundo
2	Primeiro operando da comparação igual ao segundo
3	Primeiro operando da comparação menor ou igual ao segundo
4	Primeiro operando da comparação maior que o segundo
5	Primeiro operando da comparação diferente do segundo
6	Primeiro operando da comparação maior ou igual ao segundo
7	O resultado é sempre "true"

Figura VI.6. Condições necessárias para a obtenção de resultados "true" nas comparações

As operações de comparação de cadeias são realizadas por um "loop" controlado pelo registrador "SZ" (que é decrementado a cada iteração). Desta forma, os "bytes" das duas cadeias são comparados um a um. Quando é encontrado um "byte" diferente, o micro-programa considera

que as cadeias são diferentes e armazena o valor booleano "false" em um registrador de trabalho. Caso o valor do registrador "SZ" atinja zero, então é o valor booleano "true" que é armazenado.

Independentemente do resultado da comparação, o micro-programa faz um desvio condicionado pelo bit menos significativo do campo OPERAÇÃO. Se o valor deste bit for um, o resultado da comparação é complementado.

A seguir, independentemente do resultado deste último teste, o valor armazenado neste registrador de trabalho é colocado no topo da pilha.

Esta estratégia permite a realização das duas operações de comparação de cadeias da máquina básica.

Se a operação a ser realizada for a movimentação de uma cadeia constante para o topo da pilha, o micro-programa utiliza um "loop" controlado pelo registrador "SZ" (cujo valor é decrementado a cada iteração) que é responsável pela cópia ("byte" a "byte") desta cadeia.

Quando o valor do registrador "SZ" atinge zero, o micro-programa verifica se o apontador de pilha tem um valor ímpar, colocando na pilha, neste caso, um último "byte" (com valor zero) de modo a manter um endereço de palavra no apontador de pilha.

Para realizar a operação que movimenta "brancos" para o topo da pilha, o micro-programa utiliza também um "loop" controlado pelo registrador "SZ" (cujo valor é decrementado de duas unidades a cada iteração) que movimenta uma palavra composta por dois caracteres

"brancos" para o topo da pilha a cada iteração.

Quando o valor do registrador "SZ" atingir zero está concluída o operação. Se o valor de "SZ" for negativo, o micro-programa substitui o último "byte" colocado no topo da pilha por zero.

A operação que converte uma cadeia de palavras em uma cadeia de "bytes" é levada a cabo desprezando os "bytes" mais significativos de todos os valores que compõem a cadeia. Para realizar esta operação o micro-programa decrementa o apontador de pilha de um valor correspondente ao dobro do tamanho da cadeia e passa a realizar um "loop" controlado pelo registrador "SZ" (cujo valor é decrementado a cada iteração). Esse "loop" lê as palavras que compõem a cadeia e escreve os "bytes" menos significativos destas palavras no topo da pilha.

Quando o valor de "SZ" atingir zero, o micro-programa verifica se o apontador de pilha contem um valor ímpar, colocando no topo da pilha, neste caso, um último "byte" (com valor zero).

Para realizar a operação que calcula a posição relativa de um elemento em um arranjo, o micro-programa lê o segundo operando imediato da instrução (que corresponde ao limite inferior do intervalo de índices válidos para o arranjo - vide sub-ítem f do ítem II.2.4) e o subtrai do valor armazenado no topo da pilha (índice).

A seguir, o terceiro operando imediato da instrução (correspondente ao limite superior do intervalo de índices válidos para o arranjo) é desprezado porque o micro-programa não realiza testes para a validação do índice.

Finalmente, o micro-programa realiza a primeira parte de

uma operação de escrita na memória (no topo da pilha) e desvia o fluxo de controle para a seqÜência de micro-instruções que calcula o produto do tamanho dos elementos do arranjo em questão (informado pelo primeiro operando imediato da instrução, que foi armazenado no registrador "SZ") pelo resultado da diferença acima.

d) Interpretação das instruções relacionadas a conjuntos

A interpretação das instruções desta categoria é iniciada por um desvio no fluxo de controle do micro-programa condicionado pelo bit mais significativo do campo OPERAÇÃO.

Caso a operação a ser realizada corresponda a criação de um conjunto vazio no topo da pilha, é atribuído o valor 16 ao registrador "SZ". A seguir, é executado um "loop" controlado por este registrador (cujo valor é decrementado a cada iteração). Cada uma das iterações coloca uma palavra com valor zero no topo da pilha. Quando o valor do registrador "SZ" atingir zero é concluída a realização da instrução.

A realização da instrução que inclui uma seqÜência de elementos em um conjunto armazenado no topo da pilha é iniciada pela leitura do operando imediato "SZ" que informa o tamanho da seqÜência de elementos a serem incluídos no conjunto.

A seguir, o micro-programa realiza um "loop" controlado pelo registrador "SZ" (cujo valor é decrementado a cada iteração). Cada iteração realiza a inclusão de um elemento no conjunto. Quando o "SZ" for zero é concluída a interpretação desta instrução.

A operação que inclui um elemento em um conjunto não fazia parte do conjunto original da máquina básica mas foi incluída para completar a árvore de decodificação.

A realização desta instrução compreende a atribuição do valor zero ao registrador "SZ" seguido de um desvio no fluxo de controle do micro-programa para dentro do "loop" utilizado para implementar a instrução de inclusão de uma seqüência de elementos em um conjunto.

A realização da instrução que verifica se um elemento pertence a um conjunto é iniciada pela retirada do conjunto considerado do topo da pilha seguido da leitura (também do topo da pilha) do valor do elemento cuja relação de pertinência ao conjunto deseja-se testar. A interpretação termina com a atribuição ao topo da pilha do resultado do teste de pertinência.

Para a realização das operações que envolvem dois conjuntos (UNIÃO, INTERSEÇÃO ou DIFERENÇA) o micro-programa coloca o valor 16 no registrador "SZ" e a seguir passa a executar um "loop" controlado por este registrador. Em cada iteração é lida uma palavra de cada um dos dois conjuntos envolvidos e a seguir é feito um desvio condicional no fluxo de controle do micro-programa.

Se a operação que está sendo realizada for a união de dois conjuntos, é realizada a conjunção ("OU" lógico) das palavras destes dois conjuntos.

Se a operação que está sendo realizada for a interseção de dois conjuntos, é realizada a disjunção ("E" lógico) das palavras destes dois conjuntos.

Se a operação que está sendo realizada for a obtenção da diferença dos dois conjuntos, é realizada a disjunção ("E" lógico) da palavra do primeiro conjunto com o complemento lógico ("negação") da palavra do segundo conjunto.

Independentemente do tipo de operação que está sendo realizada, o resultado da operação executada sobre as palavras dos dois conjuntos é salvo sobre a palavra correspondente ao primeiro conjunto e o micro-programa prossegue a execução do "loop".

Quando o valor de "SZ" for nulo, é concluída a execução destas instruções.

e) Interpretação das instruções relacionadas a constantes e desvios

Embora a máquina básica originalmente definida só utilizasse uma instrução de movimentação de constante absoluta para o topo da pilha, o micro-programa implementa quatro modalidades de movimentação.

Na primeira, a constante zero é movimentada para o topo da pilha. Na segunda modalidade, é movimentado para o topo da pilha a constante booleana "true", enquanto que na terceira o valor movimentado para o topo da pilha é o do "byte" menos significativo da instrução.

Finalmente, na quarta modalidade, um operando imediato é lido e movimentado para o topo da pilha.

Se a instrução interpretada for um desvio (ou uma atribuição ao topo da pilha de uma constante relativa ao "PC"), um operando imediato é lido, sendo seu valor subtraído de quatro unidades movimentado para um registrador de trabalho.

No caso da primeira destas instruções, o micro-programa soma o valor do registrador de trabalho com o valor corrente do apontador de programa e movimenta este resultado para o topo da pilha, implementando, assim, a instrução de atribuição de uma constante relativa a

"PC" ao topo da pilha.

No caso da segunda destas instruções, o micro-programa realiza a soma do valor armazenado no registrador de trabalho com o valor corrente do apontador de programa e movimenta este resultado para o apontador de programa, implementando um desvio incondicional.

No caso de desvios condicionais, o micro-programa lê o valor armazenado no topo da pilha e realiza o "OU" exclusivo do bit menos significativo deste valor com o bit menos significativo do campo OPERAÇÃO. Se o resultado desta operação for zero o valor armazenado no registrador de trabalho é adicionado ao valor corrente do apontador de programa, sendo o resultado desta soma movimentado para o apontador de programa.

Este esquema permite implementar dois tipos de desvios condicionais: o primeiro é o previsto na definição original da máquina básica EDISON (desvio realizado caso o valor do topo da pilha corresponda à constante booleana "false"). O segundo não estava previsto na definição original da máquina básica EDISON (desvio realizado caso o valor do topo da pilha corresponda à constante booleana "true").

f) Interpretação das instruções relacionadas à rotinas

A interpretação destas instruções é iniciada com a leitura de um operando imediato seguida de um desvio do fluxo de controle do micro-programa condicionado pelo valor do campo OPERAÇÃO.

Na realização da instrução de alocação de espaço na pilha (para armazenar o valor retornado por uma rotina com tipo), o operando imediato lido é adicionado ao apontador do topo da pilha.

Na interpretação de uma chamada direta de procedimento, o micro-programa adiciona o operando imediato lido com o valor corrente do apontador de programa subtraído de quatro unidades e armazena o resultado em um registrador de trabalho. A seguir o fluxo de controle do micro-programa é desviado de forma a permitir a execução de uma seqüência de micro-instruções que termina as operações de chamada a procedimentos.

Se a instrução for uma chamada indireta de procedimento, o micro-programa lê um segundo operando imediato que indica o nível de aninhamento da rotina que recebeu como parâmetro a rotina que está sendo invocada e, a seguir, lê a entrada da tabela de base correspondente a esse nível de aninhamento.

O valor desta entrada da tabela de base é então adicionado ao primeiro operando imediato da instrução, sendo o resultado utilizado como endereço de uma posição de memória de onde será lido o endereço da rotina que deverá ser invocada.

Este endereço é colocado em um registrador de trabalho, ocorrendo, a seguir, um desvio para uma seqüência de micro-instruções (que é responsável pelo término de todas as instruções de chamada à procedimentos) que salva o valor corrente do apontador de programa no topo da pilha e movimenta o valor armazenado no registrador de trabalho para o apontador de programa.

Existe uma instrução destinada a iniciar a execução de uma rotina que tenha parâmetros ou retorne um valor e que não tenha variáveis locais. Esta instrução não estava prevista no conjunto original de instruções da máquina básica, tendo sido incluída para completar a árvore de decodificação das instruções.

A execução desta instrução é iniciada pela atribuição do valor zero a um registrador de trabalho (utilizado para armazenar o tamanho da região das variáveis locais da rotina). A seguir, ocorre um desvio para uma seqüência de micro-instruções que termina a realização das instruções de inicialização de rotinas.

A interpretação da instrução que permite iniciar a execução de uma rotina qualquer começa com a movimentação do operando imediato já lido para o registrador de trabalho que armazena o tamanho da região das variáveis locais da rotina. A seguir, um segundo operando imediato é lido e o fluxo de controle é desviado para a execução da seqüência de micro-instruções que termina a interpretação das instruções de inicialização de rotinas.

Esta seqüência é iniciada pela movimentação do operando imediato lido para um registrador de trabalho utilizado no armazenamento do nível de aninhamento correspondente à rotina que está sendo iniciada.

A seguir, o valor do registrador virtual "BASE" é lido da posição a ele correspondente na memória principal sendo este valor colocado no topo da pilha. O valor da entrada da tabela de bases correspondente ao nível de aninhamento da rotina é também lido da memória principal e colocado no topo da pilha, juntamente com o valor deste nível de aninhamento.

Um último operando imediato (informa o tamanho da região das variáveis passadas como parâmetro) é lido em seguida e seu valor, adicionado de 18 unidades (tamanho total do contexto das rotinas), é subtraído do valor corrente do apontador de pilha, de forma a fornecer o novo valor da base local.

Este valor é copiado tanto na posição da memória principal que armazena o registrador "BASE", como na entrada da tabela de bases correspondente ao nível de aninhamento da rotina cuja execução está sendo iniciada.

Finalmente, o valor do registrador de trabalho que armazena o tamanho da região das variáveis locais é adicionado ao apontador de pilha de forma a alocar esta região. Este passo conclui a realização das instruções de inicialização de rotinas.

Se a instrução a ser executada corresponde ao término da execução de uma rotina que não utilize parâmetros ou variáveis locais e que não retorne valor (que não estava prevista no conjunto original de instruções da máquina básica), o micro-programa decrementa o apontador de programa de modo a desprezar o operando imediato já lido. A seguir o fluxo de controle é desviado para uma seqüência de micro-instruções que copia o valor armazenado no topo da pilha para o apontador de programa.

Caso a instrução a ser realizada corresponda ao término de uma rotina que utilize parâmetros ou retorne um valor mas não empregue variáveis locais (também não prevista no repertório original da máquina básica), o micro-programa copia o valor do operando imediato lido para um registrador de trabalho (que armazena o tamanho da região das variáveis passadas como parâmetro para a rotina) e, a seguir, o nível de aninhamento da rotina que está sendo terminada é lido do topo da pilha, sendo restaurada a entrada da tabela de bases correspondente a este nível de aninhamento com um valor também lido do topo da pilha.

Um terceiro valor lido do topo da pilha é copiado na posição de memória utilizada para armazenar o valor do registrador "BASE" da máquina virtual, restaurando o valor que este registrador

tinha antes do início da rotina que está sendo concluída.

Finalmente, o apontador de pilha é decrementado de um valor igual ao armazenado no registrador de trabalho que armazena o tamanho da região das variáveis passadas como parâmetro para a rotina e o fluxo de controle é desviado para a seqüência de micro-instruções responsável pela cópia do valor armazenado no topo da pilha para o apontador de programa.

Se a instrução a ser interpretada corresponde ao término de uma rotina que utilize parâmetros e variáveis locais, o apontador de pilha é decrementado de um valor igual ao do primeiro operando imediato lido (que corresponde ao tamanho da região das variáveis locais da rotina que está sendo terminada). A seguir, um segundo operando imediato é lido e o fluxo de controle do micro-programa é desviado para a execução da seqüência de micro-instruções que implementa o término das rotinas que não utilizam variáveis locais.

g) A interpretação das instruções do núcleo

A interpretação destas instruções é iniciada com a movimentação do valor zero para uma posição de memória que armazena o código de um eventual erro encontrado durante a execução destas instruções. A seguir, é realizado um desvio condicional no fluxo de controle do micro-programa para uma das seqüências de micro-instruções que realizam uma das funções do núcleo.

As ações realizadas por cada uma destas seqüências estão discutidas em detalhes no ítem VI.4. que trata especificamente da implementação do núcleo.

h) A interpretação das instruções especiais

A estratégia utilizada na realização das instruções especiais está descrita em detalhes no ítem VI.5. que trata especificamente da interface com os mecanismos de entrada e saída e de comutação de micro-programas.

VI.4. O NÚCLEO

VI.4.1. CONSIDERAÇÕES GERAIS

O núcleo é ativado durante a execução de cinco diferentes instruções da máquina básica.

Na realização destas instruções o núcleo utiliza duas listas circulares de processos (lista dos processos em região crítica e lista dos processos que não estão em região crítica). Os descritores de processos que formam estas listas ocupam 12 palavras. A utilização de cada uma destas 12 palavras está apresentada na figura VI.7.

Para gerenciar as listas de descritores de processos o núcleo utiliza duas posições de memória pré-definidas (que armazenam os endereços das tabelas de bases dos processos mais prioritários de cada uma das listas) e duas micro-rotinas que incluem (em uma posição determinada pela sua prioridade) e retiram descritores de processos de uma destas listas circulares.

Os endereços das posições da memória principal utilizados para armazenar os endereços das tabelas de bases dos processos mais prioritários de cada uma das listas estão apresentados no ítem VI.5.

POSIÇÃO DA PALAVRA	DENOMINAÇÃO DO CAMPO	CONTEÚDO DO CAMPO
0	SEGUINTE	Endereço da tabela de bases do processo seguinte na lista circular
2	ANTERIOR	Endereço da tabela de bases do processo anterior na lista circular
4	BTBASEPAI	Endereço da tabela de bases do processo pai
6	NUMFILHOS	NÚmero de processos filhos
8	PC	Valor do registrador "PC"
10	SP	Valor do registrador "SP" da máquina virtual
12	PRIORIDADE	Prioridade do processo
14	X	Registrador X de interface com E/S
16	E	Registrador E de interface com E/S
18	A	Registrador A de interface com E/S
20	STATUS	"STATUS" do processo
22	BASE	Registrador "BASE" da máquina virtual

Figura VI.7. Campos dos descritores de processos

A seguir estão descritas as estratégias de implementação das micro-rotinas de gerenciamento das listas de descritores de processos e das instruções do núcleo, bem como os esquemas utilizados na realização do escalonamento dos processos e tratamento dos erros eventualmente detetados durante a execução das instruções do núcleo.

VI.4.2. AS MICRO-ROTINAS DE GERENCIAMENTO DE LISTAS

Como a micro-máquina do computador MITRA-15 não dispõe de facilidades para a utilização de micro-rotinas, foi necessário utilizar um artifício na implementação das micro-rotinas que gerenciam as listas de descritores de processos.

A partir da constatação de que cada uma destas micro-rotinas só é invocada em, no máximo, um único ponto da seqüência de micro-instruções responsável pela realização de cada uma das instruções implementadas pelo núcleo (sendo que na instrução responsável pela criação do processo pai nenhuma chamada à micro-rotina é realizada), foi possível utilizar o próprio código da instrução como um indicador do endereço da micro-instrução sucessora das micro-rotinas.

Desta forma, cada uma das micro-rotinas é terminada por um desvio no fluxo de controle do micro-programa condicionado pelo valor dos dois bits mais significativos do campo OPERAÇÃO do código da instrução que estiver sendo executada.

Para realizar a passagem de parâmetros para as micro-rotinas foram utilizados os registradores da micro-máquina.

Assim o registrador usualmente empregado para armazenar o tamanho de cadeias ("SZ") foi empregado para informar o endereço

absoluto da posição da memória principal onde fica armazenado o endereço da tabela de bases do processo mais prioritário da lista a ser manipulada pela micro-rotina a ser invocada.

Deve-se observar que os descritores de processos estão armazenados na memória principal nas posições imediatamente anteriores às ocupadas pelas tabelas de bases dos processos correspondentes. Desta forma é possível obter o endereço do descritor de processo a partir do endereço da tabela de bases correspondente a este processo.

O segundo parâmetro passado para as micro-rotinas informa o endereço da tabela de bases do processo cujo descritor deve ser inserido ou retirado de uma das listas circulares.

O valor deste parâmetro é passado através do registrador "BTBASE" da máquina virtual. Este artifício facilita a invocação das micro-rotinas para colocar ou retirar o descritor de processo corrente em uma das listas circulares (porque o endereço da tabela de bases do processo corrente fica armazenado sempre no registrador "BTBASE" da máquina virtual).

A seguir são descritos os passos executados por cada uma das micro-rotinas utilizadas.

a) A micro-rotina "PDE FILA"

Esta micro-rotina é responsável pela colocação de um descritor de processo em uma das listas circulares.

Como os registradores da micro-máquina destinados a armazenar os apontadores de pilha e de programa são utilizados por esta rotina como registradores de trabalho, os valores destes registradores

são resguardados na memória principal para posterior restauração ao término da execução desta micro-rotina.

Com este artifício, apesar do conteúdo de um dos registradores primariamente reservado para rascunho não ser afetado pela micro-rotina (podendo portanto ser utilizado pelo micro-programa principal), foi possível empregar três registradores de trabalho na implementação desta micro-rotina.

Desta forma o endereço da tabela de bases do processo mais prioritário da lista (apontado por "SZ") pode ser copiado em um destes registradores de trabalho ("SP"). Se o valor deste endereço for -1, considera-se que a lista em questão está vazia.

Neste caso, o endereço da tabela de bases do processo a ser inserido na lista é copiado para a posição de memória reservada para o armazenamento do endereço da tabela de bases do processo de maior prioridade da lista. A seguir, este endereço é copiado também para os campos ANTERIOR e POSTERIOR do descritor do processo que está sendo incluído, permitindo desta forma a inicialização da lista.

Finalmente, o fluxo de controle do micro-programa é desviado para a seqüência de micro-instruções responsável pelo término desta micro-rotina.

Caso a lista não esteja vazia, o valor do campo PRIORIDADE do descritor do processo a ser incluído é copiado para um dos registradores de trabalho ("PC") e, a seguir, comparado com o valor do campo PRIORIDADE de cada um dos descritores da lista circular.

A varredura da lista é interrompida quando o valor do campo PRIORIDADE do descritor do processo varrido for menor que a prio-

ridade do processo cujo descritor deve ser incluído na lista circular ou quando a lista circular tiver sido completamente consultada.

A micro-rotina verifica, a seguir, se o último processo varrido foi o mais prioritário da lista. Neste caso, o endereço da tabela de bases do processo a ser inserido é copiado para a posição da memória principal destinada a armazenar o endereço do processo mais prioritário desta lista circular.

A seguir, a micro-rotina encadeia o descritor do processo em questão na lista circular na posição imediatamente anterior à do último processo varrido e realiza um desvio no fluxo de controle do micro-programa para a seqüência de micro-instruções responsável pela terminação da micro-rotina.

Esta seqüência restaura os valores dos registradores "SP" e "PC" e realiza um desvio no fluxo de controle do micro-programa condicionado pelos dois bits mais significativos do campo OPERAÇÃO do código da instrução corrente, de forma a implementar o retorno da micro-rotina.

b) A micro-rotina "TIRA FILA"

Esta micro-rotina retira um descritor de processo de uma das listas circulares. Sua execução é iniciada pela leitura do endereço da tabela de bases do processo de maior prioridade da lista. Este endereço é comparado com o endereço da tabela de bases do processo a ser retirado da lista e, se eles forem iguais, o campo SEGUINTE do descritor do processo que está sendo retirado da lista é lido. Se o valor lido deste campo também for igual ao endereço da tabela de bases do processo que está sendo retirado da lista, a micro-rotina conclui que está sendo retirado o único processo da lista, ficando esta, portanto vazia.

Neste caso o valor -1 é copiado na posição de memória que armazena o endereço da tabela de bases do processo de maior prioridade da lista e o fluxo de controle é desviado para a micro-instrução responsável pelo término desta micro-rotina.

Se o processo a ser retirado da lista for o mais prioritário mas não for o único da lista, a micro-rotina copia o valor do campo SEGUINTE do descritor deste processo na posição de memória que armazena o endereço da tabela de bases do processo mais prioritário da lista considerada (posição apontada por "SZ") e realiza um desvio no fluxo de controle para a seqüência de micro-instruções responsável pelo desencadeamento de um descritor de processo da lista circular.

Se o processo a ser retirado da lista circular não for o mais prioritário desta lista, é realizado imediatamente um desvio no fluxo de controle para a seqüência de micro-instruções responsável pelo desencadeamento de um descritor de processos da lista circular.

A seqüência de desencadeamento atualiza os campos ANTERIOR e SEGUINTE dos descritores de processo adjacentes ao descritor do processo que está sendo excluído da lista circular e desvia o fluxo de controle para a micro-instrução que implementa o retorno desta micro-rotina.

VI.4.3. AS INSTRUÇÕES EXECUTADAS PELO NÚCLEO

Como já foi visto, a execução de uma instrução do núcleo é sempre iniciada pela movimentação do valor zero para uma posição da memória principal destinada a armazenar o código de um eventual erro encontrado durante sua execução. A seguir é realizada a decodificação das cinco possíveis instruções desta categoria.

A seguir estão discutidos os passos realizados na execução de cada uma destas instruções.

a) A instrução de criação do processo pai

A execução desta instrução é iniciada pela leitura do operando imediato que informa o tamanho do código do programa cuja execução está sendo iniciada.

O valor deste operando é adicionado ao valor corrente do apontador de programa e o resultado é copiado no apontador do topo da pilha. A seguir são reservadas 12 palavras no topo da pilha para o armazenamento do descritor do processo pai (inicializadas com zeros).

A posição do descritor do processo que armazena o endereço da tabela de bases do processo pai recebe o valor -1, indicando que este processo não tem processo pai.

Finalmente, a posição da memória principal que armazena o tamanho das tabelas de bases dos processos (vide ítem VI.5.) é lida e seu conteúdo é adicionado ao apontador de pilha de modo a reservar o espaço necessário para a tabela de bases do processo pai. Neste ponto a execução desta instrução é encerrada.

b) A instrução de criação de um processo filho

A interpretação desta instrução começa com a leitura da palavra "STATUS" do descritor do processo que está sendo executado.

Se o "STATUS" não for zero, o núcleo conclui que o processo está em região crítica e desvia para a execução da seqüência de

micro-instruções responsável pelo tratamento de erros (vide sub-ítem a do ítem VI.4.4).

Caso o valor de "STATUS" seja zero, o micro-programa lê o campo BTBASEPAI do descritor do processo que está sendo executado.

Se este valor deste campo não for -1, o micro-programa conclui que o processo que está sendo executado é um processo filho e realiza um desvio em seu fluxo de controle para a seqüência de micro-instruções responsável pelo tratamento de erros.

Caso não tenha detetado nenhum erro, o micro-programa salva o endereço da tabela de bases do processo corrente em uma posição da memória principal (vide ítem VI.5.) e em seguida, o valor do seu apontador de pilha é salvo no descritor do processo pai. Finalmente o núcleo passa a executar um "loop" responsável pela criação de processos.

Cada iteração deste "loop" é iniciada pela incrementação do campo NUMFILHOS do descritor do processo pai, seguido da montagem do contexto (descritor de processo e tabela de bases) e da pilha do processo filho. O descritor do processo criado é então colocado na lista dos descritores dos processos que não estão em região crítica e a iteração do "loop" é terminada pela leitura da próxima instrução a ser executada pelo processo pai.

A montagem do contexto do novo processo é iniciada pela alocação, no topo da pilha do processo pai, de um espaço para o descritor do novo processo. Isto é feito adicionando o valor 24 (tamanho do descritor do processo) ao apontador de pilha e, a seguir, movimentando o novo valor do apontador de pilha para o registrador que armazena o endereço da tabela de bases.

A seguir os campos PRIORIDADE, X, E, A, STATUS e BASE do descritor do processo pai são copiados para o descritor do novo processo.

Para inicializar o campo SP do descritor é realizada uma leitura na posição de memória que armazena o tamanho das tabelas de bases dos processos, sendo o valor lido adicionado ao valor corrente do apontador de pilha. O resultado desta adição é, então, copiado no campo SP do descritor do processo filho.

A inicialização do campo PC do descritor é feita copiando o valor corrente do apontador de programa adicionado de quatro unidades para este campo (as quatro unidades adicionadas são necessárias porque a instrução de criação de processo filho tem dois operandos imediatos que ainda serão lidos).

O campo NUMFILHOS do descritor do processo filho é inicializado com zero, enquanto que o campo BTOASEPAI é inicializado com o endereço da tabela de bases do processo pai.

No final da construção do descritor do novo processo, a tabela de bases do processo pai é copiada para o topo da pilha para inicializar a tabela de bases do processo filho.

A construção do contexto do processo filho é terminada com a leitura dos dois operandos imediatos da instrução de criação de processos.

O primeiro destes operandos é subtraído de quatro unidades e adicionado ao apontador de programa de forma a permitir que este apontador passe a armazenar o endereço da próxima instrução do processo pai a ser executada. O segundo operando imediato é multiplicado por 256

e adicionado ao apontador de pilha de forma a reservar espaço para a pilha do processo que está sendo criado.

A seguir, o endereço da posição de memória utilizada para armazenar o endereço da tabela de bases do processo de maior prioridade da lista dos processos que não estão em região crítica é copiado no registrador "SZ" e a micro-rotina "POE FILA" é invocada de forma a incluir o novo processo nesta lista.

Após a conclusão da execução desta micro-rotina, a próxima instrução do processo pai é lida e seu código é comparado com o código da instrução de criação de processo filho. Se estes valores forem iguais, mais uma iteração do "loop" de criação de processos será realizada.

Caso contrário, o valor corrente do apontador de programa subtraído de duas unidades é copiado no campo PC do descritor do processo pai e o fluxo de controle é desviado para a seqüência de micro-instruções que seleciona o próximo processo a assumir o controle do processador (vide sub-ítem b do ítem VI.4.4.).

c) A instrução de término de processo

A interpretação desta instrução é iniciada pela leitura do valor da palavra "STATUS" do descritor do processo que está sendo executado.

Se este valor não for zero o núcleo conclui que o processo corrente está em região crítica e desvia o fluxo de controle para a seqüência de micro-instruções responsável pelo tratamento de erros (vide sub-ítem a do ítem VI.4.4.).

Caso contrário o campo BTBASEPAI do descritor do processo corrente é lido. Se BTBASEPAI for -1, o núcleo conclui que o processo que está sendo terminado é o processo pai e realiza um desvio no fluxo de controle para a seqüência de micro-instruções que comuta o controle da micro-máquina para o micro-programa original (vide item VI.5.).

Caso contrário, o núcleo decrementa o campo NUMFILHOS do descritor do processo pai. Se o resultado for zero, o núcleo conclui que o processo que está sendo terminado é o último processo filho. Neste caso o endereço da tabela de bases do processo pai é copiado no registrador "BTBASE" e a micro-rotina "POE FILA" é invocada para incluir o descritor do processo pai na lista dos processos que não estão em região crítica.

Após a conclusão desta micro-rotina, é realizado um desvio para a seqüência de micro-instruções que seleciona o próximo processo a conquistar o controle do processador (vide sub-ítem b do ítem VI.4.4.).

Caso o valor do campo NUMFILHOS do descritor do processo pai não tenha atingido o valor zero, ocorre imediatamente o desvio para a seqüência de micro-instruções mencionada no parágrafo anterior.

d) Instrução de indicação de início de região crítica

A execução desta instrução é iniciada pela leitura da palavra "STATUS" do descritor do processo que está sendo executado.

Se o "STATUS" não for zero, o núcleo conclui que o processo corrente já está em região crítica e desvia o fluxo de controle para a seqüência de micro-instruções responsável pelo tratamento de erros (vide sub-ítem a do ítem VI.4.4.).

Caso contrário o valor um é copiado no campo STATUS do descritor do processo que está sendo executado e os valores correntes dos registradores responsáveis pelo armazenamento dos apontadores de pilha e de programa ("SP" e "PC") são copiados nos campos correspondentes do descritor do processo corrente.

A seguir o descritor do processo corrente é colocado na lista de processos em região crítica pela micro-rotina "POE FILA".

Ao final da execução desta micro-rotina ocorre um desvio no fluxo de controle para a seqüência de micro-instruções responsável pela seleção do próximo processo a conquistar o controle do processador (vide sub-ítem b do ítem VI.4.4.).

e) Instrução de indicação de término de região crítica

A execução desta instrução é iniciada pela leitura da palavra "STATUS" do descritor do processo que está sendo executado.

