SMC: SISTEMA DE MULTIPROCESSAMENTO CLEPSIDRA

Sérgio Amélio Ribeiro Cintra

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

Aprovada por:

Profª. Lídia Micaela Segre
(Presidente)

Profª. Sueli Mendes dos Santos

Prof. Euclides Robert Filho

Prof. Ricardo Correa de Oliveira Martins

Prof José Hiroki Saito

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 1984

CINTRA, SÉRGIO AMÉLIO RIBEIRO

SMC: Sistema de Multiprocessamento Clepsidra (Rio de Janeiro) 1984.

VII, 79 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1984).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

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

Aos meus pais
Silvio e Cristina e
à minha companheira Silvia

AGRADECIMENTOS

Ao companheiro Euclides pelos bons momentos de discussões e reflexões sobre a vida e também de discussões técnicas.

Às professoras Lidia e Sueli pela orientação e apoio.

À Universidade Federal de São Carlos pela cessão do laboratório para a programação e depuração dos módulos do SMC.

Ao Instituto de Pesquisas Espaciais, na pessoa do Dr. Eduardo Whitaker Bergamini, por propiciar a continuidade deste trabalho.

À Maria Aparecida G. Lima e a Sueli Maria Vicente pela paciência e destreza apresentadas quando da datilografia desta dissertação.

A todos que direta ou indiretamente me auxiliaram ao longo do tempo para que este objetivo fosse alcançado.

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

SMC: SISTEMA DE MULTIPROCESSAMENTO CLEPSIDRA

Sérgio Amélio Ribeiro Cintra

SETEMBRO, 1984

Orientadora: Lídia Micaela Segre
Programa: Engenharia de Sistemas e Computação

Este trabalho descrece a especificação de um Sistema Operacional, visando realizar multiprocessamento, para o computador Clepsidra. Este computador foi projetado originalmente para processamento a Fluxo de Dados, que possue uma taxa muito alta de comunicação entre as unidades do sistema que executam em paralelo (Tarefas). O Sistema Operacional especificado procura diminuir a taxa de comunicação entre as Tarefas, através do aumento do tamanho destas, proporcionando uma diminuição no tempo de processamneto inútil ("overhead"), embora diminua o paralelismo. Esta especificação consiste na: forma de decomposição do programas em Tarefas, forma de alocação e comunicação entre elas, e detecção e recuperação de falhas.

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

SMC: CLEPSIDRA MULTIPROCESSING SYSTEM

Sérgio Amélio Ribeiro Cintra

SEPTEMBER, 1984

Chairman: Lídia Micaela Segre
Department: Engenharia de Sistemas e Computação

This work presents a specification of an Operating System, which manages multiprocessing system for the Clepsidra computer. This computer is a data flow machine with a high rate communication among its tasks. This Operating System decreases the communication rate among tasks by increasing their lenght which decreases the overhead time wasted in communication control, even decreasing the amount of parallelism. This specification consists of: program decomposition in tasks, alocation and communication among them, and faults detection and recovering.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

Ao longo da história da evolução dos sistemas de computação, houve um esforço grande no sentido de aumentar o desempenho deles, que resultou em mais avanços em "hardware" do que em "software". Isto pode ser constatado ao se verificar as características básicas das gerações de computadores. Elas estão diretamente relacionadas com o avanço tecnológico dos componentes empregados no desenvolvimento do "hardware". Assim, como pode ser visto em ROSEN (1) e DENNIS (2), a primeira geração é constituída basicamente por aqueles computadores em cujas construções foram empregadas válvulas (início em 1946, computador ENIAC), a segunda por computadores nos quais foram utilizados transistores (início em 1956, computador TX-0), e a terceira geração, de 1965 em diante, caracteriza-se pelo emprego de circuitos integrados.

A fabricação de integração de circuitos passou por avanços consideráveis, representados por tecnologias de integração. A mais recente (década 70), denominada "Very Large Scale Integration", permite atualmente uma integração superior, em circuitos comerciais, a 4 centenas de milhares de transistores em espaços menores do que $35mm^2$, como mostrado em BEYERS (3), sendo por isto considerada como uma das características de computadores da quarta geração.