Se o "STATUS" for zero, o núcleo conclui que o processo corrente não está em região crítica e desvia o fluxo de controle para a seqüência de micro-instruções responsável pelo tratamento de erros (vide sub-ítem a do ítem VI.4.4.).

Caso contrário o núcleo copia o valor do campo SEGUINTE do descritor do processo corrente para um registrador de trabalho para obter o endereço da tabela de bases do próximo processo da lista dos processos em região crítica.

A seguir este endereço é comparado com o conteúdo da posição de memória que armazena o endereço da tabela de bases do pro-

cesso mais prioritário da lista de processos em região crítica. Se estes valores forem iguais, o núcleo conclui que o processo corrente é o menos prioritário da lista de processos em região crítica. Neste caso o valor -1 é copiado no registrador de trabalho para o qual tinha sido copiado o conteúdo do campo SEGUINTE do descritor do processo corrente.

A seguir o núcleo lê o valor armazenado no topo da pilha do processo corrente e verifica se este valor corresponde ao da constante booleana "true".

Neste caso, a condição utilizada para guardar a região crítica condicional cuja execução está sendo terminada não foi atendida e o fluxo de controle é desviado para a seqüência de micro-instruções que termina a interpretação da instrução que indica o final de uma região crítica.

Caso contrário, o valor zero é movido para o campo "STATUS" do descritor do processo e o valor corrente do apontador de programa é salvo no campo PC deste descritor. A seguir a micro-rotina "TIRA FILA" é invocada para retirar este descritor da lista dos processos em região crítica.

Após esta retirada, o descritor de processos é incluído na lista dos processos que não estão em região crítica (pela micro-rotina "POE FILA"). Em seguida, é realizado um desvio no fluxo de controle para a seqüência de micro-instruções que termina a interpretação das instruções de indicação de final de região crítica.

Esta seqüência é iniciada por um desvio condicionado pelo valor do registrador de trabalho que recebeu o endereço do descritor do próximo processo da lista dos processos em região crítica.

Se este valor for -1, é porque o processo que estava com o controle do processador era o menos prioritário da lista dos processos em região crítica. Neste caso, o fluxo de controle do micro-programa é desviado para a seqÜência de micro-instruções que selecionará para execução um dos processos da lista dos processos que não estão em região crítica (vide sub-ítem c do ítem VI.4.4.).

Caso contrário, o valor do registrador de trabalho considerado é copiado no registrador que armazena o endereço da tabela de bases do processo corrente ("BTBASE") e o valor um é atribuído ao "byte" mais significativo do registrador que armazena o código da instrução que está sendo executada.

A seguir o fluxo de controle é desviado para a seqÜência de micro-instruções que carrega o contexto de um processo (vide sub-ítem d do ítem VI.4.4.).

VI.4.4. O ESCALONAMENTO DE PROCESSOS

A parte do micro-programa destinada a realizar o escalonamento de processos utiliza quatro seqÜências de micro-instruções que terminam a interpretação das instruções do núcleo.

A primeira seqÜência destina-se a tratar o término anormal de uma instrução do núcleo, sendo utilizada sempre que for detetado um erro.

A segunda é utilizada na finalização das instruções de criação de processo filho, de término de processo e de indicação de início da região crítica. Esta seqÜência seleciona o próximo processo que deve assumir o controle do processador ao final de uma destas instruções.

As duas últimas seqüências são utilizadas para terminar a execução das instruções de indicação de final de região crítica. Estas seqüências são necessárias em virtude do tipo de política adotada na seleção do próximo processo a assumir o controle do processador (um processo que não esteja em região crítica só deve ser selecionado para execução depois que todos os processos que estejam em região crítica tenham assumido uma vez o controle do processador).

A seguir são discutidas em detalhes cada uma destas seqüências de micro-instruções.

a) O término anormal de instruções do núcleo

A execução desta seqüência de micro-instruções é iniciada pelo armazenamento do "byte" mais significativo do registrador "IR" no "byte" mais significativo da posição da memória principal que armazena o valor do código do erro encontrado. No "byte" menos significativo desta posição de memória é armazenado o "byte" menos significativo do campo STATUS do descritor do processo corrente.

A seguir os valores de "SP", "PC" e "BTBASE" são também armazenados em posições pré-definidas da memória principal (vide ítem VI.5.4. para discussão dos endereços destas posições) e o fluxo de controle do micro-programa sofre um desvio condicionado pelo valor do campo STATUS do descritor do processo que está sendo executado.

Assim, se o processo que utilizou a instrução na qual foi detetado o erro estava em região crítica, o código da instrução de término de processo é copiado para o registrador "IR" e a micro-rotina "TIRA FILA" é invocada de forma a retirar o descritor do processo em

questão da lista de processos em região crítica.

Após a retirada, o fluxo de controle é desviado para a seqÜência de micro-instruções responsável pelo término normal de processos a partir da sua segunda micro-instrução.

Como a primeira micro-instrução desta seqÜência é a responsável pela verificação do STATUS do processo que está sendo abortado, este esquema garante que o processo que utilizou a instrução na qual foi detetado o erro será terminado, independentemente do valor do campo STATUS do seu descritor de processo.

Caso o processo que executou a instrução causadora do erro não esteja em região crítica, o desvio do fluxo de controle do micro-programa para a segunda instrução da seqÜência de micro-instruções responsável pela término normal de processos é feito imediatamente.

b) A seleção do processo a ser ativado

A execução desta seqÜência de micro-instruções é iniciada pela leitura da posição da memória principal que armazena o endereço da tabela de bases do processo mais prioritário da lista de processos em região crítica para o registrador "BTBASE".

Se o valor lido for -1, esta lista é considerada vazia. Neste caso o fluxo de controle é desviado para a seqÜência de micro-instruções responsável pelo escalonamento de um processo que não está em região crítica (vide sub-item c).

Caso a lista de processos em região crítica não esteja vazia, o processo a ser selecionado para execução é o mais prioritário desta lista. Neste caso o valor um é colocado no registrador "IR" e o

controle é transferido para a seqÜência de micro-instruções responsável pelo carregamento do contexto de um processo (vide sub-ítem d).

c) O escalonamento de um processo que
não esteja em região crítica

Esta seqÜência de micro-instruções é iniciada com a leitura da posição da memória principal que armazena o endereço da tabela de bases do processo mais prioritário da lista dos processos que não estão em região crítica para o registrador "BTBASE".

Se o valor lido não for -1, o micro-programa conclui que esta lista não está vazia e que o processo a ser selecionado é o mais prioritário desta lista (i.e., o processo a ser selecionado é aquele cujo endereço da tabela de bases foi lido para o registrador "BTBASE").

Como foi adotado o critério de que sempre que um processo que não esteja em região crítica estiver com o controle do processador, o descritor de processo a ele correspondente não deve estar em nenhuma das listas circulares de processos, é necessário retirar o descritor do processo que vai assumir o controle do processador da lista dos processos que não estão em região crítica.

Assim o valor zero é atribuído ao registrador "IR" e a micro-rotina "TIRA FILA" é invocada para retirar o descritor do processo em questão desta lista.

A atribuição do valor zero ao registrador "IR" é realizada para permitir que a micro-rotina "TIRA FILA" atinja corretamente o endereço de retorno.

Como já foi assinalado, o valor deste registrador é

utilizado como indicação deste endereço de retorno. Em conseqüência da não utilização de qualquer chamada à micro-rotina "TIRA FILA" durante a execução da instrução de criação de processo filho, foi possível empregar o valor dos dois bits mais significativos do campo OPERAÇÃO desta instrução (zeros) para assinalar que esta micro-rotina está sendo invocada durante a seleção do próximo processo a assumir o controle do processador.

Após o término desta micro-rotina, o fluxo de controle do micro-programa é desviado para a seqüência de micro-instruções responsável pelo carregamento do contexto de um processo (vide sub-ítem d).

Se a lista dos processos que não estão em região crítica estiver vazia, o micro-programa realiza uma leitura da posição de memória que armazena o endereço da tabela de bases do processo mais prioritário da lista de processos em região crítica para o registrador "BTBASE".

Se o valor lido não for -1, a lista não está vazia e o processo a ser selecionado, neste caso, é o mais prioritário desta lista (i.e., o processo a ser selecionado é aquele cujo endereço da tabela de bases foi carregado no registrador "BTBASE").

Desta forma, o micro-programa move o valor um para "IR" e realiza um desvio para a seqüência de micro-instruções responsável pelo carregamento do contexto de um processo (vide sub-ítem d).

Se a lista dos processos que não estão em região crítica estiver vazia, existe uma inconsistência nas listas de processos (uma vez que não foi detetado o final do processo pai e não existe mais nenhum processo no sistema). Neste caso é forçado um término anormal da execução do programa.

Para isto, o valor dois é movido para o "byte" menos significativo da posição de memória que armazena o código do erro encontrado e o controle é transferido para a seqÜência de micro-instruções que comuta o controle da micro-máquina para o micro-programa original do computador MITRA-15 (vide item VI.5.).

d) O carregamento do contexto de um processo

A seqÜência de micro-instruções responsável pelo carregamento do contexto de um processo é iniciada pela cópia dos valores armazenados nos campos "SP" e "PC" do descritor do processo cujo endereço da tabela de bases está armazenado no registrador "BTBASE" para os registradores correspondentes.

A seguir o valor de "IR" é testado. Se IR não for zero (indicando que o processo cujo contexto está sendo carregado está em região crítica), o valor correspondente à constante booleana "false" é colocada no topo da pilha.

Finalmente, é lida a posição da memória principal que armazena o código de um eventual erro detetado durante a execução da instrução que está sendo terminada.

Se este código for zero, é concluída a execução da instrução. Caso contrário, o controle é desviado para a seqÜência de micro-instruções que faz a comutação do controle da micro-máquina para o micro-programa original do computador MITRA-15 (vide item VI.5.).

VI.5. A COMUTAÇÃO DE MICRO-PROGRAMAS E E/S

VI.5.1. CONSIDERAÇÕES GERAIS

Como já foi mencionado, procurou-se aproveitar os mecanismos de E/S já disponíveis no computador MITRA-15.

Para isto foi necessário conceber um esquema que permite a comutação dinâmica do controle da micro-máquina entre dois micro-programas diferentes.

O primeiro micro-programa envolvido nesta comutação é o original do computador MITRA-15 (que fica residente na parte da micro-memória original do MITRA-15 que é do tipo "PROM").

O segundo micro-programa é o que implementa a máquina básica EDISON (e está discutido nos ítens anteriores deste capítulo). Este micro-programa é carregado na nova memória de controle construída para o computador MITRA-15 (do tipo "RAM").

Para facilitar a compreensão do esquema empregado na comutação do controle da micro-máquina entre estes dois micro-programas é apresentada, a seguir, uma breve discussão do nível convencional implementado pelo micro-programa original do compuador MITRA-15.

É também apresentado o esquema empregado na interface dos programas codificados utilizando o repertório original do computador MITRA-15 com os mecanismos de entrada e saída utilizados neste computador, bem como o esquema utilizado para realizar a interface dos programas EDISON com estes mecanismos de entrada e saída.

A comutação do controle da micro-máquina para o micro-programa original possibilita a utilização de um programa codificado no

nível convencional original do computador MITRA-15 para realizar as operações de entrada e saída (e tratamento de exceções) requeridas (programa "MICMON" - vide capítulo V).

A parte deste programa ("MICMON") diretamente envolvida na realização destas operações é também apresentada.

VI.5.2. O NIVEL CONVENCIONAL ORIGINAL DO COMPUTADOR MITRA-15

Os mesmos oito registradores empregados na realização da máquina básica EDISON são também utilizados na realização do nível convencional original.

Na figura VI.8. estão apresentadas as denominações destes registradores no conjunto de instruções original do computador MITRA-15. As funções reservadas para cada um destes registradores são:

- Registrador "P" - é utilizado como apontador de programa (armazena o endereço da próxima instrução do nível convencional a ser executada);
- Registrador "L" - armazena o endereço de uma região de dados locais (base local);
- Registrador "G" - armazena o endereço da primeira posição da região de memória controlada pelo programa do nível convencional (base global);
- Registradores "A" e "E" - registradores de uso geral;
- Registrador "X" - armazena o índice empregado nos acessos à variáveis indexadas (arranjos) utilizados por

algumas instruções implementadas pelo micro-programa original;

- Registradores "V" e "W" - registradores de trabalho do micro-programa (não são acessíveis ao nível convencional).

DENOMINAÇÃO DO REGISTRADOR NA MAQUINA BASICA EDISON	DENOMINAÇÃO DO REGISTRADOR NA MAQUINA VIRTUAL ORIGINAL DO COMPUTADOR MITRA-15
"PC"	"P"
"BTBASE"	"L"
"MONITORBASE"	"G"
"ACC"	"A"
"ACC1"	"E"
"SP"	"X"
"IR"	"V"
"SZ"	"W"

Figura VI.8. CorrespondÊncia entre os registradores da máquina básica EDISON e os registradores da máquina virtual original do computador MITRA-15

Os programas do nível convencional original do computador MITRA-15 são organizados em segmentos que ficam armazenados em regiões contíguas da memória principal e podem ser de três tipos:

- Segmento de dados globais ("CDS");
- Segmento de dados locais ("LDS");
- Segmento de código ("LPS").

Cada programa do nível convencional utiliza um segmento de dados globais (que é o primeiro segmento do programa) e um ou mais segmentos de dados locais.

A seguir a cada segmento de dados locais são colocados um ou mais segmentos de código.

Um segmento de código é composto pelas instruções utilizadas para implementar uma rotina. Estas instruções podem acessar diretamente tanto os dados armazenados no segmento de dados locais que precede o segmento de código em questão como os dados do segmento de dados globais.

Para permitir estes acessos o registrador "C" armazena o endereço da posição de memória a partir da qual está implantado o segmento de dados globais do programa enquanto que o registrador "L" aponta para o segmento de dados locais que precede o segmento de código correspondente à rotina que está sendo executada.

VI.5.3. INTERFACE DOS PROGRAMAS DO NIVEL CONVENCIONAL ORIGINAL COM OS MECANISMOS DE ENTRADA E SAIDA

As 36 primeiras palavras (72 "bytes") do segmento de dados globais dos programas do nível convencional são reservadas como

área de trabalho e comunicação com os módulos do sistema operacional do computador MITRA-15.

Estes módulos, incluindo os mecanismos de E/S, fazem parte de um programa supervisor (codificado utilizando as instruções do nível convencional original) que se comunica com as partes do micro-programa original que controlam diretamente as ações dos periféricos e linhas de comunicação utilizados no computador MITRA-15.

Este programa é invocado por um programa qualquer do nível convencional original através de uma instrução especial do conjunto original do MITRA-15 denominada "CSV" ("Call SuperVisor"). Esta instrução realiza um desvio no fluxo de controle do programa para um dos módulos do programa supervisor.

A passagem de parâmetros para as rotinas do supervisor é feita utilizando os registradores "A", "E" e "X", sendo que eventuais valores retornados por estas rotinas são também colocados nestes registradores.

As descrições das funções realizadas por cada um dos módulos do supervisor podem ser encontradas no manual do "Moniteur Temps réel disque - MTRD" (7).

VI.5.4. INTERFACE DOS PROGRAMAS EDISON COM OS MECANISMOS DE ENTRADA E SAIDA

Na definição original da linguagem EDISON foram previstas quatro "rotinas embutidas" ("ADDR", "PLACE", "OBTAIN" e "SENSE") destinadas a fornecer facilidades para a realização das operações de entrada e saída.

Como já foi visto, a "rotina embutida ADDR" não afeta diretamente as operações de entrada e saída, sendo utilizadas apenas na obtenção do endereço absoluto de uma variável que será utilizada em uma operação de entrada ou saída. Desta forma esta "rotina embutida" pode ser implementada através do uso de uma das instruções de obtenção de endereço de variável previstas no conjunto de instruções da máquina básica EDISON.

As outras três "rotinas embutidas" são implementadas pelas instruções especiais "PLACE", "OBTAIN" e "SENSE" e operam sobre dados que ocupam uma palavra (dois "bytes"), utilizando endereços que ocupam duas palavras (quatro "bytes").

Como já foi visto, o computador MITRA-15 utiliza endereços compostos por apenas 16 bits (dois "bytes"). Desta forma a palavra mais significativa dos endereços empregados pode ser utilizada como um indicador do tipo de operação a ser efetivamente realizada.

Assim, na instrução especial "PLACE", um valor negativo na palavra mais significativa do endereço da posição afetada é utilizado para indicar que a palavra de dados empregada pela instrução deve ser copiada em uma das palavras do descritor do processo que a empregar.

A tabela da figura VI.9. apresenta o campo do descritor de processo afetado pela execução desta instrução neste caso, segundo os valores presentes na palavra menos significativa do endereço utilizado.

Caso o valor da palavra mais significativa do endereço utilizado em uma instrução "PLACE" seja zero, o conteúdo da palavra de dados empregada na instrução será copiado para a posição da memória principal do computador MITRA-15 cujo endereço é informado pelo conteúdo da palavra menos significativa do endereço fornecido para a instrução.

VALOR DA PALAVRA MENOS SIGNIFICATIVA DO ENDEREÇO	POSIÇAO DO DESCRITOR DE PROCESSO AFETADA
0	A
1	A
2	E
3	PRIORIDADE
4	SP
5	PC
6	NUMFILHOS
7	BTBASEPAI
8	ANTERIOR
9	SEGUINTE

Figura VI.9. Campo do descritor de processo afetado segundo os valores da palavra menos significativa do endereço utilizado em instruções especiais

Se a palavra mais significativa do endereço for maior que zero, o valor da palavra de dados empregada por esta instrução é copiado

em uma posição da memória principal e o código da instrução "PLACE" é colocado na posição da memória principal que armazena o código de um erro encontrado durante a execução de uma instrução.

A seguir o controle da micro-máquina é devolvido ao micro-programa original de modo a permitir que uma rotina do programa "MICMON" passe a ser executada (vide ítens V.3. e VI.5.5.).

Este programa lê código do erro encontrado e verifica que a última instrução realizada pelo novo micro-programa foi uma instrução especial.

Neste caso, este programa conclui que não foi detetado erro algum e carrega os registradores "A", "E" e "X" com os valores lidos dos campos correspondentes do descritor do processo que utilizou a instrução especial e a seguir invoca a rotina do supervisor cujo número é indicado pelo conteúdo da posição de memória na qual foi armazenada a palavra de dado utilizada pela instrução "PLACE".

Após o término da execução desta rotina do supervisor, o programa "MICMON" copia os valores dos registradores "A", "E" e "X" para os campos correspondentes do descritor do processo que utilizou a instrução "PLACE" e devolve o controle da micro-máquina ao micro-programa responsável pela implementação da máquina básica EDISON.

Este esquema permite a realização de chamadas às rotinas do supervisor por programas EDISON de uma forma cômoda.

Para realizar uma destas chamadas é suficiente o uso de instruções "PLACE" com a palavra mais significativa do endereço com valores negativos, de forma a atribuir valores aos campos correspondentes aos registradores da máquina virtual original do descritor de pro-

cesso ("A", "E" e "X") e a seguir utilizar novamente a instrução "PLACE", desta vez com um valor positivo na palavra mais significativa do endereço e com um valor da palavra de dados correspondendo ao número da rotina do supervisor cuja execução deve ser invocada.

Nas instruções "OBTAIN" e "SENSE", sempre que a palavra mais significativa do endereço utilizado for negativa, será realizada uma referência ao valor do registrador da micro-máquina cujo número é informado pelo valor da palavra menos significativa do endereço utilizado.

Nestas instruções, caso o valor da palavra mais significativa do endereço empregado seja zero, será referenciada a posição da memória principal apontada pelo conteúdo da palavra menos significativa do endereço utilizado.

Finalmente, caso o valor da palavra mais significativa do endereço utilizado em uma instrução "OBTAIN" ou "SENSE" seja positivo, uma referência ao campo do descritor do processo especificado pela palavra menos significativa do endereço empregado (vide figura VI.9.) será realizada.

VI.5.5. AS SEQUÊNCIAS DE MICRO-INSTRUÇÕES RELACIONADAS COM A INTERFACE COM O PROGRAMA "MICMON"

Como já foi visto, as operações de E/S são sempre realizadas através de uma comutação do controle da micro-máquina para o micro-programa original, de forma a permitir a execução da parte do programa "MICMON" que é responsável pela chamada às rotinas do programa supervisor do computador MITRA-15.

Esta parte do programa "MICMON" é responsável também pelo envio de mensagens de erro ao usuário e pelo término da execução de um programa EDISON.

Finalmente, existem ainda duas instruções implementadas pelo micro-programa responsável pela realização da máquina básica EDISON que se relacionam diretamente com este programa.

A primeira destas instruções ("INI") carrega os valores dos registradores da máquina básica a partir de posições pré-definidas da memória principal e destina-se a facilitar a comutação do controle da micro-máquina do micro-programa original para o novo micro-programa.

A segunda destas instruções ("TRAP") realiza a operação inversa, permitindo a implementação dos "breakpoints" previstos para facilitar a depuração dos programas EDISON.

Para permitir a comunicação entre o micro-programa que realiza a máquina básica EDISON e o programa "MICMON" foi utilizada uma região do segmento de dados globais deste programa.

Esta região ocupa 14 palavras (28 "bytes") e começa na trigésima-sétima palavra do segmento de dados globais do programa "MICMON" (as 36 primeiras palavras deste segmento são utilizadas como área de trabalho do programa supervisor do computador MITRA-15). A tabela apresentada na figura VI.10. indica a utilização de cada uma destas 14 palavras.

Como o endereço do segmento de dados globais do programa "MICMON" é mantido no registrador "MONITORBASE" da máquina básica, o acesso do micro-programa a esta região de comunicação é bastante simples (o endereço de uma das palavras que a compõem pode ser obtido através

POSIÇÃO DO CAMPO NO SEGMENTO DE DADOS GLOBAIS DO PROGRAMA "MICMON"	DENOMINAÇÃO DO CAMPO	VALOR ARMAZENADO
36	"BTSIZE"	tamanho da tabela de bases
37	"FILAPRONTOS"	endereço da tabela de bases do processo mais prioritário da lista dos processos que não estão em região crítica
38	"FILACONDIÇÃO"	endereço da tabela de bases do processo mais prioritario da lista dos processos que estão em região crítica
39	"BT"	endereço da tabela de bases do processo corrente
40	"SP"	valor corrente do reg. "SP"
41	"PC"	valor corrente do reg. "PC"
42	"L"	valor do registrador "L" da máquina virtual original
43	"P"	valor do registrador "P" da máquina virtual original
44	"STDSWMASC"	valor a ser colocado no reg. de controle para comutar para o micro-programa original
45	"STDSWADDR"	endereço do reg. de controle
46	"ERRIR"	código do erro detetado na execução de uma instrução
47	"ERRSP"	valor do registrador "SP" no momento do erro
48	"ERRPC"	valor do registrador "PC" no momento do erro
49	"ERRBTBASE"	valor do registrador "BTBASE" no momento do erro

Figura VI.10. Campos da área de comunicação entre o novo micro-programa e o programa "MICMON"

de uma adição da posição relativa da palavra no segmento de dados globais do programa "MICMON" com o valor armazenado no registrador "MONITORBASE").

Como o registrador "MONITORBASE" foi mapeado no mesmo registrador da micro-máquina utilizado para armazenar o valor da base global utilizada pelo micro-programa original, não é necessário realizar qualquer modificação no conteúdo deste registrador quando for feita a comutação do controle da micro-máquina do micro-programa original para o novo micro-programa (e vice-versa).

A seguir estão descritas as ações realizadas por cada uma das seqüências de micro-instruções relacionadas com a interface com o programa "MICMON".

a) Seqüência de micro-instruções responsável
pela implementação da instrução "PLACE"

Esta seqüência é iniciada pela leitura de três valores do topo da pilha para registradores. O primeiro corresponde ao valor a ser copiado para uma posição da memória principal (ou para um dos campos do descritor do processo corrente, ou ainda o número da rotina do programa supervisor a ser invocada).

O segundo valor lido especifica o tipo de acesso a ser realizado (palavra mais significativa do endereço) e o terceiro especifica o endereço da posição de memória ou o número do campo do descritor do processo corrente que será afetado pela execução da instrução "PLACE".

Após a leitura destes valores, o micro-programa realiza um desvio no seu fluxo de controle condicionado pelo segundo valor lido.

Se ele for nulo, o primeiro valor lido do topo da pilha é copiado na posição da memória principal apontada pelo terceiro valor lido do topo da pilha.

Se o segundo valor for negativo, o primeiro valor é copiado na posição da memória apontada pela diferença entre o valor do registrador "BTBASE" e o dobro do segundo valor lido do topo da pilha adicionado de seis unidades (endereço = BTBASE - (2 * segundo valor lido do topo da pilha + 6)).

Se o segundo valor for positivo, o primeiro valor é copiado no campo ERRSP da área de comunicação com o programa "MICMON" e o fluxo de controle do micro-programa é desviado para a seqüência de micro-instruções que comuta o controle para o micro-programa original e invoca o programa "MICMON" (vide sub-item e).

b) Seqüência de micro-instruções responsável
pela implementação da instrução "OBTAIN"

Esta seqüência é iniciada pela leitura de dois valores do topo da pilha seguida da realização de um desvio no fluxo de controle do micro-programa condicionado pelo primeiro valor lido.

Se ele for negativo, o segundo valor é duplicado e adicionado de seis unidades. A seguir o conteúdo do registrador "BTBASE" subtraído do resultado destas operações é utilizado como endereço de uma posição de memória.

O conteúdo desta posição de memória é lido para um registrador de trabalho e o fluxo de controle é desviado para a seqüência de micro-instruções que encerra as instruções "OBTAIN" e "SENSE".

Se o primeiro valor for zero, ocorre a leitura da palavra de memória apontada pelo segundo valor para um registrador de trabalho e o fluxo de controle do micro-programa é desviado para a seqÜência de micro-instruções que termina as instruções "OBTAIN" e "SENSE".

Se o primeiro valor lido for positivo, o conteúdo do registrador da micro-máquina cujo número é indicado pelo segundo valor lido é copiado em um registrador de trabalho e o fluxo de controle do micro-programa é desviado para a seqÜência de micro-instruções que termina as instruções "OBTAIN" e "SENSE".

Esta seqÜência é iniciada por um desvio no fluxo de controle do micro-programa condicionado pelo bit mais significativo do campo OPERAÇÃO do código da instrução para decidir se a instrução que está sendo executada é um "OBTAIN" ou "SENSE". Se for um "OBTAIN", o valor do registrador de trabalho para o qual foi copiado o valor lido da memória principal ou de um dos registradores da micro-máquina é colocado no topo da pilha.

Se a instrução que está sendo terminada for um "SENSE", o valor mencionado no parágrafo anterior sofre uma operação de disjunção ("E" lógico) com o conteúdo de um registrador de trabalho que foi atualizado no início da execução da instrução "SENSE".

Se o resultado desta operação for zero, a constante booleana "false" é copiada no topo da pilha. Caso contrário, a constante booleana "true" é colocada no topo da pilha.

c) SeqÜência de micro-instruções responsável
pela execução da instrução "SENSE"

Esta seqÜência é iniciada pela leitura de um valor do topo da pilha para um registrador de trabalho que será utilizado no término da execução desta instrução.

A seguir o fluxo de controle do micro-programa é desviado para a seqÜência de micro-instruções que interpreta a instrução "OBTAIN" (vide item *b*).

d) SeqÜência de micro-instruções destinada a
carregar os registradores da máquina básica

Esta seqÜência realiza a cópia dos valores dos campos "BT", "SP" e "PC" da área de comunicação do micro-programa com o programa "MICMON" para os registradores "BTBASE", "SP" e "PC" da máquina básica.

e) SeqÜência de micro-instruções destinada a
invocar o programa "MICMON" (instrução "TRAP")

Esta seqÜência inicia com a cópia do conteúdo de "IR" para o campo "ERRIR" da área de comunicação do micro-programa com o programa "MICMON".

A seguir o fluxo de controle do micro-programa é desviado para a seqÜência de micro-instruções que comuta o controle da micro-máquina para o micro-programa original (vide sub-item seguinte).

f) SeqÜÊncia de micro-instruções responsável
pela comutação do controle da micro-máquina

Esta seqÜÊncia começa com a cópia dos valores correntes dos registradores "BTBASE", "SP" e "PC" para a área de comunicação do micro-programa com o programa "MICMON".

A seguir os campos "L" e "P" desta área de comunicação são transferidos para os respectivos registradores da micro-máquina (vide ítem VI.5.2).

Finalmente, os campos "STDSWMASC" e "STDSWADDR" da área de comunicação são lidos, e o valor do campo "STDSWMASC" é copiado na posição da memória principal apontada pelo valor do campo "STDSWADDR".

Nesta posição está mapeado o registrador que indica qual das micro-memórias está com o controle da micro-máquina (vide ítem IV.3.) e o valor armazenado no campo "STDSWMASC" corresponde ao padrão de bits que deve ser colocado neste registrador afim de que o controle da micro-máquina retorne ao micro-programa original. Assim, a realização desta última operação causa uma imediata troca do controle da micro-máquina entre os dois micro-programas, permitindo que o programa "MICMON" passe a ser executado (a próxima instrução deste programa a ser executada é a apontada pelo conteúdo do campo "P" da área de comunicação entre o novo micro-programa e o programa "MICMON").

VI.5.6. A INTERFACE DO PROGRAMA "MICMON" COM O NOVO MICRO-PROGRAMA

Tão logo o micro-programa original do computador MITRA-15 assuma o controle da micro-máquina, o programa "MICMON" passa a ser executado a partir de uma instrução pré-fixada.

Esta instrução realiza um desvio no fluxo de controle do programa "MICMON" condicionado pelo valor do "byte" mais significativo do campo ERRIR da área de comunicação do novo micro-programa com este programa.

Se este valor for zero, o programa "MICMON" conclui que o programa EDISON que estava sendo executado terminou normalmente. Neste caso o programa "MICMON" verifica se tinha sido colocada alguma instrução "TRAP" (para implementar "breakpoints") no programa cuja execução está sendo encerrada, substituindo esta eventual instrução "TRAP" pela instrução que havia sido removida para lhe dar lugar.

A seguir o programa "MICMON" se prepara para receber um comando do operador (imprimindo uma mensagem do tipo "prompt" na teletipo principal do computador MITRA-15).

Se o valor do "byte" mais significativo do campo ERRIR da área de comunicação não for zero, o programa "MICMON" realiza outro desvio, desta vez condicionado pelo valor dos bits de peso quatro e oito deste valor.

Se o bit de peso oito for zero, o programa "MICMON" conclui que a execução do micro-programa foi interrompida durante a realização de uma instrução do núcleo. Isto indica que um erro foi encontrado. Neste caso o programa "MICMON" imprime uma mensagem explicativa do erro encontrado e a seguir se prepara para receber um comando do operador (imprimindo uma mensagem do tipo "prompt" na teletipo).

Esta mensagem de erro indica o tipo de instrução na qual foi detetado o erro e o conteúdo dos registradores no momento em que o erro foi detetado (estes valores são salvos pelo novo micro-programa

nas posições "ERRSP", "ERRPC" e "ERRBTBASE" da área de comunicação no momento em que o erro foi detetado - vide item VI.4.). Além disto, o programa "MICMON" informa ainda se o processo causador do erro estava ou não em região crítica. No caso de erros fatais, é ainda impressa uma mensagem indicando esta característica do erro.

Se o valor do bit de peso oito do "byte" mais significativo do campo ERRIR for um e o valor do bit de peso quatro deste "byte" for zero, o programa "MICMON" conclui que a execução do micro-programa foi interrompida durante a execução de uma instrução "PLACE" (vide sub-item a do item VI.5.5.).

Isto indica que uma chamada ao supervisor foi requerida pelo programa EDISON. Para realizar esta chamada o programa "MICMON" copia para os registradores "A", "E" e "X" os valores dos campos correspondentes do descritor do processo que estava sendo executado quando o micro-programa foi interrompido. A seguir, a rotina do programa supervisor cujo número corresponde ao valor armazenado no campo ERRSP da área de comunicação é invocada. Ao final da execução desta rotina do programa supervisor o programa "MICMON" armazena os valores dos registradores "A", "E" e "X" nos campos correspondentes do descritor do processo que executou a chamada ao supervisor e a seguir retorna o controle da micro-máquina para o novo micro-programa.

Para realizar este retorno o programa "MICMON" copia o padrão de bits que comanda a utilização da nova memória de controle para o registrador que seleciona qual das memórias de controle está sendo utilizada. Tão logo o registrador muda de valor, o novo micro-programa assume o controle da micro-máquina.

Como o valor do registrador "PC" da máquina básica é

armazenado no mesmo registrador da micro-máquina que é utilizado para armazenar o valor "P" da máquina virtual implementada pelo micro-programa original do computador MITRA-15, a primeira instrução interpretada pelo novo micro-programa pertence ao programa "MICMON" e é uma instrução "INI". Assim, no novo micro-programa, os valores dos campos "BT", "SP" e "PC" da área de comunicação são copiados para os registradores "BTBASE", "SP" e "PC" respectivamente, permitindo que o programa EDISON que havia sido interrompido para realizar a chamada ao programa supervisor continue sua execução.

Caso os valores dos bits de peso quatro e oito do "byte" mais significativo do campo ERRIR tenham valor um, o programa "MICMON" conclui que o micro-programa foi interrompido por ter atingido um "breakpoint". Neste caso o programa "MICMON" imprime uma mensagem que assinala este fato e substitui a instrução "TRAP" que tinha sido incluída no programa para implementar o "breakpoint" pela instrução que havia sido removida para lhe dar lugar.

A seguir o programa "MICMON" se prepara para receber um comando do operador (imprimindo na teletipo uma mensagem do tipo "prompt").

VII. AS CONCLUSOES E PROPOSTAS

Neste trabalho foi apresentada uma máquina básica definida para a linguagem EDISON, bem como a forma intermediária utilizada pelo seu interpretador.

Esta máquina pode ser implementada em ambientes que disponham de um ou mais processadores.

Quatro implementações desta máquina básica (utilizando programas de diversos níveis) foram realizadas.

Foi possível constatar que a máquina básica implementada no nível de micro-programa no computador MITRA-15 é cerca de cem vezes mais eficiente em termos de uso do processador que a implementação desta mesma máquina básica neste mesmo computador utilizando um programa codificado em "assembly".

Não foi possível comparar o esforço necessário para levar a cabo estas duas implementações porque o próprio "software" de apoio às atividades relacionadas à micro-programação foi desenvolvido em paralelo com o micro-programa responsável pela realização desta máquina.

Assim a construção do "software" de apoio às atividades de micro-programação do computador MITRA-15 e do micro-programa que realiza a máquina básica consumiu dez meses de esforço enquanto que a construção da implementação a nível de programa "assembly" consumiu cerca de dois meses.

Afim de realizar uma comparação criteriosa das quantidades de esforços necessários para realizar implementações em nível de programa "assembly" e de micro-programa, seria interessante definir

agora uma outra máquina básica e a seguir implementá-la nestes dois níveis.

Como o "software" de suporte à micro-programação do computador MITRA-15 já está, atualmente, disponível, esta comparação não seria viciada pelo tempo necessário à construção e depuração deste "software".

A nível do projeto EDISON é necessário realizar uma revisão das especificações da máquina básica aqui definida de forma a produzir uma versão definitiva e a seguir adaptar as diversas implementações realizadas a esta especificação final.

Esta revisão é indispensável porque ao longo do período de tempo utilizado no desenvolvimento das diversas implementações foram encontradas algumas deficiências na definição original que foram sanadas entre implementações consecutivas. Assim, a máquina básica realizada em cada uma das implementações é ligeiramente diferente das demais.

Estas diferenças incluem a ordem na qual os operandos imediatos são apresentados para as instruções e os códigos atribuídos a cada uma das instruções.

Outra idéia surgida ao longo deste trabalho, inspirada em um artigo de Flynn e Hoevel (8), envolveria uma alteração mais profunda no repertório utilizado de modo a dotar algumas instruções de uso mais freqüente (p.ex. as instruções aritméticas, lógicas e de comparação) de variações que ao invés de operarem sobre operandos obtidos exclusivamente da pilha (e devolvendo valores também na pilha) fossem capazes de operar diretamente sobre valores de variáveis (utilizando para tanto de um a três endereços).

Esta estratégia permitiria que as sentenças do tipo A := B + C; (que são executadas com muita freqüência nos programas) fossem codificadas (e posteriormente executadas) com apenas uma instrução ao invés das quatro instruções necessárias para a sua realização utilizando o conjunto de instruções definido para a máquina básica atual.

Ainda nesta mesma linha de atuação, seria conveniente a realização de estudos mais aprofundados a respeito dos tipos de endereçamento a serem utilizados afim de determinar se os oito tipos adotados são os mais convenientes e, caso contrário, quais tipos deveriam ser suprimidos ou acrescentados.

VIII. BIBLIOGRAFIA

(1) HANSEN, P. B., "Edison - a multiprocessor language", "Software - Practice and Experience", vol 12, pp 325 - 361 (1981)

(2) ZANCANELLA, L. C., Um processador Edison, Tese M.Sc, COPPE/UFRJ (1983)

(3) ALGOL B7000/B8000, "Language reference manual", "Burroughs Corporation" (1974)

(4) MITRA-15, "Manuel de référence", "Compagnie Internationale pour L'informatique" (1972)

(5) BREWEE, LECONTE, THIERON, "Ordinateur MITRA-15 - Organes Centraux", "Compagnie Internationale pour L'informatique" (1972)

(6) SALISBURY, A. B., "Microprogramable Computer Architectures", "American Elsevier Publishing Company" (1976)

(7) MITRA-15, "Moniteur Temps Réel Disque - MTRD", "Compagnie Internationale pour L'informatique" (1975)

(8) FLYNN, M. J., MOEVEL, L. W., "Execution Architecture: The DELtran Experiment", "IEEE Transactions on computers", vol C-32, no. 2, pp 156 - 174 (February 1983)

(9) MITRA-15, "SGF15 - UGF15 Manuel d'utilization", "Compagnie Internationale pour L'informatique" (1976)

(10) MITRA-15, "Bibliothécaire BIB - Manuel d'utilization", "Compagnie Internationale pour L'informatique" (1975)

(11) MITRA-15, "FORTRAN IV - Manuel d'utilization", "Compagnie Internationale pour L'informatique" (1975)

(12) VASCONCELOS, N. Q., Uma Máquina Básica EDISON, Anais do III Congresso da Sociedade Brasileira de Computação, vol 1, pp 283 - 305 (1983)

(13) HANSEN, P. B., "The Architecture of Concurrent Programs", Prentice-Hall INC (1977)

(14) HANSEN, P. B., "Operating System Principles", Prentice-Hall INC (1973)

(15) HANSEN, P. B., "The Design of EDISON", "Software - Practice and Experience", vol 12, pp 363 - 396 (1981)

(16) HANSEN, P. B., "EDISON Programs", "Software - Practice and Experience", vol 12, pp 397 - 410 (1981)

(17) HANSEN, P. B., "Programing a Personal Computer", Prentice-Hall INC (1982)

ANEXO I. PARAMETROS DEPENDENTES DE IMPLEMENTAÇÃO ADOTADOS

1) Variáveis do tipo "bool" e do tipo "char" são armazenadas em um "byte" (8 bits)

2) Variáveis de tipo "enum" e do tipo "int" são armazenadas em dois "bytes" (16 bits). O menor número inteiro é -32768 e o maior é 32767.

3) Variáveis do tipo "set" são armazenadas em 32 "bytes" (256 bits). Os valores dos elementos pertencentes a estes conjuntos devem estar no intervalo [0, 255].

4) Os endereços de memória são compostos por até quatro "bytes" (32 bits).

5) As constantes de processos informam o número de grupos de 256 "bytes" de memória a serem alocados para o processo.

ANEXO II. TABELA DAS INSTRUÇÕES UTILIZADAS

Na tabela apresentada a seguir estão relacionadas todas as instruções utilizadas, acompanhadas de seus mnemônicos, operandos imediatos e ações implementadas.

As instruções cujos mnemônicos estão assinalados com * não estavam previstas na definição original da máquina básica EDISON e não são geradas pelo compilador EDISON presentemente utilizado.

Os tipos de operandos imediatos utilizados pelas instruções são os seguintes:

1. Deslocamento ("DSP") - Informa a distância (em "bytes" de uma base a uma variável. É composto por 24 bits (nas implementações realizadas no computador NITRA-15 apenas os 16 bits menos significativos deste operando são consideradas).

2. Nível ("LV") - Informa a base a ser utilizada na obtenção de um endereço. É composto por 16 bits.

3. Tamanho ("SZ") - Informa o tamanho de uma variável ou cadeia (em "bytes" ou palavras, dependendo da instrução). É composto por 16 bits.

4. Limite inferior ("LB") - Informa o limite inferior do intervalo de índices válidos para um arranjo. É composto por 16 bits.

5. Limite superior ("UB") - Informa o limite superior do intervalo de índices válidos para um arranjo. É composto por 16 bits.

Nas descrições das ações implementadas pelas instruções foram adotadas as seguintes convenções:

1. Os símbolos DSP, LV, SZ, LB e UB representam os valores dos possíveis operandos imediatos das instruções.

2. Os símbolos SP, PC e BASE estão associados aos registradores do interpretador e o símbolo STKBASE representa um registrador (de uso opcional) destinado a armazenar o limite inferior da região da pilha utilizável. Estes símbolos podem ser usados para referenciar os valores armazenados nestes registradores, bem como na representação de atribuições de valores a estes registradores.

3. Constantes inteiras são representadas pelos seus valores decimais.

4. O símbolo BLANC representa o valor associado ao caracter branco (espaço).

5. O símbolo AUX representa um registrador de rascunho que tem propriedades idÊnticas às dos registradores do interpretador.

6. O símbolo SZ* representa o menor número inteiro par maior ou igual ao operando imediato SZ (SZ + SZ MOD 2).

7. A representação de um valor entre colchetes ("[ ]") seguido de um outro valor está associada a uma referÊncia a um número de "bytes" da memória principal igual ao valor colocado a seguir aos colchetes, a partir da posição de memória apontada pelo valor colocado entre os colchetes. Esta representação é utilizada tanto nas referÊncias aos valores armazenados nestas posições de memória, quanto na representação de uma atribuição de valores a estas posições.

8. A representação de um valor seguido por um ponto e de um outro valor significa a referÊncia a um bit do primeiro valor cujo número de ordem (a partir do bit menos significativo, que tem número de

ordem zero) corresponde ao segundo valor. Esta representação é utilizada tanto em referências ao valor deste bit como em atribuições de valor a este bit.

9. As operações aritméticas de soma, subtração, multiplicação, divisão e obtenção do resto da divisão inteira são representadas respectivamente pelos símbolos +, -, *, / e MOD. Estes símbolos aparecem entre dois valores, gerando, assim, um novo valor que corresponde ao resultado da operação representada.

10. A colocação de uma barra sobre um valor corresponde à complementação lógica (negação) de todos os bits do valor original.

11. As operações de disjunção e conjunção ("E" lógico e "OU" lógico) são representadas pelos símbolos AND e OR. Estes símbolos aparecem entre dois valores, gerando um novo valor que corresponde ao resultado da operação representada.

12. As comparações menor, menor ou igual, maior, maior ou igual e diferente são representadas pelos símbolos <, <=, =, >, >=, <> respectivamente. Estes símbolos aparecem entre dois valores, gerando um novo valor, booleano, correspondente ao resultado da comparação representada.

13. A representação BT(VALOR), onde VALOR é um valor qualquer (usualmente LV) significa uma referência a uma posição da tabela de bases. Esta posição pode ter seu conteúdo (valor) referenciado ou sofrer atribuições.

14. As atribuições são representadas pelo símbolo da entidade que sofre a atribuição seguido de um sinal de atribuição (<-) e um valor. Nos casos em que a obtenção deste valor envolvem o uso de

várias operações aritméticas ou lógicas, foram utilizados parËntesis para explicitar a ordem na qual estas operações devem ser realizadas.

15. Os símbolos KERNEL e STOP representam, respectivamente, referËncias a ações do núcleo e o final de execução do processo.

16. As atribuições e as referËncias aos símbolos KERNEL e STOP denotam ações realizadas pelo núcleo.

17. A representação ?VALOR => AÇÃO é utilizada para indicar uma realização condicional de uma ação. Assim a ação que compõe esta representação só será realizada se o valor utilizado corresponder à constante booleana "true".

18. SeqÜËncias de ações são compostas por uma ou mais ações separadas por vírgulas.

19. A representação composta por um valor seguido de uma seqÜËncia de ações entre parËntesis indica que as ações que compõem esta seqÜËncia devem ser realizadas tantas vezes quantas for especificado pelo valor utilizado.

20. As funções desempenhadas por cada uma das instruções são representadas por seqÜËncias de ações. As ações que compõem estas seqÜËncias não precisam ser realizadas na ordem em que aparecem mas o resultado final desta execução deve ser equivalente ao obtido em uma execução nesta ordem.

1. Instruções para obtenção de endereços de variáveis

MNEMONICO	OPERANDOS IMEDIATOS	AÇOES EXECUTADAS
ADDRL	DSP	[SP]<(-BASE+DSP, SP<(-SP+4
ADDRG	DSP, LV	[SP]<(-BT(LV)+DSP, SP<(-SP+4
ADDRLI	DSP	[SP]<(-[BASE+DSP]4, SP<(-SP+4
ADDRGI	DSP, LV	[SP]<(-[BT(LV)+DSP]4, SP<(-SP+4
ADDRLX	DSP	[SP-2]<(-BASE+DSP+[SP-2]2, SP<(-SP+2
ADDRGX	DSP, LV	[SP-2]<(-BT(LV)+DSP+[SP-2]4, SP<(-SP+4
ADDRLIX	DSP	[SP-2]<(-[BASE+DSP]4+[SP-2]4, SP<(-SP+4
ADDRGIX	DSP, LV	[SP-2]<(-[BT(LV)+DSP]4+[SP-2]4, SP<(-SP+4

2. Instruções para consulta a valores de variáveis de um "byte"

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
PUSHBL	DSP	[SP]2(-[BASE+DSP]1, SP(-SP+2
PUSHBG	DSP, LV	[SP]2(-[BT(LV)+DSP]1, SP(-SP+2
PUSHBLI	DSP	[SP]2(-[[BASE+DSP]4]1, SP(-SP+2
PUSHBGI	DSP, LV	[SP]2(-[[BT(LV)+DSP]4]1, SP(-SP+2
PUSHBLX	DSP	[SP-2]2(-[BASE+DSP+[SP-2]2]1
PUSHBGX	DSP, LV	[SP-2]2(-[BT(LV)+DSP+[SP-2]2]1
PUSHBLIX	DSP	[SP-2]2(- [[BASE+DSP]4+[SP-2]2]1
PUSHBGIX	DSP, LV	[SP-2]2(- [[BT(LV)+DSP]4+[SP-2]2]1

3. Instruções para atribuição de valores a variáveis de um "byte"

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
POPBL	DSP	SP(-SP-2, [BASE+DSP]1(-[SP]1
POPBG	DSP, LV	SP(-SP-2, [BT(LV)+DSP]1(-[SP]1
POPBLI	DSP	SP(-SP-2, [[BASE+DSP]4]1(-[SP]1
POPBGI	DSP, LV	SP(-SP-2, [[BT(LV)+DSP]4]1(-[SP]1
POPBLX	DSP	SP(-SP-4, [BASE+DSP+[SP]2]1(-[SP+2]1
POPBGX	DSP, LV	SP(-SP-4, [BT(LV)+DSP+[SP]2]1(-[SP+2]1
POPBLIX	DSP	SP(-SP-4, [[BASE+DSP]4+[SP]2]1(-[SP+2]1
POPBGIX	DSP, LV	SP(-SP-4, [[BT(LV)+DSP]4+[SP]2]1(-[SP+2]1

4. Instruções para consulta a valores de variáveis de dois "bytes"

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
PUSHWL	DSP	[SP]2<-[[BASE+DSP]2], SP<-SP+2
PUSHWG	DSP, LV	[SP]2<-[[BT(LV)+DSP]2], SP<-SP+2
PUSHWLI	DSP	[SP]2<-[[[BASE+DSP]4]2], SP<-SP+2
PUSHWGI	DSP, LV	[SP]2<-[[[BT(LV)+DSP]4]2], SP<-SP+2
PUSHWLX	DSP	[SP-2]2<-[[BASE+DSP+[SP-2]2]2]
PUSHWGX	DSP, LV	[SP-2]2<-[[BT(LV)+DSP+[SP-2]2]2]
PUSHWLIX	DSP	[SP-2]2<- [[[BASE+DSP]4+[SP-2]2]2]
PUSHWGIX	DSP, LV	[SP-2]2<- [[[BT(LV)+DSP]4+[SP-2]2]2]

5. Instruções para atribuição de valores a variáveis de dois "bytes"

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
POPWL	DSP	SP(-SP-2, [BASE+DSP]2(-[SP]2
POPWG	DSP, LV	SP(-SP-2, [BT(LV)+DSP]2(-[SP]2
POPWLI	DSP	SP(-SP-2, [[BASE+DSP]4]2(-[SP]2
POPWGI	DSP, LV	SP(-SP-2, [[BT(LV)+DSP]4]2(-[SP]2
POPWLX	DSP	SP(-SP-4, [BASE+DSP+[SP]2]2(-[SP+2]2
POPWGX	DSP, LV	SP(-SP-4, [BT(LV)+DSP+[SP]2]2(-[SP+2]2
POPWLIX	DSP	SP(-SP-4, [[BASE+DSP]4+[SP]2]2(-[SP+2]2
POPWGIX	DSP, LV	SP(-SP-4, [[BT(LV)+DSP]4+[SP]2]2(-[SP+2]2

6. Instruções para consulta a valores de variáveis de tamanho arbitrário

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
PUSHSL	DSP, SZ	[SP]SZ←-[BASE+DSP]SZ, SP←-SP+SZ*
PUSHSG	DSP, LV, SZ	[SP]SZ←-[BT(LV)+DSP]SZ, SP←-SP+SZ*
PUSHSLI	DSP, SZ	[SP]SZ←-[[BASE+DSP]4]SZ, SP←-SP+SZ*
PUSHSGI	DSP, LV, SZ	[SP]SZ←-[[BT(LV)+DSP]4]SZ, SP←-SP+SZ*
PUSHSLX	DSP, SZ	SP←SP-2, AUX←-[SP]2, [SP]SZ←-[BASE+DSP+AUX]SZ, SP←-SP+SZ*
PUSHSGX	DSP, LV, SZ	SP←-SP-2, AUX←-[SP]2, [SP]SZ←-[BT(LV)+DSP+AUX]SZ, SP←-SP+SZ*
PUSHSLIX	DSP, SZ	SP←-SP-2, AUX←-[SP]2, [SP]SZ←-[[BASE+DSP]4+AUX]SZ, SP←-SP+SZ*
PUSHSGIX	DSP, LV, SZ	SP←-SP-2, AUX←-[SP]2, [SP]SZ←-[[BT(LV)+DSP]4+AUX]SZ SP←-SP+SZ*

7. Instruções para atribuição de valores a variáveis de tamanho arbitrário

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
POPSL	DSP, SZ	SP(-SP-SZ*, [BASE+DSP]SZ(-[SP]SZ
POPSG	DSP, LV, SZ	SP(-SP-SZ*, [BT(LV)+DSP]SZ(-[SP]SZ
POPSLI	DSP, SZ	SP(-SP-SZ*, [[BASE+DSP]4]SZ(-[SP]SZ
POPSGI	DSP, LV, SZ	SP(-SP-SZ*, [[BT(LV)+DSP]4]SZ(-[SP]SZ
POPSLX	DSP, SZ	SP(-SP-(2+SZ*), AUX(-[SP]2, [BASE+DSP+AUX]SZ(-[SP+2]SZ
POPSGX	DSP, LV, SZ	SP(-SP-(2+SZ*), AUX(-[SP]2, [BT(LV)+DSP+AUX]SZ(-[SP+2]SZ
POPSLIX	DSP, SZ	SP(-SP-(2+SZ*), AUX(-[SP]2, [[BASE+DSP]4+AUX]SZ(-[SP+2]SZ
POPSGIX	DSP, LV, SZ	SP(-SP-(2+SZ*), AUX(-[SP]2, [[BT(LV)+DSP]4+AUX]SZ(-[SP+2]SZ

8. Instruções aritméticas e lógicas

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
ADD		SP(-SP-2, [SP-2]2(-[SP-2]2+[SP]2
SUB		SP(-SP-2, [SP-2]2(-[SP-2]2-[SP]2
MTPY		SP(-SP-2, [SP-2]2(-[SP-2]2*[SP]2
DIV		SP(-SP-2, [SP-2]2(-[SP-2]2/[SP]2
MOD		SP(-SP-2, [SP-2]2(-[SP-2]2 MOD [SP]2
NOT		AUX(-0, ?[SP-2]2=0 =) AUX(-1, [SP-2]2(-AUX
OR		SP(-SP-2, [SP-2]2(-[SP-2]2 OR [SP]2
AND		SP(-SP-2, [SP-2]2(-[SP-2]2 AND [SP]2

9. Instruções para realizar comparações

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
LSS		SP(-SP-2, [SP-2]2(-[SP-2]2 < [SP]2
EQL		SP(-SP-2, [SP-2]2(-[SP-2]2 = [SP]2
LEQ		SP(-SP-2, [SP-2]2(-[SP-2]2 <= [SP]2
GTR		SP(-SP-2, [SP-2]2(-[SP-2]2 > [SP]2
DIFF		SP(-SP-2, [SP-2]2(-[SP-2]2 <> [SP]2
GEQ		SP(-SP-2, [SP-2]2(-[SP-2]2 >= [SP]2

10. Instruções para realização de operações com arranjos

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
IS	SZ	SP(-SP-(SZ*+SZ*), [SP]2 (- [SP]SZ = [SP+SZ*]SZ, SP(-SP+2
ISNT	SZ	SP(-SP-(SZ*+SZ*), [SP]2 (- [SP]SZ () [SP+SZ*]SZ, SP(-SP+2
CONSTS	SZ	[SP]SZ (- [PC]SZ, SP(-SP+SZ*, PC(-PC+SZ*
BLANCS	SZ	SZ ([SP]1(-BLANC, SP(-SP+1), SP(-SP+SZ.0
PACK	SZ	SP(-SP-(SZ+SZ), AUX(-SP, SZ ([SP]1(-[AUX]1, SP(-SP+1, AUX(-AUX+2), SP(-SP+SZ.0
INDEX	SZ, LB, UB	? [SP-2]2(LB =) STOP, ? [SP-2]2)UB =) STOP, [SP-2]2(-([SP-2]2-LB)*SZ

11. Instruções para realização das operações com conjuntos

MNEMONICO	OPERANDOS IMEDIATOS	AÇOES EXECUTADAS
EMPTYSET		[SP]32(-0, SP(-SP+32
INCLUD	SZ	SP(-SP-(SZ+SZ), AUX(-SP, SZ ([SP-32]32.[AUX]1(-1, AUX(-AUX+2)
INCLUD1 *		SP(-SP-2, [SP-32]32.[SP]1(-1
IN		SP(-SP-32, [SP-2]2(-[SP]32.[SP-2]2()0
UNION		SP(-SP-32, [SP-32]32(-[SP-32]32 OR [SP]32
INTER		SP(-SP-32, [SP-32]32(-[SP-32]32 AND [SP]32
DIFF		SP(-SP-32, [SP-32]32(-[SP-32]32 AND [SP]32

12. Instruções para atribuição de constantes e realização de desvios

MNEMONICO	OPERANDOS IMEDIATOS	AÇOES EXECUTADAS
ZERO *		[SP]2(-0, SP(-SP+2
TRUE *		[SP]2(-1, SP(-SP+2
CONSTB *		[SP]2(-[PC-1]1, SP(-SP+2
CONSTW		[SP]2(-[PC]2, SP(-SP+2, PC(-PC+2
CODEAO	DSP	[SP]4(-PC+DSP, SP(-SP+4
JP	DSP	PC(-PC+DSP
JPFALSE	DSP	SP(-SP-2, ? [SP]2=0 => PC(-PC+DSP
JPTRUE *	DSP	SP(-SP-2, ? [SP]2 => PC(-PC+DSP

13. Instruções para chamar, iniciar e terminar rotinas

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
ALOC	SZ	SP(-SP+SZ*
CALL	DSP	[SP]4(-PC, SP(-SP+4, PC(-PC+DSP
CALLI	DSP, LV	[SP]4(-PC, SP(-SP+4, PC(-[BT(LV)+DSP]4
PROCP *	LV, SZ	[SP]4(-BASE, [SP+4]4(-STKBASE, [SP+8]4(-BT(LV), [SP+12]2(-LV, SP(-SP+14, BASE(-SP-SZ-18, BT(LV)(-BASE
PROCPV	DSP, LV, SZ	[SP]4(-BASE, [SP+4]4(-STKBASE, [SP+8]4(-BT(LV), [SP+12]2(-LV, SP(-SP+14, BASE(-SP-SZ-18, BT(LV)(-BASE, SP(-SP+DSP
RET *		SP(-SP-4, PC(-[SP]4
RETP *	SZ	SP(-SP-18, PC(-[SP]4, BASE(-[SP+4]4, STKBASE(-[SP+8]4, BT([SP+16]2)(-[SP+12]4, SP(-SP-SZ
RETPV	DSP, LV, SZ	SP(-SP-DSP-18, PC(-[SP]4, BASE(-[SP+4]4, STKBASE(-[SP+8]4, BT([SP+16]2)(-[SP+12]4, SP(-SP-SZ

14. Instruções associadas ao gerenciamento de processos e operações especiais

MNEMONICO	OPERANDOS IMEDIATOS	AÇÕES EXECUTADAS
CREATEFAT	DSP	SP←PC+DSP, KERNEL
CREATESON	DSP, SZ	KERNEL
HALT		KERNEL
WHEN		KERNEL, [SP]2←0, SP←SP+2
WAIT		KERNEL, SP←SP-2
PLACE		SP←SP-6, [[SP]4]2←[SP+4]2
OBTAIN		SP←SP-2, [SP-2]2←[[SP-2]4]2
SENSE		SP←SP-4, AUX←([SP-2]4]2 AND [SP+2]2, [SP-2]2←AUX () 0

ANEXO III. PROGRAMA RESPONSAVEL PELA

IMPLEMENTAÇÃO DA MAQUINA BASICA EM ALGOL

```
$ RESET LIST SET LINEINFO SET NOBINDINFO RESET SEQ
BEGIN
BOOLEAN LISTINST;          %    INDICA SE HA' TRACE.
INTEGER MEMSIZE,           %    TAMANHO DA MEMORIA (  WORDS ).
        LVLIM,             %    NUMERO MAXIMO DE NIVEIS LEXICOS.
        TIMELIMIT,         %    TEMPO MAXIMO DA SIMULACAO.
        NSECTORPERTRK,     %    NUMERO DE SETORES POR TRILHA DO DSK.
        STEPTIME,          %    TEMPO NECESSARIO PARA AVANCAR UMA TRK.
        SEEKCONST,         %    TEMPO MINIMO DE SEEK.
        SEEKVAR,           %    PARTE VARIAVEL DO TEMPO DE SEEK.
        DMATRANSFERTIME,   %    TEMPO NECESSARIO PARA TRANSFERIR 128 W.
        WORDTRANSFCONSTT,  %    TEMPO PARA TRANSFERENCIA DO TERMINAL.
        WORDTRANSFVART,    %
        WAITADDEDTIME,     %    TEMPO ADICIONAL A INSTRUCAO WAIT.
        CREATEADDEDTIME,   %    TEMPO ADICIONAL A INSTRUCAO CREATE.
        COPYWORDADDTIME,   %    TEMPO ADICIONAL POR PALAVRA ( STRINGS )
        PACKADDEDTIME,     %
        ALOCADDEDTIME;     %     TEMPO ADICIONAL PARA MOVER BRANCOS.

INTEGER ARRAY
        INSTTIME [ 0 : 95 ],          %  TEMPO DE EXECUCAO DAS INSTRUCOES.
        TERMSETUPCONST [ 0 : 1 ],     %  TEMPO DE SET-UP DO TERMINAL.
        TERMSETUPVAR   [ 0 : 1 ];     %

INTEGER ARRAY UTILIZADAS [ 0 : 15 ]; % ESTATISTICA DE INST. UTILIZADAS.

FILE DISCO ( KIND=DISK, MAXRECSIZE=45, BLOCKSIZE=450,
             AREASIZE=1000, AREAS=200, FLEXIBLE, MYUSE=IO );

FILE TERM  ( KIND=DC, UNITS=CHARACTERS, MYUSE=IO,
             MAXRECSIZE=300, BLOCKSIZE=300, FILETYPE=3 );

FILE ERRORFILE ( KIND=DISK, PROTECTION=SAVE, FLEXIBLE,
                 MAXRECSIZE=14, BLOCKSIZE=420, AREASIZE=300, AREAS=10 );

REAL ARRAY DISKBUFF [ 0 : 42 ],   TERMBUFF [ 0 : 42 ];
```

```
$ SET PAGE

PROCEDURE EDSONMACHINE;
  BEGIN
  BOOLEAN NOERRORS, FINISH;    INTEGER CURRENTTIME;
  POINTER POKMEMBUFF, POKBUFF, PTERMMEMBUFF, PTERMBUFF;

  INTEGER PC,              %    CONTEXTO DOS PROCESSOS.
          SP,              %
          BASE,            %    ( 6 REGISTRADORES DE 4 BYTES ).
          BTBASE,          %
          STKLIM,          %    TOTAL = 24 BYTES.
          STKBASE;         %

  INTEGER DATALIM,         %    LIMITES DA CONFIGURACAO ATUAL.
          DATABASE,        %
          CODEBASE,        %    ( 4 REGISTRADORES DE 4 BYTES ).
          CODESIZE;        %

  INTEGER PROGRAMSTATUS,   %    STATUS DA EXECUCAO DO PROGRAMA.
          RUNNINGDSCADDR,  %
          FATHERDSCADDR;   %    ( 3 REGISTRADORES DE 4 BYTES ).

  INTEGER IRSZ,            %    OPERANDOS IMEDIATOS DAS INSTRUCOES.
          DSP,             %
          ADDR,            %    ( 4 REGISTRADORES DE 4 BYTES ).
          LVFAT;           %

  INTEGER ACC,             %    STATUS DA EXECUCAO DO PROGRAMA.
          SCRTCH1,         %
          SCRTCH2;         %    ( 3 REGISTRADORES DE 4 BYTES ).

  INTEGER ARRAY MEMORY [ 0 : MEMSIZE ];


  DEFINE LOWBIT = [  0 :  1 ] #,
         SIGNAL = [ 15 :  1 ] #,
         LOWHEX = [  3 :  4 ] #,
         HIGHEX = [  7 :  4 ] #,
         LOWB   = [  7 :  8 ] #,
         HIGB   = [ 15 :  8 ] #,
         LOWW   = [ 15 : 16 ] #,
         HIGW   = [ 31 : 16 ] #,
         ADWORD = [ 31 : 32 ] #;

  DEFINE CPUSTATUS = PROGRAMSTATUS . LOWW #,
         NUMOFSONS = PROGRAMSTATUS . HIGW #;

  DEFINE IR        = IRSZ . [ 15 : 16 ] #,
         SZ        = IRSZ . [ 31 : 16 ] #,
         SZLOWBIT  = IRSZ . [ 16 :  1 ] #,
         SZDIV2    = IRSZ . [ 31 : 15 ] #,
         IRLOWB    = IRSZ . [  7 :  8 ] #,
         IRHIGB    = IRSZ . [ 15 :  8 ] #,
         IRLOWHEX  = IRSZ . [ 11 :  4 ] #,
         IRHIGHEX  = IRSZ . [ 15 :  4 ] #,
         IR2BITS   = IRSZ . [  9 :  2 ] #;
```

```
DEFINE DSPLOW    = DSP  . [ 15 : 16 ] #,
       DSPHIG    = DSP  . [ 31 : 16 ] #;

DEFINE LV        = LVFAT . [ 15 : 16 ] #,
       FAT       = LVFAT . [ 31 : 16 ] #;

DEFINE ACC1      = ACC . [ 15 : 16 ] #,
       ACC2      = ACC . [ 31 : 16 ] #,
       ACC1SIGN  = ACC . [ 15 :  1 ] #,
       ACC2SIGN  = ACC . [ 31 :  1 ] #;

DEFINE WORD ( AD ) = MEMORY [ (AD) . [ 31 : 31 ] ] #;

DEFINE GETDOUBLEWORD ( AD ) =
       WORD ( AD ) . LOWW & WORD ( AD + 2 ) HIGW #;

DEFINE PUTDOUBLEWORD ( AD, R ) =
       BEGIN WORD ( AD ) := ( R ) . LOWW;
             WORD ( AD + 2 ) := ( R ) . HIGW;   END #;

DEFINE SZSTAR = SZ + IRSZ.[ 16 : 1 ] #;

DEFINE INVALIDDATAADDR ( AD ) = ( ( (AD) < DATABASE ) OR
                                  ( (AD) > DATALIM  ) ) #;

DEFINE INVALIDCODEADDR = ( ( PC > CODESIZE ) OR
                           ( PC.LOWBIT > 0 ) ) #;

DEFINE INVALIDPOP  = ( ( SP < STKBASE ) OR ( SP.LOWBIT > 0 ) ) #,
       INVALIDPUSH = ( ( SP > STKLIM  ) OR ( SP.LOWBIT > 0 ) ) #;

DEFINE TRUEVALUE = 48"00000000FFFF" #,
       NEGATE ( R ) = ( REAL ( NOT BOOLEAN ( R ) ) ) #,
       ISTRUE ( R ) = BOOLEAN ( R ) #,
       ISFALSE ( R ) = NOT BOOLEAN ( R ) #;

DEFINE FETCH ( R ) = IF INVALIDCODEADDR THEN ERROR(0) ELSE BEGIN
                     R:=WORD ( PC+CODEBASE );   PC:=*+2;   END #;

DEFINE INTLIM = 48"00000000FFFF" #;

DEFINE DOISA16 = 48"000000001000" #;

DEFINE READBYTE ( AD, R ) = IF INVALIDDATAADDR ( AD )
                            THEN ERROR(1)
                            ELSE IF (AD).LOWBIT = 0
                              THEN R:=WORD( AD ).LOWB
                              ELSE R:=WORD( AD ).HIGB #;

DEFINE WRITEBYTE ( AD, R ) = IF NOERRORS
                             THEN IF INVALIDDATAADDR ( AD )
                               THEN ERROR (1)
                               ELSE IF (AD).LOWBIT = 0
                                 THEN WORD ( AD ).LOWB := R
                                 ELSE WORD ( AD ).HIGB := R
                             ELSE BEGIN END #;

DEFINE READWORD ( AD, R ) = IF INVALIDDATAADDR ( AD ) OR
```

```
                                ( (AD).LOWBIT ) 0 )
                                 THEN ERROR(2)
                                 ELSE R:=WORD( AD ).LOWW #;

DEFINE WRITEWORD ( AD, R ) = IF NOERRORS
                                THEN IF INVALIDDATAADDR ( AD ) OR
                                        ( (AD).LOWBIT ) 0 )
                                        THEN ERROR(2)
                                        ELSE WORD( AD ).LOWW:=R
                                ELSE BEGIN END #;

DEFINE READADDR ( AD, R ) = IF ( (AD).LOWBIT = 0 ) AND
                                ( (AD) GEQ DATABASE ) AND
                                ( ( (AD) + 2 ) LEQ DATALIM )
                                THEN R := GETDOUBLEWORD ( (AD) )
                                ELSE ERROR(2) #;

DEFINE WRITEADDR ( AD, R ) = IF NOERRORS
                                THEN IF ( (AD).LOWBIT = 0 ) AND
                                        ( (AD) GEQ DATABASE ) AND
                                        ( ( (AD) + 2 ) LEQ DATALIM )
                                        THEN PUTDOUBLEWORD ( AD, R )
                                        ELSE ERROR(2)
                                ELSE BEGIN END #;

DEFINE POPWORD ( R ) = BEGIN    SP := * - 2;
                                IF INVALIDPOP
                                   THEN ERROR(3)
                                   ELSE R := WORD ( SP ).LOWW;   END #;

DEFINE PUSHWORD ( R ) = IF NOERRORS
                           THEN IF INVALIDPUSH THEN ERROR(4) ELSE BEGIN
                             WORD ( SP ).LOWW := R;
                             SP := SP + 2;      END
                           ELSE BEGIN END #;

DEFINE POPADDR ( R ) = BEGIN    SP := * - 4;
                                IF INVALIDPOP
                                   THEN ERROR ( 3 )
                                   ELSE R := GETDOUBLEWORD ( SP );
                                END #;

DEFINE PUSHADDR ( R ) = IF NOERRORS THEN BEGIN
                           SP := * + 2;
                           IF INVALIDPUSH
                             THEN ERROR ( 4 )
                             ELSE PUTDOUBLEWORD ( ( SP - 2 ), R );
                           SP := * + 2;
                           END ELSE BEGIN END #;

DEFINE HALTINST = KERNEL ( 1 ) #;

PROCEDURE KERNEL ( N );   VALUE N;   INTEGER N;   FORWARD;
```

```
$ SET PAGE

 PROCEDURE ERROR ( N );   VALUE N;   INTEGER N;
   IF NOERRORS THEN BEGIN
     VALUE ARRAY ERRTABPTR (  0,  2,  5,  9, 11, 13, 15, 20, 23,
                             30, 32, 36, 40, 44, 47, 56, 64, 69 );

     VALUE ARRAY ERRTAB
       ( "* CODI", "GO #++",                                          %  0, 0
         "* BYTE", " DE DA", "DO #++",                                %  1, 2
         "* PALA", "VRA DE", " DADO ", "#++++++",                     %  2, 5
         "'POP' ", "#++++++",                                         %  3, 9
         "'PUSH'", " #+++++",                                         %  4,11
         "NIVEL ", "#++++++",                                         %  5,13
         "ERRO A", "RITIME", "TICO (", " OVERF", "LOW )#",            %  6,15
         "DIVISA", "O POR ", "ZERO##",                                %  7,20
         "VALOR ", "INVALI", "DO PAR",                                %  8,23
                   "A ELEM", "ENTO D", "E CONJ", "UNTO##",
         "INDICE", " #+++++",                                         %  9,30
         "INDICE", " MAIOR", " QUE 2", "**16##",                      % 10,32
         "CONTEX", "TO DE ", "RETORN", "O #+++",                      % 11,36
         "'COBEG", "IN' AN", "INHADO", "######",                      % 12,40
         "'WHEN'", " ANINH", "ADO###",                                % 13,44
         "NAO HA", "' MEMO", "RIA DI", "SPONIV", "EL PAR"             % 14,47
                   "A INIC", "IAR PR", "OCESSO", "#######",
         "NAO HA", "' PROC", "ESSOS ", "CONTIN",                      % 15,56
                   "UAVEIS", " => DE", "ADLOCK", "#######",
         "INSTRU", "CAO 'W", "AIT' I", "NVALID", "A######",           % 16,64
         "INSTRU", "CAO IN", "VALIDA", "#######"           );         % 17,69

   ARRAY ERRMSG [ 0 : 15 ];   POINTER P1, P2;

   HALTINST;   NOERRORS := FALSE;
   P1 := POINTER ( ERRMSG [ * ] );
   REPLACE P1 BY "..     " FOR 15 WORDS;
   P2 := POINTER ( ERRTAB [ ERRTABPTR [ N ] ] );
   IF P2 EQL "*" FOR 1 THEN BEGIN
     REPLACE P1:P1 BY "ENDERECO DE";
     P2 := P2 + 1;
     END;
   REPLACE P1:P1 BY P2:P2 UNTIL EQL "#";
   IF ( P2 + 1 ) EQL "+" FOR 1 THEN REPLACE P1 BY "INVALIDO";
   WRITE (ERRORFILE,("**** ERRO TIPO ",J3,":"),N);
   WRITE (ERRORFILE,14,POINTER(ERRMSG[*]));
   WRITE (ERRORFILE,(X4,"CONTEXTO:"));
   WRITE (ERRORFILE,(X6,"PC=",H8,X2,"SP=",H8,X2,"BASE=",H8),
                        PC.ADWORD,  SP.ADWORD,  BASE.ADWORD);
   WRITE (ERRORFILE,(X6,"BTBASE=",H8,X2,"STKLIM=",H8,X2,"STKBASE=",H8),
                        BTBASE.ADWORD, STKLIM.ADWORD, STKBASE.ADWORD);
   WRITE (ERRORFILE,(X6,"DATALIM=",H8,X2,"DATABASE=",H8,X2,
                       "CODEBASE=",H8,X2,"CODESIZE=",H8),
                        DATALIM.ADWORD,   DATABASE.ADWORD,
                        CODEBASE.ADWORD,  CODESIZE.ADWORD);
   WRITE (ERRORFILE,(X6,"PROGRAMSTATUS=",H8,X2,
                       "RUNNINGDSCADDR=",H8,X2,
                       "FATHERDSCADDR=",H8,X2),
                        PROGRAMSTATUS.ADWORD,
                        RUNNINGDSCADDR.ADWORD,
```

```
                              FATHERDSCADDR.ADWORD);
    WRITE (ERRORFILE,(X6,"IR=",H4,X2,"SZ=",H4,X2,"DSP=",H8,X2,
                              "ADDR=",H8,X2,"LV=",H4,X2,"FAT=",H4),
                         IR, SZ, DSP.ADWORD, ADDR.ADWORD, LV, FAT);
    WRITE (ERRORFILE,(X6,"ACC1=",H4,X2,"ACC2=",H4,X2,
                              "SCRTCH1=",H8,X2,"SCRTCH2=",H8,//),
                         ACC1, ACC2, SCRTCH1.ADWORD, SCRTCH2.ADWORD);
    END ERROR;

  INTEGER PROCEDURE BT ( LV );   VALUE LV;   INTEGER LV;
    IF LV ) LVLIM THEN ERROR ( 5 )
                  ELSE BT := GETDOUBLEWORD  ( BTBASE + 4 * LV );
%   END BT;

  PROCEDURE GETADDRESS1;
    BEGIN
    FETCH ( DSPLOW );   DSPHIG := IRLOWB;
    IF (IRHIGB).[0:1] = 0 THEN ADDR := BASE + DSP ELSE BEGIN
      FETCH ( LV );
      ADDR := BT ( LV ) + DSP;
      END;
    IF (IRHIGB).[1:1] = 1 THEN READADDR ( ADDR, ADDR );
    END GETADDRESS1;

  PROCEDURE GETADDRESS2;
    IF (IRHIGB).[2:1] = 1 THEN BEGIN
      POPWORD ( SCRTCH1 );
      ADDR := * + SCRTCH1;
      END GETADDRESS2;

  PROCEDURE POPSTRING;
    BEGIN
    ACC1 := SZSTAR;                 %  ACC1 := SZ*
    SCRTCH2 := SP := SP - ACC1;     %  SP := SP - SZ*
    CURRENTTIME := * + COPYWORDADDTIME * ACC1;
    IF INVALIDPOP THEN ERROR ( 3 );   GETADDRESS2;
    IF ( ADDR.LOWBIT ) 0 ) OR ( ADDR ( DATABASE ) OR
       ( ( ADDR + ACC1 - 2 ) ) DATALIM ) THEN ERROR ( 2 );
    IF NOERRORS THEN BEGIN
      REPLACE POINTER ( WORD ( ADDR ) ) BY
               POINTER ( WORD ( SCRTCH2 ) ) FOR ( SZDIV2 ) WORDS;
      IF SZLOWBIT ) 0
        THEN WORD ( ADDR + SZ ).LOWB := WORD ( SP + SZ ).LOWB;
      END;
    END POPSTRING;

  PROCEDURE PUSHSTRING;
    BEGIN
    ACC1 := SZSTAR;   GETADDRESS2;
    CURRENTTIME := * + COPYWORDADDTIME * ACC1;
    SCRTCH2 := SP;   SP := SP + ACC1 - 2;
    IF INVALIDPUSH THEN ERROR ( 4 );
    IF ( ADDR.LOWBIT ) 0 ) OR ( ADDR ( DATABASE ) OR
       ( ( ADDR + ACC1 - 2 ) ) DATALIM ) THEN ERROR ( 2 );
    IF NOERRORS THEN BEGIN
      REPLACE POINTER ( WORD ( SCRTCH2 ) ) BY
               POINTER ( WORD (  ADDR   ) ) FOR ( SZDIV2 ) WORDS;
      IF SZLOWBIT ) 0
```

```
      THEN WORD ( SP ) := WORD ( ADDR+SZ ).LOWB;
    END;
  SP := * + 2;
  END PUSHSTRING;

PROCEDURE COPYSTRING;
  BEGIN
  ACC1 := SZSTAR - 2;          % ACC1 := SZ* - 2
  PC := * + ACC1;   SP := * + ACC1;
  CURRENTTIME := * + COPYWORDADDTIME * ACC1;
  IF INVALIDCODEADDR THEN ERROR ( 0 );
  IF INVALIDPUSH THEN ERROR ( 4 );
  IF NOERRORS THEN BEGIN
    REPLACE POINTER ( WORD ( SP-ACC1 ) ) BY
            POINTER ( WORD ( PC-ACC1 ) ) FOR ( SZDIV2 ) WORDS;
    IF SZLOWBIT > 0 THEN WORD ( SP ) := WORD ( PC ).LOWB;
    END;
  PC := * + 2;   SP := * + 2;
  END COPYSTRING;

PROCEDURE COMPARESTRING;
  BEGIN
  ACC1 := SZSTAR;   SP := SP - ACC1 - ACC1;
  CURRENTTIME := * + COPYWORDADDTIME * ACC1;
  IF INVALIDPOP THEN ERROR ( 3 ) ELSE BEGIN
    SCRTCH1 := SP;   SCRTCH2 := SP + ACC1;
    ACC1 := SZDIV2;   ACC2 := TRUEVALUE;
    WHILE ( ACC1 > 0 ) AND ( ACC2 > 0 ) DO BEGIN
      IF WORD ( SCRTCH1 ).LOWW NEQ WORD ( SCRTCH2 ).LOWW THEN ACC2:=0;
      ACC1 := ACC1 - 1;   SCRTCH1 := * - 2;   SCRTCH2 := * - 2;
      END;
    IF ( SZLOWBIT > 0 ) AND ( ACC2 > 0 )
      THEN IF WORD ( SCRTCH1 ).LOWB NEQ WORD ( SCRTCH2 ).LOWB
        THEN ACC2 := 0;
    END;
  ACC := ACC2;
  END COMPARESTRING;

INTEGER PROCEDURE CODEADDR ( N );   VALUE N;   INTEGER N;
  BEGIN
  IF DSP . [ 23:1 ] > 0 THEN DSP . [ 31:8 ] := 255
                        ELSE DSP . [ 31:8 ] := 0;
  CODEADDR := ( PC . ADWORD - N + DSP . ADWORD ) . ADWORD;
  END CODEADDR;
```

```
$ SET PAGE

 PROCEDURE ADD;
   BEGIN
   SCRTCH1 := 0;
   IF ACC1SIGN EQL ACC2SIGN
     THEN SCRTCH1.[ ( ( ACC1SIGN + 1 ).LOWBIT ) : 1 ] := 1;
   ACC := ACC1 + ACC2;   ACC.[47:32] := 0;
   IF SCRTCH1.[ ( ACC.SIGNAL ) : 1 ] EQL 1 THEN ERROR ( 6 );
   END ADD;

 INTEGER PROCEDURE COMP2 ( R );   VALUE R;   INTEGER R;
   COMP2 := ( ( REAL ( NOT BOOLEAN ( R ) ) ).LOWW + 1 ).LOWW;

 DEFINE SIGNRESULT = ( ACC1SIGN + ACC2SIGN ).LOWBIT #;

 DEFINE ABSVALUEACC1 = IF ACC1SIGN ) 0 THEN ACC1 := COMP2 ( ACC1 ) #;

 DEFINE ABSVALUEACC2 = IF ACC2SIGN ) 0 THEN ACC2 := COMP2 ( ACC2 ) #;

 PROCEDURE MTPY;
   BEGIN
   SCRTCH1 := SIGNRESULT;
   ABSVALUEACC1;   ABSVALUEACC2;   ACC := ACC1 * ACC2;
   IF ACC ) ( INTLIM + SCRTCH1 ) THEN ERROR ( 6 );
   IF SCRTCH1 ) 0 THEN ACC := COMP2 ( ACC );
   END MTPY;

 PROCEDURE DIVIDE;
   BEGIN
   SCRTCH1 := SIGNRESULT;
   ABSVALUEACC1;   ABSVALUEACC2;
   IF ACC2 EQL 0 THEN ERROR ( 7 ) ELSE ACC := ACC1 DIV ACC2;
   IF SCRTCH1 ) 0 THEN ACC := COMP2 ( ACC );
   END DIVIDE;

 PROCEDURE MODULE;
   BEGIN
   SCRTCH1 := ACC1SIGN;
   ABSVALUEACC1;   ABSVALUEACC2;
   IF ACC2 EQL 0 THEN ERROR ( 7 ) ELSE ACC := ACC1 MOD ACC2;
   IF SCRTCH1 ) 0 THEN ACC := COMP2 ( ACC );
   END MODULE;

 PROCEDURE ORWORD;
   ACC := ( REAL ( BOOLEAN ( ACC1 ) OR BOOLEAN ( ACC2 ) ) ).LOWW;

 PROCEDURE ANDWORD;
   ACC := ( REAL ( BOOLEAN ( ACC1 ) AND BOOLEAN ( ACC2 ) ) ).LOWW;

 PROCEDURE ARITMET;
   CASE IRLOWHEX OF BEGIN
     0: ADD;
     1: ACC2 := COMP2 ( ACC2 );   ADD;
     2: MTPY;
     3: DIVIDE;
     4: MODULE;
     5: BEGIN END;
```

```
      6: ORWORD;
      7: ANDWORD;
      END ARITMET;

  PROCEDURE LOGICALOP;
    IF ACC1 < ACC2
      THEN IF (IRLOWHEX).[0:1] EQL 0 THEN ACC:= 0 ELSE ACC:=TRUEVALUE
      ELSE IF ACC1 EQL ACC2
        THEN IF (IRLOWHEX).[1:1] EQL 0 THEN ACC:=0 ELSE ACC:=TRUEVALUE
        ELSE IF (IRLOWHEX).[2:1] EQL 0 THEN ACC:=0 ELSE ACC:=TRUEVALUE;
%   END LOGICALOP;

  PROCEDURE EMPTYSET;
    IF NOERRORS
      THEN IF ( SP.LOWBIT EQL 0 ) AND ( ( SP + 30 ) LEQ STKLIM )
        THEN THRU 16 DO BEGIN
          WORD ( SP ) := 0;
          SP := SP + 2;
          END
        ELSE ERROR ( 4 )
      ELSE BEGIN END;
%   END EMPTYSET

  PROCEDURE INCLUDE;
    BEGIN
    FETCH ( FAT );   SCRTCH1 := SP - 32 - FAT - FAT;
    IF ( BOOLEAN ( SP.LOWBIT  ) OR ( SCRTCH1 < STKBASE ) ) AND NOERRORS
      THEN ERROR ( 3 )
      ELSE WHILE NOERRORS AND FAT > 0 DO BEGIN
           SP := SP - 2;   FAT := FAT - 1;   ACC := WORD ( SP );
           WORD ( SCRTCH1 + ACC.HIGHEX ).[ (ACC.LOWHEX) : 1 ] := 1;
           END
    END INCLUDE;

  PROCEDURE INTEST;
    IF NOERRORS
      THEN IF ( SP.LOWBIT = 0 ) AND ( ( SP - 34 ) GEQ STKBASE )
        THEN BEGIN
          SP := * - 34;
          ACC := WORD ( SP ).LOWW;
          IF ACC.HIGB NEQ 0
            THEN ERROR ( 8 )
            ELSE BEGIN
              IF WORD ( SP + 2 + ACC.HIGHEX ).
                    [ ( ACC.LOWHEX ) : 1 ] EQL 0
                THEN ACC:=0
                ELSE ACC:=TRUEVALUE;
              PUSHWORD ( ACC );
              END;
          END
        ELSE ERROR ( 3 );
%   END INTEST;

  PROCEDURE SETOPER ( N );   VALUE N;  INTEGER N;
    IF NOERRORS
      THEN IF ( SP.LOWBIT = 0 ) AND ( ( SP - 64 ) GEQ STKBASE )
        THEN BEGIN
          SCRTCH1 := SP - 32;
```

```
        IF N = 0
          THEN THRU 16 DO BEGIN
            SP := SP - 2;   SCRTCH1 := SCRTCH1 - 2;
            WORD ( SCRTCH1 ) :=
              REAL ( BOOLEAN ( WORD ( SCRTCH1 ) ) OR
                     BOOLEAN ( WORD (   SP    ) ) ) . LOWW;
            END
          ELSE IF N = 1
            THEN THRU 16 DO BEGIN
              SP := SP - 2;   SCRTCH1 := SCRTCH1 - 2;
              WORD ( SCRTCH1 ) :=
                REAL ( BOOLEAN ( WORD ( SCRTCH1 ) ) OR
                       BOOLEAN ( WORD (   SP    ) ) ) . LOWW;
              END
            ELSE THRU 16 DO BEGIN
              SP := SP - 2;   SCRTCH1 := SCRTCH1 - 2;
              WORD ( SCRTCH1 ) :=
                REAL ( BOOLEAN ( WORD ( SCRTCH1 ) ) AND NOT
                       BOOLEAN ( WORD (   SP    ) ) ) . LOWW;
              END
          END
        ELSE ERROR ( 3 );

PROCEDURE PACKINST;
  BEGIN
  FETCH ( SZ );    SP := SP - SZ - SZ;   SCRTCH1 := SP;
  IF INVALIDPOP THEN ERROR ( 3 );
  IF NOERRORS THEN BEGIN
    THRU SZDIV2 DO BEGIN
      ACC.LOWB := WORD ( SCRTCH1 ) . LOWB;   SCRTCH1 := * + 2;
      ACC.HIGB := WORD ( SCRTCH1 ) . LOWB;   SCRTCH1 := * + 2;
      WORD ( SP ) := ACC . LOWW;             SP := * + 2;
      END;
    IF SZLOWBIT > 0 THEN BEGIN
      WORD ( SP ) := WORD ( SCRTCH1 ) . LOWB;
      SP := SP + 2;
      END;
    END;
  CURRENTTIME := * + ( SZSTAR * PACKADDEDTIME );
  END PACKINST;

PROCEDURE INDEX;
  BEGIN
  FETCH ( DSP );   DSP.HIGW := IRLOWB;
  FETCH ( FAT );   FETCH ( SZ );   POPWORD ( ACC );
  IF ( ACC < DSP ) OR ( ACC > SZ ) THEN ERROR ( 9 );
  IF NOERRORS THEN BEGIN
    ACC := ( ACC - DSP ) * FAT;
    IF ACC GEQ DOISA16 THEN ERROR ( 10 ) ELSE BEGIN
      WORD ( SP ) := ACC;
      SP := SP + 2;
      END;
    END;
  END INDEX;
```

```
$ SET PAGE

 PROCEDURE STORENEWBASE ( NEWBASE, AD );
   VALUE NEWBASE, AD;   INTEGER NEWBASE, AD;
   IF NOERRORS THEN BEGIN
     IF LV ) LVLIM
        THEN ERROR ( 5 )
        ELSE IF ( AD . LOWBIT = 0 ) AND
                ( AD GEQ BTBASE ) AND ( ( AD + 2 ) LEQ DATALIM )
                THEN BEGIN
                  WORD ( AD ) := NEWBASE . LOWW;
                  WORD ( AD + 2 ) := NEWBASE . HIGW;
                  END
                ELSE ERROR ( 2 );
   END STORENEWBASE;

 PROCEDURE PROC ( N );   VALUE N;   INTEGER N;
   IF N < 2
     THEN IF N = 0
       THEN BEGIN          %   CALLDIR
         FETCH ( DSPLOW );   DSPHIG := IRLOWB;
         PUSHADDR ( PC );   IF NOERRORS THEN PC := CODEADDR ( 4 );
         END CALLDIR
       ELSE BEGIN          %   CALLIND
         FETCH ( DSP );   DSP.HIGW := IRLOWB;
         FETCH ( LV );   ADDR := BT ( LV ) + DSP;
         PUSHADDR ( PC );   READADDR ( ADDR, PC );
         END CALLIND
     ELSE IF N = 2
       THEN BEGIN
         FETCH ( DSP );   DSP.HIGW := IRLOWB;
         FETCH ( LV );   FETCH ( SZ );
         PUSHADDR ( BASE );   PUSHADDR ( BT ( LV ) );
         PUSHWORD ( LV );   PUSHADDR ( STKBASE );
         BASE := SP - SZ - 18;   STKBASE := SP := SP + DSP;
         STORENEWBASE ( BASE, ( BTBASE + 4 * LV ) );
         IF SP ) STKLIM THEN ERROR ( 4 );
         END
       ELSE BEGIN
         FETCH ( DSP );   DSP.HIGW := IRLOWB;
         FETCH ( SZ );   SP := SP - DSP - 18;
         READADDR ( SP, PC );   READADDR ( SP+4, BASE );
         READADDR ( SP+8, ACC );   READWORD ( SP+12, LV );
         READADDR ( SP+14, STKBASE );    SP := SP - SZ;
         STORENEWBASE ( ACC, ( BTBASE + 4 * LV ) );
         IF ( SP < STKBASE ) OR ( STKBASE < ( BASE + 18 ) ) OR
            ( BASE < DATABASE ) THEN ERROR ( 11 );
         END PROC;
```

```
$ SET PAGE

 DEFINE CONTEXTSIZE = 32 + 4 * LVLIM #;

 PROCEDURE KERNEL ( N );   VALUE N;   INTEGER N;

   BEGIN

   DEFINE SAVECTX ( WHAT, DSCADDR, WHERE ) =
          BEGIN MEMORY [ ( DSCADDR.[15:15]+WHERE ) ] := WHAT.LOWW;
                MEMORY [ ( DSCADDR.[15:15]+WHERE+1 ) ] := WHAT.HIGW;
                END #;

   DEFINE LOADCTX ( WHAT, DSCADDR, WHERE ) =
          BEGIN WHAT.LOWW:=MEMORY [ ( DSCADDR.[15:15]+WHERE ) ].LOWW;
                WHAT.HIGW:=MEMORY [ ( DSCADDR.[15:15]+WHERE+1 ) ].LOWW;
                END #;

   DEFINE PCPTR           =  0 #,
          SPPTR           =  2 #,
          BASEPTR         =  4 #,
          STKLIMPTR       =  6 #,
          STKBASEPTR      =  8 #,
          PRIOSEQNUMPTR   = 10 #,
          LASTADPTR       = 12 #,
          NEXTADPTR       = 14 #;


   DEFINE NEWBTBASE ( DSCADDR ) = DSCADDR + 32 #;

   DEFINE INVALIDDSCADDR ( DSCADDR ) = ( ( DSCADDR.LOWBIT ) 0 ) OR
                                       ( DSCADDR ( DATABASE ) OR
                                       ( (DSCADDR+32) ) DATALIM ) )#;

   DEFINE CREATEINST  = 48"000000000054" #;

   PROCEDURE SAVECONTEXT ( DSCADDR );
     VALUE DSCADDR;   INTEGER DSCADDR;
     IF INVALIDDSCADDR ( DSCADDR ) THEN ERROR ( 2 ) ELSE BEGIN
       SAVECTX ( PC,              DSCADDR,       PCPTR       );
       SAVECTX ( SP,              DSCADDR,       SPPTR       );
       SAVECTX ( BASE,            DSCADDR,       BASEPTR     );
       SAVECTX ( STKLIM,          DSCADDR,       STKLIMPTR   );
       SAVECTX ( STKBASE,         DSCADDR,       STKBASEPTR  );
       SAVECTX ( FATHERDSCADDR,   DSCADDR,       NEXTADPTR   );
       LOADCTX ( ADDR,            FATHERDSCADDR, LASTADPTR   );
       SAVECTX ( DSCADDR,         FATHERDSCADDR, LASTADPTR   );
       SAVECTX ( DSCADDR,         ADDR,          NEXTADPTR   );
       SAVECTX ( ADDR,            DSCADDR,       LASTADPTR   );
       END SAVECONTEXT;

   PROCEDURE LOADCONTEXT ( DSCADDR );
     VALUE DSCADDR;   INTEGER DSCADDR;
     IF INVALIDDSCADDR ( DSCADDR ) THEN ERROR ( 2 ) ELSE BEGIN
       LOADCTX ( PC,      DSCADDR,  NEXTADPTR   );
       LOADCTX ( SP,      DSCADDR,  LASTADPTR   );
       SAVECTX ( PC,      SP,       NEXTADPTR   );
       SAVECTX ( SP,      PC,       LASTADPTR   );
```

```
    LOADCTX ( PC,       DSCADDR,  PCPTR       );
    LOADCTX ( SP,       DSCADDR,  SPPTR       );
    LOADCTX ( BASE,     DSCADDR,  BASEPTR     );
    LOADCTX ( STKLIM,   DSCADDR,  STKLIMPTR   );
    LOADCTX ( STKBASE,  DSCADDR,  STKBASEPTR  );
    BTBASE := NEWBTBASE ( DSCADDR );
    END LOADCONTEXT;

PROCEDURE DEQUEUE ( DSCADDR );
  VALUE DSCADDR;   INTEGER DSCADDR;
  IF INVALIDDSCADDR ( DSCADDR ) THEN ERROR ( 2 ) ELSE BEGIN
    LOADCTX ( SCRTCH1, DSCADDR, NEXTADPTR );
    LOADCTX ( SCRTCH2, DSCADDR, LASTADPTR );
    SAVECTX ( SCRTCH1, SCRTCH2, NEXTADPTR );
    SAVECTX ( SCRTCH2, SCRTCH1, LASTADPTR );
    END DEQUEUE;

PROCEDURE MUTEXBEGIN;   BEGIN END;

PROCEDURE MUTEXEND;     BEGIN END;

PROCEDURE CREATE;
  IF CPUSTATUS NEQ 0 THEN ERROR ( 12 ) ELSE BEGIN
    MUTEXBEGIN;   CPUSTATUS := 1;
    SAVECONTEXT ( FATHERDSCADDR );
    DO BEGIN
      FETCH ( DSP );   DSP.HIGW := IRLOWB;   FETCH ( SZ );
      RUNNINGDSCADDR := SP;   BTBASE := NEWBTBASE ( SP );
      STKBASE := SP := SP + CONTEXTSIZE;
      STKLIM := ( STKBASE - 2 ) + ( 256 * SZ );
      IF STKLIM ) DATALIM THEN ERROR ( 14 );
      SAVECONTEXT ( RUNNINGDSCADDR );   SP := STKBASE := BTBASE;
      BTBASE := NEWBTBASE ( FATHERDSCADDR );   PC := CODEADDR ( 6 );
      THRU LVLIM DO BEGIN
        READADDR ( BTBASE, ACC );
        PUSHADDR ( ACC );   BTBASE := BTBASE + 4;
        END;
      IF NOERRORS THEN BEGIN
        NUMOFSONS := NUMOFSONS + 1;
        SAVECTX ( PROGRAMSTATUS, RUNNINGDSCADDR, PRIOSEQNUMPTR );
        END;
      FETCH ( IR );   SP := STKLIM+2;
      CURRENTTIME := * + CREATEADDEDTIME;
      END UNTIL ( NOT NOERRORS ) OR ( IRHIGB NEQ CREATEINST );
    SAVECTX ( PC, FATHERDSCADDR, PCPTR );
    IF NOERRORS THEN BEGIN
      LOADCTX ( RUNNINGDSCADDR, FATHERDSCADDR, NEXTADPTR );
      LOADCONTEXT ( RUNNINGDSCADDR );
      END;
    CPUSTATUS := 2;   MUTEXEND;
    END CREATE;

PROCEDURE HALTINST;
  IF NOERRORS THEN BEGIN
    MUTEXBEGIN;   ACC := RUNNINGDSCADDR;
    IF CPUSTATUS ( 2 THEN FINISH := TRUE ELSE BEGIN
      IF CPUSTATUS = 3 THEN BEGIN
        CPUSTATUS := 1;
```

```
        LOADCONTEXT ( ACC );
        END;
      NUMOFSONS := NUMOFSONS - 1;   CPUSTATUS := 1;
      IF NUMOFSONS LEQ 0
        THEN BEGIN
          RUNNINGDSCADDR := FATHERDSCADDR;
          LOADCONTEXT ( RUNNINGDSCADDR );   CPUSTATUS := 0;
          END
        ELSE BEGIN
          LOADCTX ( ACC, FATHERDSCADDR, NEXTADPTR );
          IF ACC = FATHERDSCADDR THEN FINISH := TRUE ELSE BEGIN
            RUNNINGDSCADDR := ACC;
            LOADCONTEXT ( ACC );   CPUSTATUS := 2;
            END;
          END;
      END;
    MUTEXEND;
    END HALTINST;

PROCEDURE WHENINST;
  IF CPUSTATUS = 3 THEN ERROR ( 13 ) ELSE BEGIN
    MUTEXBEGIN;   CPUSTATUS:=1;
    PC := * - 2;   SAVECONTEXT ( RUNNINGDSCADDR );
    PC := * + 2;   CPUSTATUS:=3;   PUSHWORD ( 0 );
    END WHENINST;

PROCEDURE WAITINST;
  IF CPUSTATUS NEQ 3 THEN ERROR ( 16 ) ELSE BEGIN
    POPWORD ( ACC );   CPUSTATUS := 1;
    IF ISTRUE ( ACC ) THEN DEQUEUE ( RUNNINGDSCADDR ) ELSE BEGIN
      LOADCTX ( RUNNINGDSCADDR, FATHERDSCADDR, NEXTADPTR );
      LOADCONTEXT ( RUNNINGDSCADDR );
      CURRENTTIME := * + WAITADDEDTIME;
      END;
    IF RUNNINGDSCADDR = FATHERDSCADDR
      THEN CPUSTATUS := 0
      ELSE CPUSTATUS := 2;
    MUTEXEND;
    END WAITINST;

IF N < 2
  THEN IF N < 1
    THEN IF N < 0
      THEN BEGIN                %      INICIALIZACAO DO KERNEL.
        SAVECTX ( FATHERDSCADDR, FATHERDSCADDR, LASTADPTR );
        SAVECTX ( FATHERDSCADDR, FATHERDSCADDR, NEXTADPTR );
        SAVECONTEXT ( FATHERDSCADDR );
        LOADCONTEXT ( FATHERDSCADDR );
        END
      ELSE CREATE
    ELSE HALTINST
  ELSE IF N = 2 THEN WHENINST ELSE WAITINST;
END KERNEL;
```

```
$ SET PAGE

 PROCEDURE IO ( N );   VALUE N;   INTEGER N;
   BEGIN
   DEFINE NPERIF    = 2 #;
   DEFINE DISKCMDAD = 0 #;
   DEFINE TERMCMDAD = 8 #;
   DEFINE IOADINI   = 48"0000FFFFFFE0" #;

   REAL ARRAY AUXBUFFTERM [ 0 : 1 ];
   OWN BOOLEAN SOMEBODYWAITING;    OWN REAL SEMENTE;
   INTEGER POS, NEXTPOS, TIMETOSEEK, TERMSETUPTIME;
   OWN INTEGER ARRAY T [ 0:NPERIF ], NEXTWAITING [ 0:NPERIF ];
   OWN INTEGER ARRAY IOMAP [ 0 : 15 ];
   OWN INTEGER DKBFAD, TRACK, TERMBFAD, BYTECOUNT;
   OWN INTEGER FIRST, FREEIO, DISKAD;

   PROCEDURE IODONE ( N );    VALUE N;   INTEGER N;   FORWARD;

   DEFINE FIRSTBYTE  = AUXBUFFTERM [ 0 ] . [ 47 : 8 ] #,
          SECONDBYTE = AUXBUFFTERM [ 0 ] . [ 39 : 8 ] #;

   PROCEDURE IOINITIALIZE;
     BEGIN
     SOMEBODYWAITING := FALSE;   SEMENTE := TIME ( 1 );
     TRACK := NEXTWAITING [ 0 ] := 0;   FREEIO := 48"00000000FFFF";
     REPLACE POINTER ( IOMAP [ * ] ) BY 48"FF" FOR 6*16;
     END IOINITIALIZE;

   PROCEDURE ENQUEUE ( WHO, WHEN );
     VALUE WHO, WHEN;   INTEGER WHO, WHEN;
     BEGIN
     NEXTPOS := NEXTWAITING [ 0 ];   POS := 0;
     WHILE ( NEXTPOS NEQ 0 ) AND ( CURRENTTIME GEQ T [ NEXTPOS ] )
       DO BEGIN
         POS := NEXTPOS;
         NEXTPOS := NEXTWAITING [ POS ];
         END;
     NEXTWAITING [ POS ] := WHO;
     T [ WHO ] := WHEN + CURRENTTIME;   NEXTWAITING [ WHO ] := POS;
     SOMEBODYWAITING := TRUE;   FIRST := T [ NEXTWAITING [0] ];
     END ENQUEUE;

   PROCEDURE CHECKTIME;
     WHILE SOMEBODYWAITING AND FIRST LEQ CURRENTTIME DO BEGIN
       POS := NEXTWAITING [ 0 ];   NEXTPOS := NEXTWAITING [ POS ];
       NEXTWAITING [0] := NEXTPOS;    SOMEBODYWAITING := NEXTPOS NEQ 0;
       FIRST := T [ NEXTPOS ];   IODONE ( POS );
       END CHECKTIME;

   PROCEDURE DISKIOINI;        %   CONTROLADOR DE DISCO C/ DMA.
     IF ACC . [ 47 :47 ] = 0 THEN BEGIN
       DKBFAD . LOWW := IOMAP [ 2 ];   DKBFAD . HIGW := IOMAP [ 3 ];
       PDKBUFF := POINTER ( DISKBUFF [ * ] );   FREEIO . LOWB := 0;
       PDKMEMBUFF := POINTER ( WORD [ DKBFAD ] );
       DISKAD := IOMAP [ 5 ] * NSECTORPERTRK +
                 IOMAP [ 4 ] MOD NSECTORPERTRK;
       TIMETOSEEK := STEPTIME * ABS ( TRACK - IOMAP [ 5 ] ) +
```

```
                    SEEKCONST + SEEKVAR * RANDOM ( SEMENTE );
      ENQUEUE ( 1, TIMETOSEEK );   TRACK := IOMAP [ 5 ];
      IF ACC = 0 THEN BEGIN   % ESCRITA
        THRU 128 DO BEGIN
          PDKMEMBUFF := PDKMEMBUFF + 4;
          REPLACE PDKBUFF : PDKBUFF BY PDKMEMBUFF : PDKMEMBUFF FOR 2;
          END;
        WRITE ( DISCO [ DISKAD ], 43, POINTER ( DISKBUFF [*] ) );
        CURRENTTIME := * + DMATRANSFERTIME;   CHECKTIME;
        END;
      END DISKIOINI;

PROCEDURE TERMIOINI;
  IF ACC . [ 47 : 47 ] = 0 THEN BEGIN
    FREEIO . HIGB := 0 ;
    TERMBFAD . LOWW := IOMAP [ 10 ];
    TERMBFAD . HIGW := IOMAP [ 11 ];
    PTERMBUFF := POINTER ( TERMBUFF [ * ] );
    TERMSETUPTIME := TERMSETUPCONST [ ACC ] +
                     TERMSETUPVAR   [ ACC ] * RANDOM ( SEMENTE );
    ENQUEUE( 2, TERMSETUPTIME );   REPLACE PTERMBUFF BY " " FOR 256;
    IF ACC = 0 THEN BYTECOUNT := IOMAP [ 9 ] ELSE BEGIN
      READ ( TERM, 256, PTERMBUFF );
      BYTECOUNT := TERM . STATE . [ 47 : 20 ];
      IOMAP [ 9 ] := 0;
      END;
    END TERMIOINI;

PROCEDURE IODONE ( WHO );   VALUE WHO;   INTEGER WHO;
  IF WHO = 1
    THEN BEGIN                          %   DISCO
      IF IOMAP [ 0 ] = 1 THEN BEGIN     %   LEITURA
        READ ( DISCO [ DISKAD ], 43, PDKBUFF );
        THRU 128 DO REPLACE PDKMEMBUFF : PDKMEMBUFF BY
                    48"00" FOR 4, PDKBUFF : PDKBUFF FOR 2;
        CURRENTTIME := * + DMATRANSFERTIME;   CHECKTIME;
        END;
      IOMAP [ 0 ] := 48"00000000FFFF";   FREEIO . LOWB := 255;
      END
    ELSE IF BYTECOUNT ) 0               %    TERMINAL
      THEN BEGIN
        IF IOMAP [ 8 ] = 0
          THEN BEGIN                    %    ESCRITA
            FIRSTBYTE := WORD ( TERMBFAD ) . LOWB;
            SECONDBYTE := WORD ( TERMBFAD ) . HIGB;
            REPLACE PTERMBUFF : PTERMBUFF BY
                    POINTER ( AUXBUFFTERM [ * ] )
                    FOR MIN ( 2, BYTECOUNT );
            END
          ELSE BEGIN                    %    LEITURA
            REPLACE POINTER ( AUXBUFFTERM [ * ] ) BY
                    PTERMBUFF : PTERMBUFF FOR 2;
            WORD ( TERMBFAD ) . LOWB := FIRSTBYTE;
            WORD ( TERMBFAD ) . HIGB := SECONDBYTE;
            IF BYTECOUNT ) 1 THEN IOMAP [ 9 ] := * + 2
                             ELSE IOMAP [ 9 ] := * + 1;
            END;
        BYTECOUNT := * - 2;   TERMBFAD := TERMBFAD + 2;
```

```
        ENQUEUE ( 2, ( WORDTRANSFCONSTT +
                       WORDTRANSFVART   * RANDOM ( SEMENTE ) ) );
        END
      ELSE BEGIN
        IF IOMAP [ 8 ] = 0
          THEN WRITE ( TERM, IOMAP [9], POINTER ( TERMBUFF [*] ) );
        IOMAP [ 8 ]  := 48"00000000FFFF";  FREEIO . HIGB := 255;
        END IODONE;

PROCEDURE PLACE;
  BEGIN
  POPWORD ( ACC );
  POPADDR ( ADDR );
  IF ADDR < IOADINI
    THEN WRITEWORD ( ADDR, ACC )
    ELSE BEGIN
      ADDR := ADDR . [ 4 : 4 ];
      IF ( FREEIO . [ ADDR : 1 ] > 0 ) THEN BEGIN
        IOMAP [ ADDR ] := ACC;
        IF ADDR = TERMCMDAD THEN TERMIOINI;
        IF ADDR = DISKCMDAD THEN DISKIOINI;
        END;
      END;
  END PLACE;

PROCEDURE OBTAIN;
  BEGIN
  POPADDR ( ADDR );
  IF ADDR < IOADINI THEN READWORD ( ADDR, ACC )
                    ELSE ACC := IOMAP [ ADDR . [ 4 : 4 ] ];
  PUSHWORD ( ACC );
  END OBTAIN;

PROCEDURE SENSE;
  BEGIN
  POPWORD ( ACC1 );   POPADDR ( ADDR );
  IF ADDR LSS IOADINI THEN READWORD ( ADDR, ACC2 )
                      ELSE ACC2 := IOMAP [ ADDR . [ 4 : 4 ]  ];
  ANDWORD;   IF ACC NEQ 0 THEN ACC := TRUEVALUE;   PUSHWORD ( ACC );
  END SENSE;

IF N < 2
  THEN IF N < 1
    THEN IF N < 0
      THEN IOINITIALIZE
      ELSE PLACE
    ELSE OBTAIN
  ELSE IF N < 3
    THEN SENSE
    ELSE CHECKTIME;
END IO;
```

```
$ SET PAGE

 PROCEDURE ALOC;
   BEGIN
   FETCH ( SZ );   SCRTCH1 := SP + SZ;
   IF SCRTCH1 ) STKLIM THEN ERROR ( 4 ) ELSE WHILE SZ ) 0 DO BEGIN
     WORD ( SP ) . LOWW := 48"000000004040";
     SP := * + 2;   SZ := SZ - 2;   CURRENTTIME := * + ALOCADDEDTIME;
     END;
   END ALOC;

 PROCEDURE JUMPCONST ( N );   VALUE N;   INTEGER N;
   IF N < 2
     THEN BEGIN
       IF N < 1 THEN ACC := 0 ELSE POPWORD ( ACC );
       IF ISTRUE ( ACC ) THEN PC := PC + 2 ELSE BEGIN
         FETCH ( DSPLOW );   DSPHIG := IRLOWB;
         PC := CODEADDR ( 4 );
         END;
       END
     ELSE IF N < 3        %  CONSTW
       THEN BEGIN
         FETCH ( ACC );
         PUSHWORD ( ACC );
         END
       ELSE BEGIN         %  CONSTS
         FETCH ( SZ );
         COPYSTRING;
         END JUMPCONST;
```

```
$ SET PAGE

 PROCEDURE PUSHELEMENT;
   BEGIN
   GETADDRESS1;   GETADDRESS2;
   IF IRLOWHEX < 8 THEN READBYTE ( ADDR, ACC )
                   ELSE READWORD ( ADDR, ACC );
   PUSHWORD ( ACC );
   END PUSHELEMENT;

 PROCEDURE POPELEMENT;
   BEGIN
   POPWORD ( ACC );   GETADDRESS1;   GETADDRESS2;
   IF IRLOWHEX < 8 THEN WRITEBYTE ( ADDR, ACC )
                   ELSE WRITEWORD ( ADDR, ACC );
   END POPELEMENT;

 PROCEDURE PUSHPOPSTRING;
   BEGIN
   GETADDRESS1;   FETCH ( SZ );
   IF IRLOWHEX < 8 THEN PUSHSTRING ELSE POPSTRING;
   END PUSHPOPSTRING;

 PROCEDURE LOGICALARITIM;
   IF ( IRLOWHEX = 8 ) OR ( IRLOWHEX = 15 )
     THEN BEGIN
       FETCH ( SZ );   COMPARESTRING;
       IF IRLOWHEX = 15 THEN ACC := NEGATE ( ACC );
       PUSHWORD ( ACC );
       END
     ELSE BEGIN
       POPWORD ( ACC2 );
       IF IRLOWHEX = 5
         THEN PUSHWORD ( NEGATE ( ACC2 ) )
         ELSE BEGIN
           POPWORD ( ACC1 );
           IF IRLOWHEX < 8 THEN ARITMET ELSE LOGICALOP;
           PUSHWORD ( ACC );
           END;
       END LOGICALARITIM;

 PROCEDURE SETORARRAYORADDR;
   IF IRLOWHEX < 8
     THEN CASE IRLOWHEX OF BEGIN
       0: EMPTYSET;
       1: INCLUDE;
       2: INTEST;
       3: SETOPER ( 0 );
       4: SETOPER ( 1 );
       5: SETOPER ( 2 );
       6: PACKINST;
       7: INDEX;
       END
     ELSE BEGIN
       GETADDRESS1;
       GETADDRESS2;
       PUSHADDR ( ADDR );
       END SETORARRAYORADDR;
```

```
  PROCEDURE MISCELANEOUS;
    IF IRLOWHEX < 8
      THEN IF IRLOWHEX < 4
        THEN PROC ( IR2BITS )
        ELSE KERNEL ( IR2BITS )
      ELSE IF IRLOWHEX < 12
        THEN IF IRLOWHEX < 11 THEN IO ( IR2BITS ) ELSE ALOC
        ELSE JUMPCONST ( IR2BITS );
%   END MISCELANEOUS;

  PROCEDURE EXTRA;
    IF IRLOWHEX = 0 THEN BEGIN
      FETCH ( DSPLOW );   DSPHIG := IRLOWB;
      ACC := CODEADDR ( 2 );   PUSHADDR ( ACC );
      END EXTRA;
```

```
$ SET PAGE

 PROCEDURE FETCHANDDECODE;
   BEGIN
   FETCH ( IR );   NOERRORS := TRUE;
   IF LISTINST THEN WRITE ( ERRORFILE,
           ( "T=",  J12, X5,
             "PC=", J10, X5,     "IR=", H4, "H", X5,
             "SP=", J10, X5,     "TOPO=", H4, "H" ),
           CURRENTTIME,
           ( PC . ADWORD - 2 ), IR,
           SP . ADWORD, WORD ( SP . ADWORD - 2 ) . LOWW );
   UTILIZADAS [ IRHIGHEX ] . [ IRLOWHEX : 1 ] := 1;
   CURRENTTIME := * + INSTTIME [ IRHIGB ];
   IF IRHIGB ) 97 THEN ERROR (17) ELSE CASE IRHIGHEX OF BEGIN
     0: PUSHELEMENT;
     1: POPELEMENT;
     2: PUSHPOPSTRING;
     3: LOGICALARITIM;
     4: SETORARRAYORADDR;
     5: MISCELANEOUS;
     6: EXTRA;
     END;
   IO ( 4 );
   END FETCHANDDECODE;

 PROCEDURE INITIALIZE;
   BEGIN
   FILE CODE ( KIND=DISK, FILETYPE=7 );
   INTEGER ARRAY AUXLE [ 0 : 3 ];
   ARRAY REC [ 0:15 ];   FORMAT F ( 16 ( H4, X1 ) );   POINTER P;

   READ ( CODE, F, AUXLE [ * ] );
   PC . [15:16] := AUXLE [ 0 ];
   PC . [31:16] := AUXLE [ 1 ];
   CODESIZE . [15:16] := AUXLE [ 2 ];
   CODESIZE . [31:16] := AUXLE [ 3 ];

   CODEBASE := PROGRAMSTATUS := 0;
   DATABASE := CODEBASE + CODESIZE;
   STKLIM := DATALIM := 2 * MEMSIZE;
   BASE := STKBASE := DATABASE + CONTEXTSIZE;
   BTBASE := STKBASE - 4 * LVLIM;   SP := STKBASE + 4;
   RUNNINGDSCADDR := FATHERDSCADDR := DATABASE;

   P := POINTER ( MEMORY [ * ] );
   WHILE NOT READ ( CODE, F, REC [ * ] )
     DO REPLACE P : P BY POINTER ( REC [ * ] ) FOR 16 WORDS;

   KERNEL ( -1 );   FINISH := FALSE;   CURRENTTIME := 0;   IO ( -1 );
   END INITIALIZE;

 INITIALIZE;
 DO FETCHANDDECODE UNTIL FINISH;
 END EDSONMACHINE;
```

```
$ SET PAGE

PROCEDURE CONFIGURATION;
  BEGIN
  FILE IMP ( KIND=PRINTER );
  FILE CONTR ( KIND=DISK, FILETYPE=7 );
  INTEGER TEMTRACE;

  READ ( CONTR , / , TEMTRACE );

  LISTINST := TEMTRACE ) 0;

  IF LISTINST THEN WRITE ( IMP, ( "SERA' GERADO O 'TRACE'." ) );

  READ ( CONTR , / , MEMSIZE, LVLIM, NSECTORPERTRK, TIMELIMIT );

  WRITE ( IMP, ( X10, "TAMANHO DA MEMORIA: ", J20, " PALAVRAS.", /,
                  X10, "NUMERO MAXIMO DE NIVEIS LEXICOS: ", J10,  /,
                  X10, "NUMERO DE SETORES POR TRILHA :   ", J10,  /,
                  X10, "TEMPO MAXIMO DA SIMULACAO:       ", J20,  / ),
                MEMSIZE, LVLIM, NSECTORPERTRK, TIMELIMIT            );

  READ ( CONTR , / , STEPTIME, SEEKCONST, SEEKVAR, DMATRANSFERTIME );

  WRITE ( IMP, ( X5,  "TEMPOS DO DISCO:" ) );
  WRITE ( IMP, ( X10, "TEMPO DE 'STEP' ENTRE TRILHAS:    ", J20, /,
                  X10, "TEMPO MINIMO DE LEITURA/ESCRITA: ", J20, /,
                  X10, "TEMPO MEDIO DE LEITURA / ESCRITA: ", J20, /,
                  X10, "TEMPO DE TRANSF. DE 256 BYTES ( DMA ): ", J20 ),
                STEPTIME, SEEKCONST, SEEKVAR, DMATRANSFERTIME         );

  SEEKVAR := ( SEEKVAR - SEEKCONST ) * 2;

  READ ( CONTR , / , TERMSETUPCONST [ * ], TERMSETUPVAR [ * ],
                     WORDTRANSFCONSTT,     WORDTRANSFVART    );

  WRITE ( IMP, ( /, X5, "TEMPOS DO TERMINAL:" ) );
  WRITE ( IMP, ( X10, "TEMPO MINIMO ENTRE O COMANDO DE ESCRITA ",
                      "E O INICIO DA OPERACAO DE ESCRITA: ", J20, /,
                  X10, "TEMPO MINIMO ENTRE O COMANDO DE LEITURA ",
                      "E O INICIO DA OPERACAO DE LEITURA: ", J20, /,
                  X10, "TEMPO MEDIO ENTRE O COMANDO DE ESCRITA E",
                      " O INICIO DA OPERACAO DE ESCRITA:  ", J20, /,
                  X10, "TEMPO MEDIO ENTRE O COMANDO DE LEITURA E",
                      " O INICIO DA OPERACAO DE LEITURA:  ", J20, /,
                  X10, "TEMPO MINIMO PARA TRANSFERENCIA DE UMA PALAVRA ",
                      "DO TERMINAL PARA MEMORIA E VICE VERSA: ", J20, /,
                  X10, "TEMPO MEDIO PARA TRANSFERENCIA DE UMA PALAVRA ",
                      "DO TERMINAL PARA MEMORIA E VICE VERSA:  ", J20 ),
                TERMSETUPCONST [ * ], TERMSETUPVAR [ * ],
                WORDTRANSFCONSTT, WORDTRANSFVART                      );

  TERMSETUPVAR [0] := ( TERMSETUPVAR [0] - TERMSETUPCONST [0] ) * 2;
  TERMSETUPVAR [1] := ( TERMSETUPVAR [1] - TERMSETUPCONST [1] ) * 2;
  WORDTRANSFVART   := ( WORDTRANSFVART   - WORDTRANSFCONSTT   ) * 2;

  READ ( CONTR , / , PACKADDEDTIME, COPYWORDADDTIME,
                     CREATEADDEDTIME, WAITADDEDTIME, ALOCADDEDTIME );
```

```
   WRITE ( IMP, ( /, X5, "TEMPOS ADICIONAIS DE INTRUCOES:" ) );
   WRITE ( IMP, ( X10, "TEMPO ADICIONAL DA INSTRUCAO ",
                       "'PACK' (POR PALAVRA): ", J20, /,
                  X10, "TEMPO ADICIONAL DAS INSTRUCOES DOS TIPOS ",
                       "'PUSHS', 'POPS' E 'CONSTS' (POR PALAVRA): ", J20,
                /, X10, "TEMPO ADICIONAL DA INSTRUCAO ",
                       "'CREATE' (POR PROCESSO CRIADO): ", J20, /,
                  X10, "TEMPO ADICIONAL DA INSTRUCAO 'WAIT' (CASO ",
                       "A CONDICAO NAO TENHA SIDO ATENDIDA): ", J20, /,
                  X10, "TEMPO ADICIONAL DA INSTRUCAO 'ALOC' ",
                       "(POR PALAVRA ALOCADA): ", J20, / ),
                PACKADDEDTIME, COPYWORDADDTIME,
                CREATEADDEDTIME, WAITADDEDTIME, ALOCADDEDTIME );

   WRITE ( IMP [ SKIP 1 ] );
   READ ( CONTR, / , INSTTIME [ * ] );
   WRITE ( IMP, ( X5, "TEMPOS DE EXECUCAO DAS INSTRUCOES:" ) );
   WRITE ( IMP, *//, INSTTIME [ * ] );

   REPLACE POINTER ( UTILIZADAS [ * ] ) BY 48"00" FOR 96;

   END CONFIGURATION;

 PROCEDURE STATISTICS;
   BEGIN
 %
 %
 %       ESTA ROTINA REGISTRA OS VALORES FINAIS DE VARIAVEIS
 %       DE INTERESSE PARA AVALIACAO DA SIMULACAO REALIZADA.
 %
 %
   FILE STAT ( KIND=DISK, MAXRECSIZE=14, BLOCKSIZE=420,
               AREASIZE=300, AREAS=10, PROTECTION=SAVE );
   INTEGER INDICE;
   DEFINE EUTILIZADA ( I ) =
          UTILIZADAS [ I . [7:4] ] . [ ( I . [3:4] ) = 1 ] > 0 #;

   WRITE ( STAT, ( "CODIGO DAS INSTRUCOES UTILIZADAS:") );
   WRITE ( STAT, /, FOR INDICE := 0 STEP 1 UNTIL 255
                        DO IF EUTILIZADA ( INDICE ) THEN INDICE );
   CLOSE ( STAT, CRUNCH );

   END STATISTICS;

 CONFIGURATION;
 EDSONMACHINE;
 STATISTICS;
 END.
```

ANEXO IV. SINTAXE DA LINGUAGEM DE MICRO-PROGRAMAÇÃO DO MITRA-15

1. GENERALIDADES

1. Os micro-programas devem ser escritos como um texto composto por linhas de até 72 caracteres EBCDIC.

2. Os caracteres brancos inseridos neste texto não são considerados pelo micro-montador.

3. Um caracter % indica o fim lógico de uma linha. Todos os caracteres existentes entre a primeira ocorrência do caracter % e o fim da linha não são analisados pelo micro-montador, permitindo assim a inserção de comentários no texto do programa.

2. RESUMO DA SINTAXE

1. ⟨PROGRAMA⟩ ::= ⟨ASSERTIVA⟩ ; [⟨PROGRAMA⟩]

2. ⟨ASSERTIVA⟩ ::= ⟨DIRETIVA⟩ / ⟨INSTRUÇÃO COMPLETA⟩

3. ⟨DIRETIVA⟩ ::= ⟨RÓTULO⟩ : EQU ⟨PARÂMETRO⟩

4. ⟨INSTRUÇÃO COMPLETA⟩ ::= [⟨RÓTULO⟩ :]
   [⟨MNEMÔNICO DE OPERAÇÃO DE MEMÓRIA⟩]
   ⟨INSTRUÇÃO⟩

5. ⟨PARÂMETRO⟩ ::= (⟨OPERANDO⟩)

6. ⟨MNEMÔNICO DE OPERAÇÃO DE MEMÓRIA⟩ ::= LEIA / ESCREVA / REESCREVA

7. ⟨INSTRUÇÃO⟩ ::= ⟨INSTRUÇÃO COM PARÂMETRO⟩ /
   ⟨INSTRUÇÃO COM INDICAÇÃO DE SUCESSORA⟩ /
   ⟨INSTRUÇÃO COM PARÂMETRO E INDICAÇÃO DE SUCESSORA⟩ /
   ⟨INSTRUÇÃO COM MÚLTIPLA INDICAÇÃO DE SUCESSORA⟩

8. ⟨OPERANDO⟩ ::= ⟨RØTULO⟩ / ⟨NØMERO⟩

9. ⟨INSTRUÇÃO COM PARAMETRO⟩ ::=
⟨MNEMONICO DE INSTRUÇÃO COM PARAMETRO⟩ ⟨PARAMETRO⟩

10. ⟨INSTRUÇÃO COM INDICAÇÃO DE SUCESSORA⟩ ::=
⟨MNEMONICO DE INSTRUÇÃO COM INDICAÇÃO DE SUCESSORA⟩
[, ⟨OPERANDO⟩]

11. ⟨INSTRUÇÃO COM PARAMETRO E INDICAÇÃO DE SUCESSORA⟩ ::=
⟨MNEMONICO DE INSTRUÇÃO COM PARAMETRO E INDICAÇÃO DE SUCESSORA⟩
⟨PARAMETRO [, ⟨OPERANDO⟩]

12. ⟨INSTRUÇÃO COM MÚLTIPLA INDICAÇÃO DE SUCESSORA⟩ ::=
⟨MNEMONICO DE INSTRUÇÃO COM MÚLTIPLA INDICAÇÃO DE SUCESSORA⟩
⟨PARAMETRO⟩ [,] ⟨ESPECIFICAÇÃO DE MASCARA⟩
⟨LISTA DE INDICAÇÕES DE SUCESSORAS⟩

13. ⟨MNEMONICO DE INSTRUÇÃO COM PARAMETRO⟩ ::=
MCUD / MCUG / MCR / HALT / LD /
ED / AC / ACR / DMS / MCJ / ACIR

14. ⟨MNEMONICO DE INSTRUÇÃO COM INDICAÇÃO DE SUCESSORA⟩ ::=
=BBP / TI / P+2P / UXOU / =0B /
=1B / =SB / =PB / =10 / =UP /
=RJU / =UE / =UA / =BP / =ZP /
=0U / =1U / =IU / =URJ / M+1U /
=MU / =UIP / UOUO / OUUO / UUU3

15. ⟨MNEMONICO DE INSTRUÇAO COM PARAMETRO E INDICAÇAO DE SUCESSORA⟩ ::=

TT / TF / R+2U / R+2R / R+UU /

R+IU / R-UU / R-1R / R.UI / RIUI /

U/ØR / RXØR / =UR / =UI / =RU /

URR1 / URRØ / RURØ / LI / EI /

R+UØ / R+UR / R-UR / R-UI / R+NR /

R-NR / RQUR / RQUI / R*MU / R.UU /

=RZ / ØRR / =ØR / RXØR / RXSR /

U/SR / U/ØR / U/PR / RMUØ / MRU2 /

RMU1 / MRUØ / RUR2 / OUR2 / OURØ /

RRR3 / = RJ / R+1R / RQUU /

16. ⟨MNEMONICO DE INSTRUÇAO E MULTIPLA INDICAÇAO DE SUCESSORA⟩ ::=

BRD / BRG / MURT

17. ⟨ESPECIFICAÇAO DE MASCARA⟩ ::= MASCARA ⟨PARAMETRO⟩ [,]

18. ⟨LISTA DE INDICAÇOES DE SUCESSORAS ⟩ ::=

⟨OPERANDO OU $⟩ [, ⟨LISTA DE INDICAÇOES DE SUCESSORAS⟩]

19. ⟨OPERANDO OU $⟩ ::= ⟨OPERANDO⟩ / $

3. ELEMENTOS LEXICOS

1. ⟨RØTULO⟩ ::= ⟨CARACTER NAO ESPECIAL NEM NUMERICO⟩

[⟨SUFIXO DE RØTULO⟩]

2. ⟨CARACTER NAO ESPECIAL NEM NUMERICO⟩ ::=

! QUALQUER CARACTER EBCDIC VALIDO EXCETO

$ % : & ( ) , ; # = + - 0 1 2 3 4 5 6 7 8 9 !

3. ⟨SUFIXO DE RÓTULO⟩ ::= ⟨CARACTER NÃO ESPECIAL⟩
[⟨SUFIXO DE RÓTULO⟩]

4. ⟨CARACTER NÃO ESPECIAL⟩ ::=
! QUALQUER CARACTER EBCDIC VALIDO EXCETO % : & ( ) , ; !

5. ⟨NÚMERO⟩ ::= ⟨NÚMERO BINARIO⟩ / ⟨NÚMERO OCTAL⟩ /
⟨NÚMERO DECIMAL⟩ / ⟨NÚMERO HEXADECIMAL⟩

6. ⟨NÚMERO BINARIO⟩ ::= # B [=] [⟨SINAL⟩]
^
⟨SEQUÊNCIA DE ALGARISMOS BINÁRIOS⟩

7. ⟨NÚMERO OCTAL⟩ ::= # O [=] [⟨SINAL⟩]
^
⟨SEQUÊNCIA DE ALGARISMOS OCTAIS⟩

8. ⟨NÚMERO DECIMAL⟩ ::= [⟨ESPECIFICAÇÃO DE BASE 10⟩]
^
[⟨SINAL⟩] ⟨SEQUÊNCIA DE ALGARISMOS DECIMAIS⟩

9. ⟨NÚMERO HEXADECIMAL⟩ ::= # H [=] ⟨SINAL⟩]
^
⟨SEQUÊNCIA DE ALGARISMOS HEXADECIMAIS⟩

10. ⟨SINAL⟩ ::= + / −

^
11. ⟨SEQUÊNCIA DE ALGARISMOS BINÁRIOS⟩ ::=
^
⟨ALGARISMO BINÁRIO⟩ [⟨SEQUÊNCIA DE ALGARISMOS BINÁRIOS⟩]

^
12. ⟨SEQUÊNCIA DE ALGARISMOS OCTAIS⟩ ::=
^
⟨ALGARISMO OCTAL⟩ [⟨SEQUÊNCIA DE ALGARISMOS OCTAIS⟩]

13. ⟨ESPECIFICAÇÃO DE BASE 10⟩ ::= # [D] [=]

^
14. ⟨SEQUÊNCIA DE ALGARISMOS DECIMAIS⟩ ::=
^
⟨ALGARISMO DECIMAL⟩ [⟨SEQUÊNCIA DE ALGARISMOS DECIMAIS⟩]

15. ⟨SEQUENCIA DE ALGARISMOS HEXADECIMAIS⟩ ::=
⟨ALGARISMO HEXADECIMAL⟩ [⟨SEQUENCIA DE ALGARISMOS HEXADECIMAIS⟩]

16. ⟨ALGARISMO BINARIO⟩ ::= 0 / 1

17. ⟨ALGARISMO OCTAL⟩ ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7

18. ⟨ALGARISMO DECIMAL⟩ ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9

19. ⟨ALGARISMO HEXADECIMAL⟩::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 /
8 / 9 / A / B / C / D / E / F

4. LIMITES DO MICRO-ASSEMBLER

1. NENHUM PROGRAMA PODE TER MAIS QUE:

   a) 1024 instruções

   b) 4001 rótulos

   c) 20000 linhas

2. NENHUM ROTULO OU NUMERO PODE SER COMPOSTO POR MAIS DE 30 CARACTERES.

3. NENHUM NUMERO PODE SER MENOR QUE -1024 OU MAIOR QUE 1023.

5. RESTRIÇOES SEMANTICAS

1. CADA ROTULO PODE APARECER SOMENTE UMA VEZ COMO PRIMEIRO ELEMENTO DE UMA ASSERTIVA.

2. CADA ROTULO DEVE APARECER UMA VEZ COMO PRIMEIRO ELEMENTO DE UMA ASSERTIVA.

3. NAO E PERMITIDO O USO DE MAIS DE 15 ASSERTIVAS QUE CONTENHAM INSTRUÇOES COM PARAMETRO CONSECUTIVAS.

4. O NUMERO MAXIMO DE OPERANDOS OU $ DE UMA LISTA DE INDICAÇOES DE SUCESSORAS NÃO PODE EXCEDER O VALOR DA POTENCIA DE DOIS CORRESPONDENTE AO NUMERO DE BITS COM VALOR 1 DENTRE OS SEIS BITS MENOS SIGNIFICATIVOS DA MASCARA ASSOCIADA.

5. NÃO É PERMITIDO O USO DE MAIS DE UMA REFERENCIA A QUALQUER INSTRUÇÃO EM UMA LISTA DE INDICAÇOES DE SUCESSORAS.

6. CADA INSTRUÇÃO COM PARAMETRO ADMITE UM INTERVALO ESPECIFICO DE VALORES VALIDOS PARA ESTE PARAMETRO. NÃO É PERMITIDO O USO DE TAIS INSTRUÇOES COM PARAMETROS FORA DOS INTERVALOS A ELAS ASSOCIADOS.

ANEXO V. EXECUTIVOS DISPONIVEIS NO COMPUTADOR MITRA-15

1. EXECUTIVOS REFERENTES AS MICRO-INSTRUÇOES Ø

Ø	MNEMONICO	MODIFICAÇAO DE "T"	OPERAÇOES REALIZADAS
4	TT	DUPLA	? IND = 1 => MODIF. NORMAL; U <- D
5	TF	DUPLA	? IND = 0 => MODIF. NORMAL; U <- D
6	MCUD	IMPLICITA	U.L <- RT; U.H <- 0
7	MCUG	IMPLICITA	U.L <- 0; U.H <- RT
8	BRO	MULTIPLA	DESVIO SEGUNDO R(J)8-13; U <- D
9	BRG	MULTIPLA	DESVIO SEGUNDO R(J)10-15; U <- D
10	MURT	MULTIPLA	DESVIO SEGUNDO D10-15; R <- D; U.L <- D.L; U.H <- 0
11	R+2U	NORMAL	U <- R + 2
12	R+2R	NORMAL	R, U <- R + 2
13	R+UU	NORMAL	U <- R + U
14	R+UI	NORMAL	R, U <- R + U; I1 <- CY; I2 <- OV
15	R-UU	NORMAL	U <- R - U
16	MCR	IMPLICITA	R.L, U.L <- RT; R.H, U.H <- 0
17	R-IR	NORMAL	R, U <- R - 1
18	R.UI	NORMAL	R, U <- R AND U
19	RIUI	NORMAL	R, U <- R OR U
20	U/BR	NORMAL	R, U <- D/2; R15, U15 <- B; D<-D0
21	RXBR	NORMAL	R, U <- R+R; R0, U0 <- B; B <- CY
22	=UR	NORMAL	R, U <- D
23	=UI	NORMAL	R, U <- D; I1 <- SZ; I2 <- SN
24	=RU	NORMAL	U <- R
25	URR1	NORMAL	R.L, U.L <- R.L; R.H, U.H <- D.L
26	URR0	NORMAL	R.L, U.L <- R.L; R.H, U.H <- D.H
27	RUR0	NORMAL	R.L, U.L <- D.L; R.H, U.H <- R.H

2.1. EXECUTIVOS REFERENTES AS MICRO-INSTRUÇÕES DO TIPO ØR
(primeira parte)

Ø	R	MNEMONICO	MODIFICAÇÃO DE "T"	OPERAÇÕES REALIZADAS
0	0	=BBP	NORMAL	Bit de prot. de mem. ⟨- B
0	1	HLT	RECOPIA	U ⟨- B
0	2	LD	IMPLICITA	A, U ⟨- LP
0	3	ED	IMPLICITA	EP, U ⟨- D
0	4	AC	IMPLICITA	Aceitação de susp. s/ ret.
0	5	ACR	RET. PILHA	Aceitação de susp. c/ ret.
0	6	DMS	RECOPIA	Desvio por violação de modo
0	7	TI	NORMAL	Teste de interrupções; U⟨-D
1	0	P+2P	NORMAL	Teste de inter.; P, U ⟨- P+2
1	1	UXOU	NORMAL	U ⟨- D + D
1	2	=0B	NORMAL	B ⟨- 0; U ⟨- D
1	3	=1B	NORMAL	B ⟨- 1; U ⟨- D
1	4	=SB	NORMAL	B ⟨- D15; U ⟨- D
1	5	=PB	NORMAL	B ⟨- D0; U ⟨- D
1	6	=1D	NORMAL	S0 ⟨- 1
1	7	=UP	NORMAL	I1 ⟨- D1; I2 ⟨- D0; U ⟨- D

2.2. EXECUTIVOS REFERENTES AS MICRO-INSTRUÇOES DO TIPO ØR
(segunda parte)

Ø	R	MNEMONICO	MODIFICAÇAO DE "T"	OPERAÇOES REALIZADAS
2	0	=RJU	NORMAL	U ⟵ R(J); J ⟵ 0
2	1	=UE	NORMAL	MS ⟵ D2; PM ⟵ D4; U ⟵ D
2	2	=UA	NORMAL	MA ⟵ D3; U ⟵ D
2	3	=BP	NORMAL	I1 ⟵ SZ; I2 ⟵ B; U ⟵ D
2	4	=ZP	NORMAL	I2 ⟵ SZ; I2 ⟵ SN; U ⟵ D
2	5	=0U	NORMAL	U ⟵ 0
2	6	=1U	NORMAL	U ⟵ 1
2	7	=IU	NORMAL	U.L ⟵ Indicadores
3	0	=URJ	NORMAL	R(J), U ⟵ 0; J ⟵ 0
3	1	M+1U	NORMAL	U ⟵ M + 1
3	2	=MU	NORMAL	U ⟵ M
3	3	=UIP	NORMAL	Indicadores ⟵ U.L
3	4	MCJ	IMPLICITA	J ⟵ T
3	5	UOU0	NORMAL	U.L ⟵ 0
3	6	0UU0	NORMAL	U.H ⟵ 0
3	7	UUU3	NORMAL	U.L ⟵ D.H; U.H ⟵ D.L

3.1. EXECUTIVOS REFERENTES AS MICRO-INSTRUÇOES DO TIPO ØS
(primeira parte)

Ø	R	MNEMONICO	MODIFICAÇAO	OPERAÇOES REALIZADAS
28	0	LI	RESTRITA	A, U ⟨- LP; AD, C ⟨- R(J)
28	1	EI	RESTRITA	EP, U ⟨- 0; AD, C ⟨- R(J)
28	2	R+UB	RESTRITA	U ⟨- R + 0; B ⟨- C4
28	3	R+UR	RESTRITA	U, R ⟨- R + D
28	4	R-UR	RESTRITA	U, R ⟨- R - U
28	5	R-UI	RESTRITA	U, R ⟨- R-D; I1⟨-CY; I2⟨-OV
28	6	R+NR	RESTRITA	U, R ⟨- R + D + D
28	7	R-NR	RESTRITA	U, R ⟨- R - D - D
29	0	R@UR	RESTRITA	R, U ⟨- R XOR D
29	1	R@UI	RESTRITA	R,U⟨-R XOR D; I1⟨-SZ; I2⟨-SN
29	2	R*MU	RESTRITA	U ⟨- NOT (R) AND M
29	3	R.UU	RESTRITA	U ⟨- R AND U
29	4	=RZ	RESTRITA	SZ ⟨- (R=0); SN ⟨- (R⟨0)
29	5	#RR	RESTRITA	R, U ⟨- NOT (R)
29	6	=OR	RESTRITA	R, U ⟨- 0
29	7	RXOR	RESTRITA	R, U ⟨- R + R; B ⟨- CY

3.2. EXECUTIVOS REFERENTES AS MICRO-INSTRUÇÕES DO TIPO ØS
(segunda parte)

Ø	R	MNEMONICO	MODIFICAÇÃO	OPERAÇÕES REALIZADAS
30	0	RXSR	RESTRITA	R, U ⟨- R + R + R15; B ⟨- C
30	1	U/SR	RESTRITA	R,U⟨-D/2; R15,U15⟨-D15; B⟨-D0
30	2	U/0R	RESTRITA	R, U ⟨- D/2; B ⟨- D0
30	3	U/PR	RESTRITA	R,U⟨-D/2; R15,U15⟨-D0; B⟨-D0
30	4	RMU0	RESTRITA	U.L ⟨- M.L; U.H ⟨- R.H
30	5	MRU2	RESTRITA	U.L ⟨- R.H; U.H ⟨- M.H
30	6	RMU1	RESTRITA	U.L ⟨- M.L; U.H ⟨- R.L
30	7	MRU0	RESTRITA	U.L ⟨- R.L; U.G ⟨- M.G
31	0	RUR2	RESTRITA	R.L,U.L⟨-D.H; R.H,U.H⟨-R.H
31	1	0UR2	RESTRITA	R.L,U.L ⟨- D.H; R.H,U.H ⟨- 0
31	2	0UR0	RESTRITA	R.L,U.L ⟨- D.L; R.H,U.H ⟨- 0
31	3	RRR3	RESTRITA	R.L,U.L⟨-R.H; R.H,U.H⟨-R.L
31	4	=RJ	RESTRITA	J ⟨- R.L; U ⟨- D
31	5	R+1R	RESTRITA	R, U ⟨- R + 1
31	6	ACIR	RET.PILHA	Aceit. de susp. ind. c/ ret.
31	7	R@UU	RESTRITA	U ⟨- R XOR U

ANEXO VI. MICRO-PROGRAMA RESPONSAVEL PELA IMPLEMENTAÇAO DA MAQUINA BASICA

MICRO-ASSEMBLER MITRA-15

MAQUINA BASICA EDISON

ULTIMA MODIFICACAO REALIZADA: 27 DE NOVEMBRO DE 1984 AS 14:05 HORAS.

CONJUNTO DE INSTRUCOES IMPLEMENTADAS:

CODE	MNEM.	CODE	MNEM.	CODE	MNEM.	CODE	MNEM.
00 *	ADDRL	10	PUSHBL	20	PUSHWL	30	PUSHSL
01 *	ADDRG	11	PUSHBG	21	PUSHWG	31	PUSHSG
02 *	ADDRLI	12	PUSHBLI	22	PUSHWLI	32	PUSHSLI
03 *	ADDRGI	13	PUSHBGI	23	PUSHWGI	33	PUSHSGI
04 *	ADDRLX	14	PUSHBLX	24	PUSHWLX	34	PUSHSLX
05 *	ADDRGX	15	PUSHBGX	25	PUSHWGX	35	PUSHSGX
06 *	ADDRLIX	16	PUSHBLIX	26	PUSHWLIX	36	PUSHSLIX
07 *	ADDRGIX	17	PUSHBGIX	27	PUSHWGIX	37	PUSHSGIX
08 *	ADDRL	18	POPBL	28	POPWL	38	POPSL
09 *	ADDRG	19	POPBG	29	POPWG	39	POPSG
0A *	ADDRLI	1A	POPBLI	2A	POPWLI	3A	POPSLI
0B *	ADDRGI	1B	POPBGI	2B	POPWGI	3B	POPSGI
0C *	ADDRLX	1C	POPBLX	2C	POPWLX	3C	POPSLX
0D *	ADDRGX	1D	POPBGX	2D	POPWGX	3D	POPSGX
0E *	ADDRLIX	1E	POPBLIX	2E	POPWLIX	3E	POPSLIX
0F *	ADDRGIX	1F	POPBGIX	2F	POPWGIX	3F	POPSGIX
40	ADD	50	IS	60	ZERO	70	CRTFAT
41	SUB	51	ISNST	61	TRUE	71	CRTSON
42	MUL	52	CONSTS	62	CONSTB	72 *	HALT
43	DIV	53	BLANCS	63	CONSTW	73 *	HALT
44	MOD	54 *	PACK	64	CODEAD	74 *	WHEN
45	NOT	55 *	PACK	65	JP	75 *	WHEN
46	OR	56 *	INDEX	66	JPFALSE	76 *	WAIT
47	AND	57 *	INDEX	67	JPTRUE	77 *	WAIT
48	NEVER	58	EMPTY	68	ALOC	78 *	PLACE
49	LSS	59	INCLUD	69	CALL	79 *	PLACE
4A	EQL	5A	INCLUD1	6A	CALLI	7A *	OBTAIN
4B	LEQ	5B	IN	6B	PROCP	7B *	OBTAIN
4C	GTR	5C	UNION	6C	PROCPV	7C *	SENSE
4D	NEQ	5D	INTER	6D	RET	7D *	SENSE
4E	GEQ	5E	DIFF	6E	RETP	7E	LOAD
4F	EVER	5F	DELSET	6F	RETPV	7F	TRAP

MICRO-ASSEMBLER NITRA-15

```
51  %
52  %       REGISTRADORES DA MAQUINA VIRTUAL:
53  %
54
55  PC:             EQU     ( 0 );          %  P E' UTILIZADO COMO PC.
56  BTBASE:         EQU     ( 1 );          %  L E' UTILIZADO COMO BTBASE.
57  MONITORBASE:    EQU     ( 2 );          %  G APONTA A AREA DO MONITOR.
58  ACC:            EQU     ( 3 );          %  A E' UTILIZADO COMO ACC.
59  ACC1:           EQU     ( 4 );          %  E E' UTILIZADO COMO ACC1.
60  SP:             EQU     ( 5 );          %  X E' UTILIZADO COMO SP.
61  IR:             EQU     ( 6 );          %  V E' UTILIZADO COMO IR.
62  SZ:             EQU     ( 7 );          %  W E' UTILIZADO COMO SZ.
63
64  ADDR1:          EQU     ( ACC1 );       %  E
65  ADDR2:          EQU     (  IR  );       %  V
66
67
68  %
69  %       INDICADORES DA MAQUINA:
70  %
71
72  EXT:            EQU     ( 0 );          %  EX: MULT & DIV POR "HARDWARE"
73  NEG:            EQU     ( 1 );          %  SN: SINAL NEGATIVO.
74  ZERO:           EQU     ( 2 );          %  SZ: VALOR ZERO.
75  IMPAR:          EQU     ( 3 );          %  SO: PARIDADE DO ENDERECO.
76  MESTRE:         EQU     ( 4 );          %  MS:MODO MESTRE.
77  OVERFLOW:       EQU     ( 5 );          %  I2: OVERFLOW.
78  CY:             EQU     ( 6 );          %  I1: CY.
79  CORINGA:        EQU     ( 7 );          %  B:  USO GERAL.
80
81
82  %
83  %       CONSTANTES:
84  %
85
86  BRANCO:         EQU     ( #H 40 );      %  ESPACO EM BRANCO.
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

MICRO-ASSEMBLER MITRA-15

```
101  %
102  %       BUSCA DA PROXIMA INSTRUCAO E INICIO DA DECODIFICACAO.
103  %
104
105  INICIO:        LEIA &          P+2P;                    % FETCH.
106                 REESCREVA &     MURT    ( IR )
107                                 MASCARA ( #B 010 000 )  PILHA, DEMAIS;
108  %
109  %              PILHA:          INSTRUCOES DOS TIPOS:
110  %                                     ADDR, PUSH E POP.
111  %
112  %              DEMAIS:         TODAS AS DEMAIS INSTRUCOES.
113  %
114
115  PILHA:         LEIA &          R+2R    ( PC );          % LE DSP.
116                 REESCREVA &     =UR     ( ADDRi ),       LOCALGLOBAL;
117
118  DEMAIS:                        MCUD    ( 2 );
119                                 BRD     ( IR )
120                                 MASCARA ( #B 111 000 )
121                                 ARILOG, COMPAR, ARROP,   SETOP,
122                                 CONSJP, PROCED, PROCES,  IOINT;
123
124  ARILOG:        LEIA &          R-UR    ( SP ),          ARILOGOP;
125
126  COMPAR:        LEIA &          R-UR    ( SP ),          COMPAROP;
127
128  ARROP:         LEIA &          R+2R    ( PC ),          ARROPOP;
129
130  SETOP:                         BRD     ( IR )
131                                 MASCARA ( #B 000 100 )  CONSTROI, OPERA;
132
133  CONSJP:                        BRD     ( IR )
134                                 MASCARA ( #B 000 100 )  CONST, CODEADJP;
135
136  PROCED:        LEIA &          R+2R    ( PC );
137                                 BRD     ( IR )
138                                 MASCARA ( #B 000 111 )
139                                 ALOC,   CALL,   CALLI,   PROCP,
140                                 PROCPV, RET,    RETP,    RETPV;
141
142  PROCES:                        MCUD    ( ERRIR );
143                 LEIA &          R+UU    ( MONITORBASE );
144                 ESCREVA &       =OU,    NUCLEO;
145
146  IOINT:                         BRD     ( IR )
147                                 MASCARA ( #B 000 110 )
148                                 PLACE,  OBTAIN, SENSE, INITRAP;
149
150
```

MICRO-ASSEMBLER MITRA-15

```
151  LOCALGLOBAL:                BRD     ( IR )
152                              MASCARA ( #B 000 001 )  LOCAL, GLOBAL;
153
154  LOCAL:                      MCUD    ( 2 );          % LV = -1.
155                  LEIA &      R-UU    ( BTBASE ),     ADDBASE;
156
157  GLOBAL:         LEIA &      R+2R    ( PC );         % LE LV.
158                  REESCREVA & UXOU;
159                  LEIA &      R+UU    ( BTBASE );     % LE BT ( LV ).
160
161  ADDBASE:                    BRD     ( IR )
162                              MASCARA ( #B 000 010 )  DIRETASJA,MALUF;
163
164  DIRETASJA:      REESCREVA & R+UU    ( ADDR1 ),      DIRINDIR;
165
166  MALUF:          REESCREVA & R+UU    ( ADDR1 );
167                  LEIA &      =RU     ( ADDR1 );
168                  REESCREVA & =UR     ( ADDR1 ),      DIRINDIR;
169
170  DIRINDIR:                   BRD     ( IR )
171                              MASCARA ( #B 110 000 )  S0, S1, S2, SN;
172
173  S0:                         MCUD    ( 02 );
174                              BRD     ( IR )
175                              MASCARA ( #B 000 100 )  ADDRNOTX, ADDRX;
176
177  ADDRX:          LEIA &      R-UR    ( SP );
178                              =0B;
179                  REESCREVA & R+UR    ( ADDR1 );
180
181  ADDRNOTX:       LEIA &      =RU     ( SP );
182                  ESCREVA &   =OU;
183                  LEIA &      R+2R    ( SP );
184                              R+2R    ( SP );
185                  ESCREVA &   =RU     ( ADDR1 ),      INICIO;
186
187  S1:                         =1U,                    PUSHPOP;
188
189  S2:                         BRD     ( IR )
190                              MASCARA ( #B 001 100 )
191                              PUSHW, PUSHWX, POPW, POPWX;
192
193  PUSHWX:                     R-1R    ( SP );
194                  LEIA &      R-1R    ( SP );
195                  REESCREVA & R+UR    ( ADDR1 );
196
197  PUSHW:          LEIA &      =RU     ( ADDR1 );
198                  REESCREVA & =UR     ( ACC );
199                  LEIA &      =RU     ( SP );
200                              R+2R    ( SP );
```

MICRO-ASSEMBLER MITRA-15

```
201                 ESCREVA &       =RU     ( ACC ),        INICIO;
202
203 POPWX:                          R-1R    ( SP );
204                 LEIA &          R-1R    ( SP );
205                 REESCREVA &     =UR     ( ACC );
206                                 R-1R    ( SP );
207                 LEIA &          R-1R    ( SP );
208                 REESCREVA &     R+UR    ( ADDR1 ),      SALVAPALAVRA;
209
210 POPW:                           R-1R    ( SP );
211                 LEIA &          R-1R    ( SP );
212                 REESCREVA &     =UR     ( ACC ),        SALVAPALAVRA;
213
214 SALVAPALAVRA:  LEIA &          =RU     ( ADDR1 );
215                 ESCREVA &       =RU     ( ACC ),        INICIO;
216
217 SN:            LEIA &          R+2R    ( PC );
218                 REESCREVA &     =UR     ( SZ ),         PUSHPOP1;
219 PUSHPOP:                       =UR     ( SZ ),         PUSHPOP1;
220 PUSHPOP1:                      BRD     ( IR )
221                                 MASCARA ( #B 001 100 )
222                                 PUSH, PUSHX, POP, POPX;
223
224 PUSH:                          R-1R    ( SZ );
225                                 TT      ( NEG ),        INICIO;
226                 LEIA &          =RU     ( ADDR1 );
227                                 TF      ( IMPAR ),      PUSHSRCPAR;
228 PUSHSRCIMPAR:   REESCREVA &     OUR2    ( ACC );        %  ACC.L = M.H
229                                 R-1R    ( SZ );
230                                 TT      ( NEG ),        PUSHSZIMPAR;
231                 LEIA &          R+2R    ( ADDR1 );      %  ACC.H = M.L
232                 REESCREVA &     URR1    ( ACC ),        SALVAPROXPAL;
233 PUSHSRCPAR:     REESCREVA &     =UR     ( ACC );
234                                 R-1R    ( SZ );
235                                 TT      ( NEG ),        PUSHSZIMPAR;
236                                 R+2R    ( ADDR1 ),      SALVAPROXPAL;
237 SALVAPROXPAL:   LEIA &          =RU     ( SP );
238                                 R+2R    ( SP );
239                 ESCREVA &       =RU     ( ACC ),        PUSH;
240 PUSHSZIMPAR:    LEIA &          =RU     ( SP );
241                                 R+2R    ( SP );
242                 ESCREVA &       OUR0    ( ACC ),        INICIO;
243
244 PUSHX:                         R-1R    ( SP );
245                 LEIA &          R-1R    ( SP );
246                 REESCREVA &     R+UR    ( ADDR1 ),      PUSH;
247
248 POP:                           =1U;
249                                 R.UU    ( SZ );
250                                 R+UU    ( SZ );
```

MICRO-ASSEMBLER MITRA-15

```
251                                 R-UR    ( SP );         % SP = SP -SZ*
252  POPSTR:                        =UR     ( ADDR2 );      % ADDR2 = SP
253                                 R-1R    ( SZ );
254                                 TT      ( NEG ),        INICIO;
255                  LEIA &         =RU     ( ADDR2 ),      POPBYTE;
256
257  TESTAFIMPOP:                   TT      ( NEG ),        INICIO;
258                                 R+1R    ( ADDR1 );
259                  LEIA &         R+1R    ( ADDR2 );
260  POPBYTE:                       TT      ( IMPAR ),      POPSRCIMPAR;
261  POPSRCPAR:      REESCREVA &    OUR0    ( ACC ),        LEIAPALAVRA;
262  POPSRCIMPAR:    REESCREVA &    OUR2    ( ACC ),        LEIAPALAVRA;
263  LEIAPALAVRA:    LEIA &         =UR     ( ADDR1 );
264                                 TT      ( IMPAR ),      POPDSTIMPAR;
265  POPDSTPAR:      ESCREVA &      MRU0    ( ACC ),        INCENDERECO;
266  POPDSTIMPAR:    ESCREVA &      MRU1    ( ACC ),        INCENDERECO;
267  INCENDERECO:                   R-1R    ( SZ ),         TESTAFIMPOP;
268
269  POPX:                          =1U;
270                                 R.UU    ( SZ );
271                                 R+UU    ( SZ );
272                                 R-UR    ( SP );         % SP = SP - SZ*
273                                 R-1R    ( SP );
274                  LEIA &         R-1R    ( SP );
275                  REESCREVA &    R+UR    ( ADDR1 );
276                                 R+2U    ( SP ),         POPSTR;
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
```

MICRO-ASSEMBLER MITRA-15

```
301 ARILOGOP:       REESCREVA &     =UR     ( ACC1 );
302                                 MCUD    ( 2 );
303                 LEIA &          R-UU    ( SP );         % LE O 1@ OPER.
304                 REESCREVA &     =UR     ( ACC );
305                 LEIA &          =RU     ( SP );         % PREP. O PUSH
306                                 =RU     ( ACC1 );       % U = 2@ OPER.
307                                 BRD     ( IR )
308                                 MASCARA ( #B 000 111 )
309                                 ADD, SUB, MUL, DIV, MOD, NOT, OR, AND;
310
311 ADD:            ESCREVA &       R+UI    ( ACC ),        INICIO;
312
313 SUB:            ESCREVA &       R-UI    ( ACC ),        INICIO;
314
315 FATOR2POS:                      U/OR    ( IR ),         PROXBITMUL;
316 NAOSOMA:                        U/SR    ( IR ),         PROXBITMUL;
317 TESTASINALFAT2:                 TF      ( NEG ),        FATOR2POS;
318 FATOR2NEG:                      U/BR    ( IR ),         PROXBITMUL;
319
320 MUL:                           =OR     ( IR );
321                                 MCUD    ( 16 );
322                                 =UR     ( SZ );
323 PROXBITMUL:                     =RU     ( ACC );
324                                 U/BR    ( ACC );
325                                 R-1R    ( SZ );
326                                 TT      ( ZERO ),       ENDMUL1;
327                                 =RU     ( IR );
328                                 TF      ( CORINGA ),    NAOSOMA;
329                                 R+UU    ( ACC1 );
330                                 =RZ     ( ACC1 ),       TESTASINALFAT2;
331 ENDMUL1:                        TT      ( CORINGA ),    ULTBITSET;
332                                 =RU     ( IR ),         ENDMUL2;
333 ULTBITSET:                      =RU     ( ACC1 );
334                                 TF      ( NEG ),        FATORPOS;
335                                 R-UU    ( IR );
336                                 U/OR    ( IR ),         ENDMUL3;
337 FATORPOS:                       R-UU    ( IR );
338 ENDMUL2:                        U/BR    ( IR );
339 ENDMUL3:                        =RU     ( ACC );
340                 ESCREVA &       U/BR    ( ACC ),        INICIO;
341
342 DIV:                           MCUD    ( #B 101 );
343                                 =UR     ( IR ),         DIVMOD;
344
345 MOD:                           =OR     ( IR );
346 DIVMOD:                        =RZ     ( ACC1 );
347                                 TF      ( NEG ),        TESTADIVIDENDO;
348                                 R-1R    ( ACC1 );
349                                 #RR     ( ACC1 );
350                                 R+1R    ( IR );
```

MICRO-ASSEMBLER MITRA-15

```
351  TESTADIVIDENDO:                 =RZ     ( ACC );
352                                  TF      ( NEG ),        DIVISAO;
353                                  R-1R    ( ACC );
354                                  #RR     ( ACC );
355                                  MCUD    ( #B 010 );
356                                  R@UR    ( IR );
357  DIVISAO:                        =RU     ( IR );
358                                  U/OR    ( IR );
359                                  =UP;    XXXXXXXX        I1=DIV, I2=SINAL
360                                  =OR     ( IR );
361                                  MCUD    ( 16 );
362                                  =UR     ( SZ );
363  PROXBITDIVMOD:                  =RU     ( ACC1 );
364                                  R-UU    ( IR );
365                                  TT      ( NEG ),        NAODEU;
366                                  =UR     ( IR );
367                                  RXOR    ( ACC ),        SHIFTIR;
368  NAODEU:                         =1B;
369                                  RXBR    ( ACC ),        SHIFTIR;
370  SHIFTIR:                        RXBR    ( IR );
371                                  R-1R    ( SZ );
372                                  TF      ( ZERO ),       PROXBITDIVMOD;
373                                  TF      ( CY ),         ENDMOD;
374  ENDDIV:                         #RR     ( ACC ),        ENDDIVMOD;
375  ENDMOD:                         =RU     ( IR );
376                                  =UR     ( ACC ),        ENDDIVMOD;
377  ENDDIVMOD:                      TT      ( OVERFLOW ),   RESULTNEG;
378  RESULTPOS:      ESCREVA &       =RU     ( ACC ),        INICIO;
379  RESULTNEG:                      R-UR    ( ACC );
380                  ESCREVA &       #RR     ( ACC ),        INICIO;
381
382  NOT:            REESCREVA &     =UR     ( ACC );
383                  LEIA &          =RU     ( SP );
384                                  R+2R    ( SP );
385                  ESCREVA &       #RR     ( ACC1 ),       INICIO;
386
387  OR:             ESCREVA &       RIUI    ( ACC ),        INICIO;
388
389  AND:            ESCREVA &       R.UI    ( ACC ),        INICIO;
390
391
392
393
394
395
396
397
398
399
400
```

MICRO-ASSEMBLER MITRA-15

```
401  COMPAROP:      REESCREVA &     =UR     ( ACC );
402                                 MCUD    ( 2 );
403                 LEIA &          R-UR    ( SP );
404                 REESCREVA &     =UR     ( ACC1 );
405                 LEIA &          =RU     ( SP );
406                                 R+2R    ( SP );
407                                 =RU     ( ACC );
408                                 R-UR    ( ACC1 );
409                                 =RZ     ( ACC1 );
410                                 TT      ( ZERO ),       VALIGUAL;
411                                 TT      ( NEG  ),       VALMAIOR;
412  VALMENOR:                      RXBR    ( IR );
413  VALIGUAL:                      RXBR    ( IR );
414  VALMAIOR:                      BRD     ( IR )
415                                 MASCARA ( #B 000 100 )  FALSO, VERD;
416
417  FALSO:         ESCREVA &       =OR     ( ACC ),        INICIO;
418
419  VERD:                          =OR     ( ACC );
420                 ESCREVA &       #RR     ( ACC ),        INICIO;
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
```

MICRO-ASSEMBLER MITRA-15

```
451 ARROPOP:      REESCREVA &     =UR     ( SZ );
452                               BRD     ( IR )
453                               MASCARA ( #B 000 110 )
454                               COMPSTR, CONSTSTR, PACKSTR, INDEX;
455
456 CONSTSTR:                     BRD     ( IR )
457                               MASCARA ( #B 000 001 )  CONSTS, BLANCS;
458
459 COMPSTR:                      =RU     ( SP );
460                               =UR     ( ADDR1 );      % ADDR1 = SP
461                               =RU     ( SZ );
462                               R-UR    ( SP );         % SP = SP - SZ
463                               =RU     ( SZ );
464                               R+1R    ( SZ );
465                               =0B;
466                               U/BR    ( SZ );         % SZ = SZ* / 2
467                               %   SE SZ PAR B = 1.
468
469 PROXCOMP:                     R-1R    ( SZ );
470                               TT      ( NEG ),        CADEIASIGUAIS;
471                               R-1R    ( ADDR1 );
472               LEIA &          R-1R    ( ADDR1 );
473               REESCREVA &     =UR     ( ACC );
474                               R-1R    ( SP );
475               LEIA &          R-1R    ( SP );
476               REESCREVA &     R-UR    ( ACC );
477                               TT      ( CORINGA ),    TESTEZERO;
478                               OURO    ( ACC );
479                               =1B;
480 TESTEZERO:                   =RZ     ( ACC );
481                               TT      ( ZERO ),       PROXCOMP;
482                               =RU     ( SZ );
483                               R+UU    ( SZ );
484                               R-UR    ( SP );
485                               R-UR    ( SP );
486                               =0R     ( ACC ),        ISISNT;
487 CADEIASIGUAIS:                R-1R    ( ACC ),        ISISNT;
488 ISISNT:                       BRD     ( IR )
489                               MASCARA ( #B 000 001 )  IS, ISNT;
490
491 ISNT:                        #RR     ( ACC );
492
493 IS:               LEIA &      =RU     ( SP );
494                               R+2R    ( SP );
495               ESCREVA &       =RU     ( ACC ),        INICIO;
496
497
498
499 CONSTS:                       R-1R    ( SZ );
500                               TT      ( NEG ),        INICIO;
```

MICRO-ASSEMBLER MITRA-15

```
501                 LEIA &          R+2R    ( PC );
502                 REESCREVA &     =UR     ( ACC );
503                 LEIA &          =RU     ( SP );
504                                 R+2R    ( SP );
505                                 R-1R    ( SZ );
506                                 TT      ( NEG ),        POEULTIMO;
507                 ESCREVA &       =RU     ( ACC ),        CONSTS;
508 POEULTIMO:                      =0U;
509                 ESCREVA &       URR0    ( ACC ),        INICIO;
510
511 BLANCS:                         R-1R    ( SZ );
512                                 TT      ( NEG ),        INICIO;
513                 LEIA &          =RU     ( SP );
514                                 R+2R    ( SP );
515                                 R-1R    ( SZ );
516                                 TT      ( NEG ),        ULTIMOBRANCO;
517                                 MCUD    ( BRANCO );
518                                 =UR     ( ACC );
519                 ESCREVA &       URR1    ( ACC ),        BLANCS;
520 ULTIMOBRANCO:   ESCREVA &       MCUD    ( BRANCO );
521                                 =0B,                    INICIO;
522
523 PACKSTR:                        R+UU    ( SZ );         % U = SZ + SZ
524                                 R-UR    ( SP );         % SP = SP - U
525                                 =UR     ( ADDR1 );      % ADDR1 = SP
526 PROXIMOPACK:                    R-1R    ( SZ );
527                                 TT      ( NEG ),        INICIO;
528                 LEIA &          =RU     ( ADDR1 );
529                 REESCREVA &     0URO    ( ACC );
530                                 R-1R    ( SZ );
531                                 TT      ( NEG ),        ULTIMOPACK;
532                 LEIA &          R+2R    ( ADDR1 );
533                                 R+2R    ( ADDR1 );
534                 REESCREVA &     URR1    ( ACC );
535 ULTIMOPACK:     LEIA &          =RU     ( SP );
536                                 R+2R    ( SP );
537                 ESCREVA &       =RU     ( ACC ),        PROXIMOPACK;
538
539 INDEX:          LEIA &          R+2R    ( PC );         % LE FAT.
540                                 R+2R    ( PC );         % DESPREZA SZ.
541                 REESCREVA &     =UR     ( ACC );
542                                 R-1R    ( SP );
543                 LEIA &          R-1R    ( SP );         % LE O INDICE.
544                 REESCREVA &     =UR     ( ACC1 );
545                                 =RU     ( SZ );
546                                 R-UR    ( ACC1 );
547                 LEIA &          =RU     ( SP );         % PREP O PUSH
548                                 R+2R    ( SP ),         MUL;
549
550
```

MICRO-ASSEMBLER MITRA-15

```
551 CONSTROI:                   =0B;
552                             BRD     ( IR )
553                             MASCARA ( #B 000 011 )
554                             EMPTY,  INCLUD, INCLUD1, IN;
555
556 EMPTY:                      MCUD    ( 16 );
557                             =UR     ( SZ );
558 PROXPALZERO:                R-1R    ( SZ );
559                             TT      ( NEG ),        INICIO;
560             LEIA &          =RU     ( SP );
561                             R+2R    ( SP );
562             ESCREVA &       =0U,                    PROXPALZERO;
563
564 INCLUD:     LEIA &          R+2R    ( PC );
565             REESCREVA &     =UR     ( SZ ),         PROXIMOELEM;
566
567 INCLUD1:                    =0R     ( SZ ),         INCLUAUM;
568
569 PROXIMOELEM:                R-1R    ( SZ );
570                             TT      ( NEG ),        INICIO;
571 INCLUAUM:                   R-1R    ( SP );
572             LEIA &          R-1R    ( SP );         % LE O ELEMENTO
573             REESCREVA &     OUR0    ( ACC1 );       %    A INCLUIR.
574                             =1U;
575                             =UR     ( ACC );        % ACC = 1
576                             MCUD    ( #B 1111 );
577                             R.UU    ( ACC1 );
578                             =UR     ( IR );         % IR = ACC1.0F
579 DESLOCAESQ:                 R-1R    ( IR );
580                             TT      ( NEG ),        MASCAREEINCLUA;
581                             RXOR    ( ACC ),        DESLOCAESQ;
582 MASCAREEINCLUA:             =RU     ( ACC1 );
583                             U/0R    ( ACC1 );
584                             U/0R    ( ACC1 );
585                             U/0R    ( ACC1 );       % ACC1 = ACC1/8
586                             MCUD    ( 32 );
587                             R+UU    ( SZ );
588                             R+UU    ( SZ );         % U = 32 + 2*SZ
589                             R-UU    ( SP );         % U = SP - U
590             LEIA &          R+UR    ( ACC1 );       % ACC1 = ACC1+U
591             REESCREVA &     RIUI    ( ACC );
592             LEIA &          =RU     ( ACC1 );
593             ESCREVA &       =RU     ( ACC ),        PROXIMOELEM;
594
595 IN:                         MCUD    ( 34 );
596             LEIA &          R-UR    ( SP );
597                             R+2R    ( SP );
598             REESCREVA &     OUR0    ( ACC1 );
599                             =1U;
600                             =UR     ( ACC );
```

MICRO-ASSEMBLER MITRA-15

```
601                                MCUD    ( #B 1111 );
602                                R.UU    ( ACC1 );
603                                =UR     ( IR );
604  DESLOQUEESQIN:                R-1R    ( IR );
605                                TT      ( NEG ),        MASCAREIN;
606                                RXOR    ( ACC ),        DESLOQUEESQIN;
607  MASCAREIN:                    =RU     ( ACC1 );
608                                U/OR    ( ACC1 );
609                                U/OR    ( ACC1 );
610                                U/OR    ( ACC1 );
611                LEIA &          R+UU    ( SP );
612                REESCREVA &     R.UI    ( ACC );
613                                TT      ( ZERO ),       NOTIN;
614                                =OR     ( ACC );
615                                #RR     ( ACC );
616  NOTIN:                        MCUD    ( 2 );
617                LEIA &          R-UU    ( SP );
618                ESCREVA &       =RU     ( ACC ),        INICIO;
619
620  OPERA:                        MCUD    ( 16 );
621                                =UR     ( SZ ),         LECONJUNTO;
622
623  PROXOPERA:                    MCUD    ( 32 );
624                LEIA &          R-UR    ( ADDR1 );
625                                BRO     ( IR )
626                                MASCARA ( #B 000 011 )
627                                UNION, INTER, DIFF, DELSET;
628
629  UNION:        REESCREVA &     RIUI    ( ACC ),        FIMOPERA;
630
631  INTER:        REESCREVA &     R.UI    ( ACC ),        FIMOPERA;
632
633  DIFF:                         #RR     ( ACC );
634                REESCREVA &     R.UI    ( ACC ),        FIMOPERA;
635
636  DELSET:       REESCREVA &     =UR     ( ACC ),        TESTEFIMOPERA;
637
638  FIMOPERA:     LEIA &          =RU     ( ADDR1 );
639                ESCREVA &       =RU     ( ACC );
640  TESTEFIMOPERA:                R-1R    ( SZ );
641                                TT      ( ZERO ),       INICIO;
642  LECONJUNTO:                   R-1R    ( SP );
643                LEIA &          R-1R    ( SP );
644                                =UR     ( ADDR1 );
645                REESCREVA &     =UR     ( ACC ),        PROXOPERA;
646
647
648
649
650
```

MICRO-ASSEMBLER MITRA-15

```
651  CONST:                          =0U;
652                                  BRD     ( IR )
653                                  MASCARA ( #B 000 011 )
654                                  ZEROINST, TRUEINST, CONSTB, CONSTW;
655
656  ZEROINST:                       =UR     ( ACC ),        PUSHCONST;
657
658  TRUEINST:                       =UR     ( ACC );
659                                  R-1R    ( ACC ),        PUSHCONST;
660
661  CONSTB:                         =RU     ( IR );
662                                  0URO    ( ACC ),        PUSHCONST;
663
664  CONSTW:         LEIA &          R+2R    ( PC );
665                  REESCREVA &     =UR     ( ACC ),        PUSHCONST;
666
667  PUSHCONST:      LEIA &          =RU     ( SP );
668                                  R+2R    ( SP );
669                  ESCREVA &       =RU     ( ACC ),        INICIO;
670
671
672  CODEADJP:       LEIA &          R+2R    ( PC );
673                  REESCREVA &     =UR     ( ACC );        %  LE DSP
674                                  MCUD    ( 4 );
675                                  R-UR    ( ACC );
676                                  BRD     ( IR )
677                                  MASCARA ( #B 000 011 )
678                                  CODEAD, JP, JPFALSE, JPTRUE;
679
680  CODEAD:         LEIA &          =RU     ( SP );
681                                  R+2R    ( SP );
682                                  =RU     ( ACC );
683                  ESCREVA &       R+UU    ( ACC );
684                  LEIA &          =RU     ( SP );
685                                  R+2R    ( SP );
686                  ESCREVA &       =0U,                    INICIO;
687
688  JP:                             R+UR    ( PC ),         INICIO;
689
690  JPFALSE:                        R-1R    ( SP ),         JPC;
691
692  JPTRUE:                         R-1R    ( SP ),         JPC;
693
694  JPC:            LEIA &          R-1R    ( SP );
695                                  RRR3    ( IR );
696                  REESCREVA &     R+UR    ( IR );
697                                  U/OR    ( IR );
698                                  TT      ( CORINGA ),    INICIO;
699                                  =RU     ( ACC ),        JP;
700
```

MICRO-ASSEMBLER MITRA-15

```
701 ALOC:       REESCREVA &   R+UR   ( SP ),        INICIO;
702
703 CALL:       REESCREVA &   =UR    ( ADDR1 );     % ADDR1 = DSP
704                           MCUD   ( 4 );
705                           R-UR   ( ADDR1 );     % ADDR1 = DSP-4
706                           =RU    ( PC );
707                           R+UR   ( ADDR1 ),     ENDCALL;
708
709 CALLI:      REESCREVA &   =UR    ( ADDR1 );     % ADDR1 = DSP
710               LEIA &      R+2R   ( PC );        % LE LV.
711               REESCREVA & UXOU;                 % U = LV+LV
712               LEIA &      R+UU   ( BTBASE );    % LE BT ( LV )
713               REESCREVA & R+UR   ( ADDR1 );     % SOMA BASE.
714
715 ENDCALL:    LEIA &        =RU    ( SP );
716                           R+2R   ( SP );
717               ESCREVA &   =RU    ( PC );        % PUSH PC (1)
718               LEIA &      =RU    ( SP );
719                           R+2R   ( SP );
720               ESCREVA &   =0U;                  % PUSH 0  (2)
721                           =RU    ( ADDR1 );
722                           =UR    ( PC ),        INICIO;
723
724 PROCP:                    =0R    ( ADDR1 ),     PROCPPV;
725
726 PROCPV:     REESCREVA &   =UR    ( ADDR1 );     % ADDR1 = DSP
727               LEIA &      R+2R   ( PC );        % LE LV.
728
729 PROCPPV:    REESCREVA &   =UR    ( IR );        % IR = LV.
730                           MCUD   ( 2 );
731               LEIA &      R-UU   ( BTBASE );    % LE BASE.
732               REESCREVA & =UR    ( ACC );       % ACC = BASE.
733               LEIA &      =RU    ( SP );
734                           R+2R   ( SP );
735               ESCREVA &   =RU    ( ACC );       % PUSH BASE (3)
736               LEIA &      =RU    ( SP );
737                           R+2R   ( SP );
738               ESCREVA &   =0U;                  % PUSH 0   (4)
739                           RXOR   ( IR );        % IR = 2 * LV
740               LEIA &      R+UU   ( BTBASE );    % LE BT ( LV )
741               REESCREVA & =UR    ( ACC );       % ACC = BT(LV)
742               LEIA &      =RU    ( SP );
743                           R+2R   ( SP );        % PUSH
744               ESCREVA &   =RU    ( ACC );       % BT(LV)   (5)
745               LEIA &      =RU    ( SP );
746                           R+2R   ( SP );
747               ESCREVA &   =0U;                  % PUSH 0   (6)
748               LEIA &      =RU    ( SP );
749                           R+2R   ( SP );
750               ESCREVA &   =RU    ( IR );        % PUSH 2*LV (7)
```

MICRO-ASSEMBLER MITRA-15

```
751                             R+2R    ( SP );         % PUSH LIXO (8)
752                             R+2R    ( SP );         % PUSH LIXO (9)
753             LEIA &          R+2R    ( PC );         % LE SZ.
754                             MCUD    ( 18 );
755                             =UR     ( ACC );        % ACC=18
756             REESCREVA &     R+UR    ( ACC );        % ACC=SZ+18
757                             MCUD    ( 2 );
758             LEIA &          R-UU    ( BTBASE );
759                             =RU     ( ACC );        % U=SZ+18
760             ESCREVA &       R-UU    ( SP );         % BASE=SP-SZ-18
761                             =RU     ( IR );         % U=2*LV
762             LEIA &          R+UU    ( BTBASE );
763                             =RU     ( ACC );
764             ESCREVA &       R-UU    ( SP );
765                             =RU     ( ADDR1 );
766                             R+UR    ( SP ),         INICIO;
767
768 RET:        REESCREVA &     =0B;
769                             R-1R    ( PC );
770                             R-1R    ( PC ),         RETPPV;
771
772 RETP:       REESCREVA &     =UR     ( SZ );         % SZ = SZ.
773                             MCUD    ( 6 );
774             LEIA &          R-UR    ( SP );         % POP
775             REESCREVA &     R+UU    ( BTBASE );     % 2*LV  (9,8,7)
776                             =UR     ( ADDR1 );      % BTBASE+2*LV
777                             MCUD    ( 4 );
778             LEIA &          R-UR    ( SP );         % POP
779             REESCREVA &     =UR     ( ACC );        % BT(LV)  (6,5)
780             LEIA &          =RU     ( ADDR1 );
781             ESCREVA &       =RU     ( ACC );
782                             MCUD    ( 4 );
783             LEIA &          R-UR    ( SP );         % POP
784             REESCREVA &     =UR     ( ACC );        % BASE    (4,3)
785                             MCUD    ( 2 );
786             LEIA &          R-UU    ( BTBASE );
787                             =RU     ( SZ );
788                             R-UU    ( SP );         % SP = SP - SZ
789             ESCREVA &       =RU     ( ACC );
790
791 RETPPV:                     MCUD    ( 4 );
792             LEIA &          R-UR    ( SP );         % POP PC  (2,1)
793             REESCREVA &     =UR     ( PC ),         INICIO;
794
795 RETPV:      REESCREVA &     R-UR    ( SP );         % SP = SP-DSP
796             LEIA &          R+2R    ( PC ),         RETP;
797
798
799
800
```

MICRO-ASSEMBLER MITRA-15

```
% 
%                NUCLEO
%                ======

%
%       CONTEXTO DOS PROCESSOS ( RELATIVO A BTBASE ):
%

BASEPOS:                        EQU     ( 02 );
STATUSPOS:                      EQU     ( 04 );
AVALPOS:                        EQU     ( 06 );
EVALPOS:                        EQU     ( 08 );
XVALPOS:                        EQU     ( 10 );
PRIOPOS:                        EQU     ( 12 );
SPPOS:                          EQU     ( 14 );
PCPOS:                          EQU     ( 16 );
NUMFILHOS:                      EQU     ( 18 );
BTBASEPAI:                      EQU     ( 20 );
ANTERIOR:                       EQU     ( 22 );
SEGUINTE:                       EQU     ( 24 );

CONTEXTSIZE:                    EQU     ( 24 );

%
%       VARIAVEIS DO NUCLEO ( RELATIVAS A MONITORBASE ):
%

BTSIZE:                         EQU     ( 72 );
FILAPRONTOS:                    EQU     ( 74 );
FILACONDICAO:                   EQU     ( 76 );
MONITBT:                        EQU     ( 78 );
MONITSP:                        EQU     ( 80 );
MONITPC:                        EQU     ( 82 );
STDCNTXL:                       EQU     ( 84 );
STDCNTXP:                       EQU     ( 86 );
STDSWMASC:                      EQU     ( 88 );
STDSWADDR:                      EQU     ( 90 );
ERRIR:                          EQU     ( 92 );
ERRSP:                          EQU     ( 94 );
ERRPC:                          EQU     ( 96 );
ERRBTBASE:                      EQU     ( 98 );


CRTOPCODE:                      EQU     ( #H 71 );
HLTOPCODE:                      EQU     ( #H 72 );


ENDFILA:                        EQU     ( SZ );
PRIMEIRO:                       EQU     ( SP );
PRIORIDADE:                     EQU     ( PC );
```

MICRO-ASSEMBLER MITRA-15

```
851 %
852 %       TERMINACAO DAS ROTINAS DE INSERCAO E RETIRADA
853 %      DE PROCESSOS DAS FILAS DE PRONTOS E DE CONDICAO.
854 %
855
856 FIMPOEFILA:                 BRD     ( IR )
857                             MASCARA ( #B 000 110 )
858                             FPOECRT, FPOEHLT, FPOEWHN, FPOEWAIT;
859
860 FPOECRT:       LEIA &       R+2R    ( PC );
861                REESCREVA &  RUR2    ( ACC );
862                             MCUD    ( CRTOPCODE );
863                             R-UR    ( ACC );
864                             TT      ( ZERO ),       CONSTROIPROC;
865 FIMCRT:                     MCUD    ( MONITBT );
866                LEIA &       R+UU    ( MONITORBASE );
867                REESCREVA &  =UR     ( ADDR1 ),      SALVA PC DO PAI;
868
869 FPOEHLT:                    =0B,                    ESCALONACOND;
870
871 FPOEWHN:                    =0B,                    ESCALONACOND;
872
873 FPOEWAIT:                   R+1R    ( ACC );
874                             TT      ( ZERO ),       ESCALONAPRONTO;
875                             R-1R    ( ACC );
876                             =UR     ( BTBASE ),     CARREGACOND;
877
878
879
880
881 FIMTIRAFILA:                BRD     ( IR )
882                             MASCARA ( #B 000 110 )
883                             FTIRACRT, FTIRAHLT, $, FTIRAWAIT;
884
885 FTIRACRT:                   MCUD    ( SPPOS );
886                LEIA &       R-UU    ( BTBASE );     % CARREGA SP.
887                REESCREVA &  =UR     ( SP );
888                             MCUD    ( PCPOS );
889                LEIA &       R-UU    ( BTBASE );     % CARREGA PC.
890                REESCREVA &  =UR     ( PC ),         TESTA ERRO;
891
892 FTIRAHLT:                   =0B,                    ABEND;
893
894
895
896
897
898
899
900
```

MICRO-ASSEMBLER MITRA-15

```
901  NUCLEO:                    MCUD    ( STATUSPOS );
902                 LEIA &      R-UU    ( BTBASE );
903                 REESCREVA & =UR     ( ACC );        % ACC = STATUS
904                             =RZ     ( ACC );
905                             BRD     ( IR )
906                             MASCARA ( #B 000 110 )
907                             CREATE, HALT, WHEN, WAIT;
908
909  CREATE:                    =0B;
910                             BRD     ( IR )
911                             MASCARA ( #B 000 001 )  CRTFAT, CRTSON;
912
913  CRTFAT:    LEIA &          R+2R    ( PC );         %  LEITURA DO
914             REESCREVA &     R+UU    ( PC ), INITSP; %   TAM. DO COD.
915
916  CRTSON:                    TF      ( ZERO ),       ERRO;   % CRIT.
917                             MCUD    ( BTBASEPAI );
918             LEIA &          R-UU    ( BTBASE ),     TESTAFILHO;
919
920  HALT:                      TF      ( ZERO ),       ERRO;   % CRIT.
921  ABEND:                     MCUD    ( BTBASEPAI );
922             LEIA &          R-UU    ( BTBASE );
923             REESCREVA &     =UR     ( ADDR1 ),      TESTAFINALPROG;
924
925  WHEN:                      TF      ( ZERO ),       ERRO;   % CRIT.
926                             MCUD    ( STATUSPOS );
927             LEIA &          R-UU    ( BTBASE );
928             ESCREVA &       =1U,                    ESPERACONDICAO;
929
930  WAIT:                      TT      ( ZERO ),       ERRO;   % N-CRT
931                             MCUD    ( SEGUINTE );
932             LEIA &          R-UU    ( BTBASE );
933             REESCREVA &     =UR     ( ACC ),        TESTAFIMDAFILA;
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
```

MICRO-ASSEMBLER MITRA-15

```
951  INITSP:                        =UR     ( SP );
952                                 MCUD    ( 12 );
953                                 =UR     ( SZ );
954  CRIACONTEXTO:   LEIA &         =RU     ( SP );
955                                 R+2R    ( SP );
956                  ESCREVA &      =OU;
957                                 R-1R    ( SZ );
958                                 TF      ( ZERO ),       CRIACONTEXTO;
959                                 MCUD    ( BTBASEPAI );
960                  LEIA &         R-UU    ( BTBASE );
961                                 =OR     ( ACC );        %  BTBASEPAI =
962                  ESCREVA &      #RR     ( ACC );        %    #H FFFF
963                                 MCUD    ( FILAPRONTOS );
964                  LEIA &         R+UU    ( MONITORBASE );%  FILAPRONTOS =
965                  ESCREVA &      =RU     ( ACC );        %    #H FFFF
966                                 MCUD    ( FILACONDICAO );
967                  LEIA &         R+UU    ( MONITORBASE );%  FILACONDICAO=
968                  ESCREVA &      =RU     ( ACC );        %    #H FFFF
969                                 MCUD    ( BTSIZE );
970                  LEIA &         R+UU    ( MONITORBASE );
971                                 =RU     ( SP );
972                                 =UR     ( BTBASE );
973                  REESCREVA &    R+UR    ( SP ),         INICIO;
974
975
976
977  TESTAFILHO:     REESCREVA &    =UR     ( ACC1 );
978                                 R+1R    ( ACC1 );
979                                 TF      ( ZERO ),       E PROC FILHO;
980  SALVA BTBASE PAI:              MCUD    ( MONITBT );
981                  LEIA &         R+UU    ( MONITORBASE );
982                  ESCREVA &      =RU     ( BTBASE ),     SALVA SP DO PAI;
983
984  E PROC FILHO:                  =OB,                    ERRO;
985
986  SALVA SP DO PAI:               MCUD    ( SPPOS );
987                  LEIA &         R-UU    ( BTBASE );
988                  ESCREVA &      =RU     ( SP );
989  CONSTROIPROC:                  MCUD    ( MONITBT );
990                  LEIA &         R+UU    ( MONITORBASE );% RECUPERA
991                  REESCREVA &    =UR     ( BTBASE );     %  BTBASE DO PAI
992  INCNUMFILHOS:                  MCUD    ( NUMFILHOS );
993                  LEIA &         R-UU    ( BTBASE );
994                  REESCREVA &    =UR     ( ACC );
995                                 MCUD    ( NUMFILHOS );  % INCREMENTA
996                  LEIA &         R-UU    ( BTBASE );     %   O NUMERO
997                  ESCREVA &      R+1R    ( ACC );        %   DE FILHOS.
998  MAKENEWBTBASE:                 MCUD    ( CONTEXTSIZE );
999                                 R+UR    ( SP );         % SP = NEWBTBASE
1000                                MCUD    ( PRIOPOS );
```

MICRO-ASSEMBLER MITRA-15

```
1001                                R-UR    ( BTBASE );
1002                                MCUD    ( PRIOPOS );
1003                                R-UR    ( SP );
1004                                MCUD    ( 6 );
1005                                =UR     ( SZ );
1006  COPIACNTX:    LEIA &          =RU     ( BTBASE );     % COPIA BASE
1007                                R+2R    ( BTBASE );     %       STATUS
1008                REESCREVA &     =UR     ( ACC );        %       A
1009                LEIA &          =RU     ( SP );         %       E
1010                                R+2R    ( SP );         %       X
1011                ESCREVA &       =UR     ( ACC );        %       PRIO
1012                                R-1R    ( SZ );         %
1013                                TF      ( ZERO ),       COPIACNTX;
1014  MAKESP:                       MCUD    ( BTSIZE );
1015                LEIA &          R+UU    ( MONITORBASE );
1016                REESCREVA &     =UR     ( SZ );         % SZ = (BTSIZE)
1017                                MCUD    ( SPPOS );
1018                LEIA &          R-UU    ( SP );
1019                                =RU     ( SP );         % SP =
1020                ESCREVA &       R+UU    ( SZ );         % SP + (BTSIZE)
1021  MAKEPC:                       MCUD    ( PCPOS );
1022                LEIA &          R-UU    ( SP );
1023                                MCUD    ( 4 );
1024                ESCREVA &       R+UU    ( PC );         % PC = PC + 4
1025  MAKENOSONS:                   MCUD    ( NUMFILHOS );
1026                LEIA &          R-UU    ( SP );
1027                ESCREVA &       =0U;                    % NUMFILHOS = 0
1028  SAVEFATBTBS:                  MCUD    ( BTBASEPAI );
1029                LEIA &          R-UU    ( SP );         % BTBASEPAI =
1030                ESCREVA &       =RU     ( BTBASE );     %    BTBASE
1031                                =RU     ( SP );
1032                                =UR     ( ADDR1 );
1033                                =RU     ( SZ );
1034                                U/OR    ( SZ );
1035  COPIATABBASES: LEIA &         =RU     ( BTBASE );
1036                                R+2R    ( BTBASE );
1037                REESCREVA &     =UR     ( ACC );
1038                LEIA &          =RU     ( ADDR1 );
1039                                R+2R    ( ADDR1 );
1040                ESCREVA &       =RU     ( ACC );
1041                                R-1R    ( SZ );
1042                                TF      ( ZERO ),       COPIATABBASES;
1043  NEWBTBASE:                    =RU     ( SP );
1044                                =UR     ( BTBASE );
1045  GETPROCDSP:   LEIA &          R+2R    ( PC );
1046                REESCREVA &     =UR     ( ACC ),        % ACC = DSP
1047                                                        GETPROCSZ;
1048
1049
1050
```

MICRO-ASSEMBLER MITRA-15

```
1051  TESTAFINALPROG:                 =1U;
1052                                  R+UU    ( ADDR1 );
1053                                  TT      ( ZERO ),       SWSTD;
1054                                  MCUD    ( NUMFILHOS );
1055                  LEIA &          R-UU    ( ADDR1 );
1056                  REESCREVA &     =UR     ( ACC );
1057                                  MCUD    ( NUMFILHOS );
1058                  LEIA &          R-UU    ( ADDR1 );
1059                  ESCREVA &       R-1R    ( ACC );
1060                                  TF      ( ZERO ),       ESCALONACOND;
1061  FIMFILHOS:                      =RU     ( ADDR1 );
1062                                  =UR     ( BTBASE ),     POEFILAPRONTOS;
1063
1064  FTIRAWAIT:                      =0B,                    POEFILAPRONTOS;
1065
1066  GETPROCSZ:      LEIA &          R+2R    ( PC );
1067                                  =0R     ( SZ );
1068                  REESCREVA &     URR1    ( SZ );         % U = 256 * SZ
1069  INCSP:                          R+UR    ( SP );         % SP = SP + U
1070                                  MCUD    ( 6 );
1071                                  R-UR    ( ACC );        % U = OSP - 6
1072                                  R+UR    ( PC ),         POEFILAPRONTOS;
1073
1074
1075  ESPERACONDICAO:                 MCUD    ( SPPOS );
1076                  LEIA &          R-UU    ( BTBASE );
1077                  ESCREVA &       =RU     ( SP );         %  SALVA SP
1078                                  MCUD    ( PCPOS );
1079                  LEIA &          R-UU    ( BTBASE );
1080                  ESCREVA &       =RU     ( PC ),         POEFILACOND;
1081
1082
1083
1084  TESTAFINDAFILA:                 MCUD    ( FILACONDICAO );
1085                  LEIA &          R+UU    ( MONITORBASE );
1086                  REESCREVA &     R-UU    ( ACC );
1087                                  TF      ( ZERO ),       TESTACOND;
1088                                  =0R     ( ACC );
1089                                  R-1R    ( ACC );
1090  TESTACOND:                      R-1R    ( SP );
1091                  LEIA &          R-1R    ( SP );
1092                  REESCREVA &     U/0R    ( SZ );
1093                                  TT      ( CORINGA ),    FPOEWAIT;
1094                                  MCUD    ( STATUSPOS );
1095                  LEIA &          R-UU    ( BTBASE );
1096                  ESCREVA &       =0U;
1097                                  MCUD    ( PCPOS );
1098                  LEIA &          R-UU    ( BTBASE );
1099                  ESCREVA &       =RU     ( PC ),         TIRAFILACOND;
1100
```

MICRO-ASSEMBLER MITRA-15

```
1101
1102
1103  SALVA PC DO PAI:              MCUD    ( PCPOS );
1104                  LEIA &        R-UU    ( ADDR1 );
1105                                R-1R    ( PC );
1106                  ESCREVA &     R-1R    ( PC ),         ESCALONACOND;
1107  ESCALONACOND:                 MCUD    ( FILACONDICAO );
1108                  LEIA &        R+UU    ( MONITORBASE );
1109                  REESCREVA &   =UR     ( BTBASE );
1110                                R+1R    ( BTBASE );
1111                                TT      ( ZERO ),       ESCALONAPRONTO;
1112                                R-1R    ( BTBASE );
1113  CARREGACOND:                  =1U;
1114                                =UR     ( IR ),         FTIRACRT;
1115
1116  ESCALONAPRONTO:               MCUD    ( FILAPRONTOS );
1117                  LEIA &        R+UU    ( MONITORBASE );
1118                  REESCREVA &   =UR     ( BTBASE );
1119                                R+1R    ( BTBASE );
1120                                TF      ( ZERO ),       CARREGAPRONTO;
1121                                MCUD    ( FILACONDICAO );
1122                  LEIA &        R+UU    ( MONITORBASE );
1123                  REESCREVA &   =UR     ( BTBASE );
1124                                R+1R    ( BTBASE );
1125                                TT      ( ZERO ),       ERROFATAL;
1126                                R-1R    ( BTBASE ),     CARREGACOND;
1127
1128  CARREGAPRONTO:                R-1R    ( BTBASE );
1129                                =0R     ( IR ),         TIRAFILAPRONTOS;
1130
1131  TESTA ERRO:                   =RZ     ( IR );
1132                                TT      ( ZERO ),       FIMNUCLEO;
1133                  LEIA &        =RU     ( SP );
1134                                R+2R    ( SP );
1135                  ESCREVA &     =0U;
1136  FIMNUCLEO:                    MCUD    ( ERRIR );
1137                  LEIA &        R+UU    ( MONITORBASE );
1138                  REESCREVA &   =UR     ( ACC );
1139                                TT      ( ZERO ),       INICIO;
1140  SWSTD:                        MCUD    ( MONITBT );
1141                  LEIA &        R+UU    ( MONITORBASE );
1142                                =UR     ( ADDR1 );
1143                  ESCREVA &     =RU     ( BTBASE );     % MONITBT
1144                  LEIA &        R+2R    ( ADDR1 );
1145                  ESCREVA &     =RU     ( SP );         % MONITSP
1146                  LEIA &        R+2R    ( ADDR1 );
1147                  ESCREVA &     =RU     ( PC );         % MONITPC
1148                                =RU     ( MONITORBASE );
1149                                =UR     ( PC );
1150                                =UR     ( BTBASE );
```

MICRO-ASSEMBLER MITRA-15

```
1151                LEIA &          R+2R    ( ADDR1 );
1152                REESCREVA &     R+UR    ( BTBASE );     % STDCNTXL
1153                LEIA &          R+2R    ( ADDR1 );
1154                REESCREVA &     R+UR    ( PC );         % STDCNTXP
1155                LEIA &          R+2R    ( ADDR1 );
1156                REESCREVA &     =UR     ( ACC );        % STDSWMASC
1157                LEIA &          R+2R    ( ADDR1 );
1158                REESCREVA &     =UR     ( ADDR1 );      % STDSWADDR
1159                LEIA &          =UR     ( ADDR1 );
1160                ESCREVA &       =RU     ( ACC ),        INICIO;
1161
1162  ERRO:                         MCUD    ( ERRIR );
1163                LEIA &          R+UU    ( MONITORBASE );
1164                                =UR     ( ADDR1 );
1165                                =RU     ( IR );
1166                ESCREVA &       URRO    ( ACC );        % ERRIR
1167                LEIA &          R+2R    ( ADDR1 );
1168                ESCREVA &       =RU     ( SP );         % ERRSP
1169                LEIA &          R+2R    ( ADDR1 );
1170                ESCREVA &       =RU     ( PC );         % ERRPC
1171                LEIA &          R+2R    ( ADDR1 );
1172                ESCREVA &       =RU     ( BTBASE );     % ERRBTBASE
1173                                =RU     ( ACC );
1174                                U/OR    ( SZ );
1175                                TT      ( CORINGA ),    ABEND;
1176                                MCUG    ( HLTOPCODE );
1177                                =UR     ( IR ),         TIRAFILACOND;
1178
1179  ERROFATAL:                    MCUD    ( ERRIR );
1180                LEIA &          R+UU    ( MONITORBASE );
1181                                =UR     ( ADDR1 );
1182                REESCREVA &     =UR     ( ACC );
1183                LEIA &          =RU     ( ADDR1 );
1184                                MCUD    ( 2 );
1185                ESCREVA &       RIUI    ( ACC );        SWSTO;
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
```

MICRO-ASSEMBLER MITRA-15

```
1201  %
1202  %       ROTINAS DE INSERCAO E RETIRADA DE PROCESSOS
1203  %     DAS FILAS PRIORIZADAS DE PRONTOS E DE CONDICAO.
1204  %
1205
1206  POEFILAPRONTOS:             MCUD   ( FILAPRONTOS );
1207                              R+UU   ( MONITORBASE );
1208                              =UR    ( ENDFILA ),    POEFILA;
1209
1210
1211  POEFILACOND:                MCUD   ( FILACONDICAO );
1212                              R+UU   ( MONITORBASE );
1213                              =UR    ( ENDFILA ),    POEFILA;
1214
1215
1216  TIRAFILAPRONTOS:            MCUD   ( FILAPRONTOS );
1217                              R+UU   ( MONITORBASE );
1218                LEIA &        =UR    ( ENDFILA ),    TIRAFILA;
1219
1220
1221  TIRAFILACOND:               MCUD   ( FILACONDICAO );
1222                              R+UU   ( MONITORBASE );
1223                LEIA &        =UR    ( ENDFILA ),    TIRAFILA;
1224
1225
1226
1227  POEFILA:                    MCUD   ( MONITSP );
1228                LEIA &        R+UU   ( MONITORBASE );
1229                ESCREVA &     =RU    ( SP );         % SALVA SP
1230                              MCUD   ( MONITPC );
1231                LEIA &        R+UU   ( MONITORBASE );
1232                ESCREVA &     =RU    ( PC );         % SALVA PC
1233                LEIA &        =RU    ( ENDFILA );
1234                REESCREVA &   =UR    ( PRIMEIRO );
1235                              =UR    ( ADDR1 );
1236                              =1U;
1237                              R+UU   ( ADDR1 );
1238                              TT     ( ZERO ),       FILAVAZIA;
1239                              MCUD   ( PRIOPOS );
1240                LEIA &        R-UU   ( BTBASE );
1241                REESCREVA &   =UR    ( PRIORIDADE );
1242  PROXCOMPPOE:                MCUD   ( PRIOPOS );
1243                LEIA &        R-UU   ( ADDR1 );
1244                REESCREVA &   R-UU   ( PRIORIDADE );
1245                              TT     ( NEG ),        TESTAPRIMEIRO;
1246                              MCUD   ( SEGUINTE );
1247                LEIA &        R-UU   ( ADDR1 );
1248                REESCREVA &   =UR    ( ADDR1 );
1249                              R-UU   ( PRIMEIRO ),   TESTAFINALFILA;
1250
```

NICRO-ASSEMBLER MITRA-15

```
1251 TESTAPRIMEIRO:              =RU    ( PRIMEIRO );
1252                             R-UU   ( ADDR1 );
1253                             TF     ( ZERO ),       INSERENAFILA;
1254                LEIA &       =RU    ( ENDFILA );
1255                ESCREVA &    =RU    ( BTBASE ),     INSERENAFILA;
1256
1257 TESTAFINALFILA:             TF     ( ZERO ),       PROXCOMPPOE;
1258
1259 INSERENAFILA:               MCUD   ( ANTERIOR );
1260                LEIA &       R-UU   ( ADDR1 );      % PRIMEIRO =
1261                REESCREVA &  =UR    ( PRIMEIRO );   % ANTERIOR.
1262                             MCUD   ( ANTERIOR );
1263                LEIA &       R-UU   ( BTBASE );     % ANT DO INS =
1264                ESCREVA &    =RU    ( PRIMEIRO );   % ANTERIOR.
1265                             MCUD   ( ANTERIOR );
1266                LEIA &       R-UU   ( ADDR1 );      % ANT DO SEG =
1267                ESCREVA &    =RU    ( BTBASE );     % INSERIDO.
1268                             MCUD   ( SEGUINTE );
1269                LEIA &       R-UU   ( PRIMEIRO );   % SEG DO ANT =
1270                ESCREVA &    =RU    ( BTBASE );     % INSERIDO.
1271                             MCUD   ( SEGUINTE );
1272                LEIA &       R-UU   ( BTBASE );     % SEG DO INS =
1273                ESCREVA &    =RU    ( ADDR1 );      % SEGUINTE.
1274                             RECUPERASPPC;
1275
1276 FILAVAZIA:     LEIA &       =RU    ( ENDFILA );
1277                ESCREVA &    =RU    ( BTBASE );
1278                             MCUD   ( ANTERIOR );
1279                LEIA &       R-UU   ( BTBASE );
1280                ESCREVA &    =RU    ( BTBASE );
1281                             MCUD   ( SEGUINTE );
1282                LEIA &       R-UU   ( BTBASE );
1283                ESCREVA &    =RU    ( BTBASE ),     RECUPERASPPC;
1284
1285 RECUPERASPPC:               MCUD   ( MONITSP );
1286                LEIA &       R+UU   ( MONITORBASE );
1287                REESCREVA &  =UR    ( SP );
1288                             MCUD   ( MONITPC );
1289                LEIA &       R+UU   ( MONITORBASE );
1290                REESCREVA &  =UR    ( PC ),         FIMPOEFILA;
1291
1292
1293
1294 TIRAFILA:      REESCREVA &  R-UU   ( BTBASE );
1295                             TF     ( ZERO ),       NAOEOPRIMEIRO;
1296 PRIMEIRODAFILA:             MCUD   ( SEGUINTE );
1297                LEIA &       R-UU   ( BTBASE );
1298                REESCREVA &  =UR    ( ADDR1 );
1299                             R-UU   ( BTBASE );
1300                             TT     ( ZERO ),       FICOUVAZIA;
```

MICRO-ASSEMBLER MITRA-15

```
1301                LEIA    &       =RU     ( ENDFILA );
1302                ESCREVA &       =RU     ( ADDR1 );
1303  NAOEOPRIMEIRO:                MCUD    ( ANTERIOR );
1304                LEIA &          R-UU    ( BTBASE );
1305                REESCREVA &     =UR     ( ADDR1 );      % ADDR1 = ANT.
1306                                MCUD    ( SEGUINTE );
1307                LEIA &          R-UU    ( BTBASE );
1308                REESCREVA &     =UR     ( ENDFILA );    % ENDFILA = SEG
1309                                MCUD    ( ANTERIOR );
1310                LEIA &          R-UU    ( ENDFILA );
1311                ESCREVA &       =RU     ( ADDR1 );
1312                                MCUD    ( SEGUINTE );
1313                LEIA &          R-UU    ( ADDR1 );
1314                ESCREVA &       =RU     ( ENDFILA ),    FIMTIRAFILA;
1315
1316  FICOUVAZIA:   LEIA &          =RU     ( ENDFILA );
1317                                =OR     ( ADDR1 );
1318                ESCREVA &       R-1R    ( ADDR1 ),      FIMTIRAFILA;
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
```

MICRO-ASSEMBLER MITRA-15

```
1351  PLACE:                    R-1R    ( SP );
1352                  LEIA &      R-1R    ( SP );
1353                  REESCREVA & =UR     ( ACC );
1354                              R-1R    ( SP );
1355                  LEIA &      R-1R    ( SP );
1356                  REESCREVA & =UR     ( SZ );          % FLAG.
1357                              R-1R    ( SP );
1358                  LEIA &      R-1R    ( SP );
1359                  REESCREVA & =UR     ( ADDR1 );
1360                              =RZ     ( SZ );
1361                              TT      ( ZERO ),        PLACEMEM;
1362  PLACEREG:                   TT      ( NEG ),         IOTRAP;
1363                              RXOR    ( ADDR1 );
1364                              MCUD    ( AVALPOS );
1365                              R+UR    ( ADDR1 );
1366                  LEIA &      R-UU    ( BTBASE ),      ESCREVAPAL;
1367  PLACEMEM:       LEIA &      =RU     ( ADDR1 ),       ESCREVAPAL;
1368  IOTRAP:                     MCUD    ( ERRSP );
1369                  LEIA &      R+UU    ( MONITORBASE );
1370                  ESCREVA &   =RU     ( ACC ),         TRAP;
1371
1372  OBTAIN:                     R-1R    ( SP );
1373                  LEIA &      R-1R    ( SP );
1374                  REESCREVA & =UR     ( SZ );          % FLAG.
1375                              R-1R    ( SP );
1376                  LEIA &      R-1R    ( SP );
1377                  REESCREVA & =UR     ( ADDR1 );
1378                              =RZ     ( SZ );
1379                              TT      ( ZERO ),        OBTAINMEM;
1380                              TT      ( NEG ),         OBTAINREG;
1381                              =RJ     ( ADDR1 );
1382                              =RJU;
1383                              =UR     ( SZ ),          OBTAINSENSE2;
1384
1385  OBTAINREG:                  RXOR    ( ADDR1 );
1386                              MCUD    ( AVALPOS );
1387                              R+UR    ( ADDR1 );
1388                  LEIA &      R-UU    ( BTBASE ),      OBTAINSENSE1;
1389
1390  OBTAINMEM:      LEIA &      =RU     ( ADDR1 ),       OBTAINSENSE1;
1391
1392  OBTAINSENSE1:   REESCREVA & =UR     ( SZ );
1393  OBTAINSENSE2:               BRO     ( IR )
1394                              MASCARA ( #B 000 100 )   FOBTAIN, FSENSE;
1395  FOBTAIN:        LEIA &      =RU     ( SP );
1396                              R+2R    ( SP );
1397                  ESCREVA &   =RU     ( SZ ),          INICIO;
1398
1399  FSENSE:                     =UR     ( ACC );
1400                              R.UI    ( SZ );
```

MICRO-ASSEMBLER MITRA-15

```
1401                                TT      ( CY ),         PUSHA;
1402                                =OR     ( ACC );
1403                                #RR     ( ACC ),        PUSHA;
1404  PUSHA:        LEIA &          =RU     ( SP );
1405                                R+2R    ( SP );
1406  ESCREVAPAL:   ESCREVA &       =RU     ( ACC ),        INICIO;
1407
1408  SENSE:                        R-1R    ( SP );
1409                LEIA &          R-1R    ( SP );
1410                REESCREVA &     =UR     ( ACC ),        OBTAIN;
1411
1412  INITRAP:                      BRD     ( IR )
1413                                MASCARA ( #B 000 001 )  INIT, TRAP;
1414
1415  INIT:                         MCUD    ( MONITBT );
1416                LEIA &          R+UU    ( MONITORBASE );
1417                                =UR     ( ADDR1 );
1418                REESCREVA &     =UR     ( BTBASE );     % MONITBT.
1419                LEIA &          R+2R    ( ADDR1 );
1420                REESCREVA &     =UR     ( SP );         % MONITSP.
1421                LEIA &          R+2R    ( ADDR1 );
1422                REESCREVA &     =UR     ( PC ),         INICIO;
1423
1424  TRAP:                         =OB;
1425                                MCUD    ( ERRIR );
1426                LEIA &          R+UU    ( MONITORBASE );
1427                ESCREVA &       =RU     ( IR ),         SWSTD;
```

ANEXO VII. TEMPOS DE EXECUÇÃO DAS INSTRUÇÕES NAS IMPLEMENTAÇÕES REALIZADAS NO COMPUTADOR MITRA-15

A seguir é apresentada uma tabela que relaciona as instruções que compõem o repertório da máquina básica definida.

Nesta tabela estão os códigos de operação associados a cada uma destas instruções (em hexadecimal) nas duas implementações realizadas no computador MITRA-15, juntamente com os tempos necessários para execução destas instruções nestas implementações (em micro-segundos).

Esta tabela não inclui, porém, as instruções executadas pelo núcleo porque os tempos de execução destas instruções depende do estado da execução do programa.

1. Instruções para obtenção de endereços de variáveis

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGOS DA OPERAÇÃO	TEMPO DE EXECUÇÃO
ADDRL	48 H	498.3	00 H / 08 H	6.1
ADDRG	49 H	604.1	01 H / 09 H	6.7
ADDRLI	4A H	578.8	02 H / 0A H	7.0
ADDRGI	4B H	684.6	03 H / 0B H	7.6
ADDRLX	4C H	569.7	04 H / 0C H	7.1
ADDRGX	4D H	675.5	05 H / 0D H	7.7
ADDRLIX	4E H	650.2	06 H / 0E H	8.0
ADDRGIX	4F H	756.0	07 H / 0F H	8.6

2. Instruções para consulta a valores de variáveis de um "byte"

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
PUSHBL	00 H	449.0	10 H	7.6
PUSHBG	01 H	555.0	11 H	8.2
PUSHBLI	02 H	529.7	12 H	8.5
PUSHBGI	03 H	635.5	13 H	9.1
PUSHBLX	04 H	520.4	14 H	8.7
PUSHBGX	05 H	626.4	15 H	9.3
PUSHBLIX	06 H	601.1	16 H	9.6
PUSHBGIX	07	706.9	17 H	10.2

3. Instruções para atribuição de valores a variáveis de um "byte"

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
POPBL	10 H	451.3	18 H	9.0
POPBG	11 H	557.3	19 H	9.6
POPBLI	12 H	532.0	1A H	9.9
POPBGI	13 H	637.8	18 H	10.5
POPBLX	14 H	522.7	1C H	10.4
POPBGX	15 H	629.7	1D H	11.0
POPBLIX	16 H	603.4	1E H	11.3
POPBGIX	17 H	709.2	1F H	11.9

4. Instruções para consulta a valores de variáveis de dois "bytes"

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
PUSHWL	08 H	472.6	20 H	5.8
PUSHWG	09 H	578.6	21 H	6.4
PUSHWLI	0A H	553.3	22 H	6.7
PUSHWGI	0B H	659.1	23 H	7.3
PUSHWLX	0C H	544.0	24 H	7.0
PUSHWGX	0D H	650.0	25 H	7.6
PUSHWLIX	0E H	624.7	26 H	8.5
PUSHWGIX	0F H	730.5	27 H	10.2

5. Instruções para atribuição de valores a variáveis de dois "bytes"

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
POPWL	18 H	577.3	28 H	6.0
POPWG	19 H	583.3	29 H	6.6
POPWLI	1A H	558.0	2A H	6.9
POPWGI	1B H	663.8	2B H	7.5
POPWLX	1C H	548.7	2C H	7.1
POPWGX	1D H	655.7	2D H	7.7
POPWLIX	1E H	629.4	2E H	8.0
POPWGIX	1F H	735.2	2F H	8.6

6. Instruções para consulta a valores de variáveis de tamanho arbitrário

MNEMONICO DA INSTRUÇAO	IMPLEMENTAÇAO NO NIVEL CONVENCIONAL		IMPLEMENTAÇAO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇAO	TEMPO DE EXECUÇAO	CODIGO DA OPERAÇAO	TEMPO DE EXECUÇAO
PUSHSL	20 H	673.8*	30 H	5.2**
PUSHSG	21 H	760.1*	31 H	5.8**
PUSHSLI	22 H	754.3*	32 H	6.1**
PUSHSGI	23 H	840.6*	33 H	6.7**
PUSHSLX	24 H	745.2*	34 H	6.9**
PUSHSGX	25 H	831.5*	35 H	7.5**
PUSHSLIX	26 H	825.7*	36 H	7.8**
PUSHSGIX	27 H	912.0*	37 H	8.4**

* +3.3 * SZ

** se o endereço da fonte for par: +1.65 * SZ; senão +1.95 * SZ.
se SZ for Impar +1.0.

7. Instruções para atribuição de valores a variáveis de tamanho arbitrário

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
POPSL	28 H	698.1*	38 H	6.1**
POPSG	29 H	784.4*	39 H	6.7**
POPSLI	2A H	778.6*	3A H	7.0**
POPSGI	2B H	864.9*	3B H	7.6**
POPSLX	2C H	769.8*	3C H	7.5**
POPSGX	2D H	883.1*	3D H	8.1**
POPSLIX	2E H	877.3*	3E H	8.4**
POPSGIX	2F H	963.6*	3F H	9.0**

* + 3.3 * SZ.

** + 2.8 * SZ.

8. Instruções aritméticas e lógicas

MNEMÔNICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NÍVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NÍVEL DE MICRO-PROGRAMA	
	CÓDIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CÓDIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
ADD	30 H	471.0	40 H	4.8
SUB	31 H	471.0	41 H	4.8
MTPY	32 H	533.2	42 H	39.6*
DIV	33 H	541.0	43 H	48.0**
MOD	34 H	545.3	44 H	48.3**
NOT	35 H	353.7	45 H	6.2
OR	36 H	468.6	46 H	4.8
AND	37 H	468.6	47 H	4.8

* + 0.9 * Número de bits posicionados em um do multiplicador.

* Se o dividendo for negativo e o divisor for positivo: +1.5;

  Se o dividendo for positivo e o divisor for negativo: +1.2;

  Se o dividendo e o divisor forem negativos: +1.8.

9. Instruções para realizar comparações

MNEMONICO DA INSTRUÇAO	IMPLEMENTAÇAO NO NIVEL CONVENCIONAL		IMPLEMENTAÇAO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇAO	TEMPO DE EXECUÇAO	CODIGO DA OPERAÇAO	TEMPO DE EXECUÇAO
LSS	39 H	458.4* 453.0** 454.0***	49 H	7.2* 5.8** 5.8***
EQL	3A H	456.1* 455.3** 454.0***	4A H	6.9* 6.1** 5.8***
LEQ	3B H	458.4* 455.3** 454.0***	4B H	7.2* 6.1** 5.8***
GTR	3C H	456.1* 453.0** 456.3***	4C H	6.9* 5.8** 6.1***
NEQ	3D	458.4* 453.0** 456.3**	4D	7.2* 5.8** 6.1***
GEQ	3E	456.1* 455.3** 456.3***	4E	6.9* 6.1** 6.1***

* Se o primeiro valor for menor que o segundo.

** Se os dois valores forem iguais.

*** Se o primeiro valor for maior que o segundo.

10. Instruções para realização de operações com arranjos.

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGOS DA OPERAÇÃO	TEMPO DE EXECUÇÃO
IS	38 H	485.7(1)	50 H	7.5(2)
ISNT	3F H	491.0(1)	51 H	7.8(2)
CONSTS	6F H	382.7(3)	52 H	3.4(4)
BLANCS	68 H	392.6(5)	53 H	3.4(6)
PACK	56 H	394.1(7)	54 H / 55 H	4.0(8)
INDEX	57 H	609.8	56 H / 57 H	41.3(9)

(1) +12.35 * SZ (Se SZ for impar, +29.9;
Se as cadeias forem diferentes, +43.5).

(2) +1.05 * SZ (Se SZ for impar + 2.5;
Se as cadeias forem diferentes, +0.6).

(3) +36.9 * SZ

(4) +1.45 * SZ (+1.45 se SZ for impar)

(5) +3.75 * SZ

(6) +1.35 * SZ (+0.5 se SZ for impar)

(7) +35.1 * SZ

(8) +1.7 * SZ (+1.2 se SZ for impar)

(9) +0.9 * Número de bits posicionados em um de SZ.

11. Instruções para realização de operações com conjuntos

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CÓDIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CÓDIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
EMPTY	50 H	488.5	58 H	27.5
INCLUD	51 H	430.4*	59 H	3.3**
INCLUD1	-----	----	5A H	10.8***
IN	52 H	512.2	5B H	9.3****
UNION	53 H	576.5	5C H	63.7
INTER	54 H	576.5	5D H	63.7
DIFF	55 H	621.3	5E H	68.5

* +93.7 * SZ

** Para cada elemento incluido no conjunto são gastos mais
8.0 + 0.9 * valor do campo formado pelos quatro bits
menos significativos do valor do elemento a incluir

*** +0.9 * valor do campo formado pelos quatro bits
menos significativos do valor do elemengo a incluir

**** +0.9 * valor do campo formado pelos quatro bits
menos significativos do elemento cuja relação
de pertinência ao conjunto deseja-se verificar
(+0.6 se o elemento pertencer ao conjunto)

12. Instruções para atribuição de constantes e realização de desvios

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO
ZERO	----	----	60 H	3.6
TRUE	----	----	61 H	3.9
CONSTB	----	----	62 H	3.9
CONSTW	6E H	392.0	63 H	4.3
CODEAD	70 H		64 H	5.8
JP	6C H	299.6	65 H	3.7
JPFALSE	60 H	378.9*	66 H	55.2*
JPTRUE	----	----	67 H	5.2*

* +0.3 se o desvio for realizado

13. Instruções para chamar, iniciar e terminar rotinas

MNEMONICO DA INSTRUÇAO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇAO	TEMPO DE EXECUÇAO	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇAO
ALOC	68 H *		68 H	2.4
CALL	60 H	464.0	69 H	6.1
CALLI	61 H	619.9	6A H	6.9
PROCP	-----	-----	6B H	14.7
PROCPV	62 H	1186.0	6C H	15.6
RET	-----	-----	6D H	2.1
RETP	-----	-----	6E H	7.8
RETPV	63 H	648.0	6F H	8.7

14. Instruções para realização das operações especiais

MNEMONICO DA INSTRUÇÃO	IMPLEMENTAÇÃO NO NIVEL CONVENCIONAL		IMPLEMENTAÇÃO NO NIVEL DE MICRO-PROGRAMA	
	CODIGO DA OPERAÇÃO	TEMPO DE EXECUÇÃO	CODIGOS DA OPERAÇÃO	TEMPO DE EXECUÇÃO
PLACE	68 H	526.3*	78 H / 79 H	6.5*
OBTAIN	69 H	524.4	7A H / 7B H	6.6*
SENSE	6A H	624.5	7C H / 7D H	9.2*

* Referenciando a memória principal