Mas, apesar do aumento de desempenho conseguido, devido ao constante desenvolvimento da microeletrônica, somado com a introdução de variações ao modo de processamento sequencial, como, entre outras, fazer operações de entrada e saída e da unidade central de processamento sobrepostas no tempo, reduzir o tempo de execução de cada instrução, através da decomposição desta em etapas, dando origem a arquitetura de encadeamento ("pipeline"). Ainda assim, estes avanços não satisfizeram as necessidades de processamento exigidas pelas aplicações mais complexas em computação.

Em vista disto, para sobrepujar as limitações existentes, foram criadas estruturas lógicas que proporcionam execuções concorrentes, seja de forma intercalada num único processador ou de forma paralela em vários processadores conectados.

A alternativa óbvia, em nível de "hardware", para a implantação dessas estruturas, como pode ser visto em STONE (4) e FLORENTIN (5), entre outros, é com o auxílio de um tipo de arquitetura de computador que provê vários processadores conectados de alguma forma, permitindo processamento paralelo.

Em "software", uma das maneiras de se implantar essas estruturas lógicas é através da detecção da concorrência a nível de algoritmo do programa, particionando-o em partes que possam ser executadas em paralelo, chamadas Tarefas, de forma cooperativa entre si.

Um sistema que suporta a execução destas Tarefas é denominado de sistema Multi-tarefas.

Este trabalho lida com sistema de processamento concorrente, implementado através de um método Multi-tarefas, fazendo uso de um computador do tipo Múltiplos Processadores ("Multiple Processors"), como definido em ENSLOW JR. (6), e pertencente à classe "Multiple Instruction Multiple Data", usando a Taxionomia de FLYNN (7).

O objetivo deste trabalho é especificar, e descrever a implementação de um Sistema Operacional, suportando multiprocessamento para a execução de programas gerais. Este objetivo foi escolhido, principalmente, devido aos Sistemas Operacionais desta classe ainda serem pouco conhecidos, sendo portanto mais interessante explorar a estrutura global destes sistemas, do que procurar aperfeiçoar suas partes. De acordo com este ponto de vista, propõe-se um Sistema Operacional simples, porém completo para um computador que permite processamento concorrente de forma paralela, chamado Clepsidra. Este computador se encontra no Departamento de Computação e Estatística da Universidade Federal de São Carlos, em São Carlos, estado de São Paulo.

O fato do Sistema Operacional ser pensado para uma arquitetura específica, visa dois aspectos: limitar o escopo de investigação, que de outra maneira poderia ser muito amplo, e servir de campo de prova para testar a exequibilidade das soluções propostas.

Para situar este trabalho no contexto do processamento paralelo, o Capítulo II descreve a arquitetura de computador do tipo Múltiplos Processadores, envolvendo conceituação, alguns motivos para o desenvolvimento de sistema de computação com esta arquitetura e considerações sobre ela. Em seguida são descritas as características da arquitetura do computador Clepsidra e a sua classificação quanto a arquitetura Múltiplos Processadores.

A proposta do Sistema Operacional, denominado Sistema de Multiprocessamento Clepsidra (SMC), é descrita no Capítulo III, mostrando sua concepção, estrutura e o Sistema Multi-tarefas para o computador Clepsidra.

No Capítulo IV, os módulos de programas que realizam o SMC são comentados e especificados, implementando os conceitos e as considerações feitas no Capítulo III.

Um exemplo de execução de Tarefas de um programa é mostrado no Capítulo V, através de uma aplicação.

As considerações sobre o SMC implementado, suas qualidades e restrições, enfim o comportamento do sistema é levantado e discutido no Capítulo VI, procurando desta forma, contribuir para o desenvolvimento de sistemas de processamento paralelo.

CAPÍTULO II

ARQUITETURA PARA PROCESSAMENTO PARALELO

II.1 - INTRODUÇÃO

A arquitetura de computador mais comum em uso, que é a de processamento sequencial, tem sido mantida, fundamentalmente, nas últimas quatro décadas, desde que foi desenvolvida por John VON NEUMAN (8). Esta arquitetura possue uma organização bastante simples, consistindo de cinco unidades, mostrada na figura (II.1). O seu ponto fraco, no atual estágio de desenvolvimento da microeletrônica, é o fato dela executar somente uma instrução por vez. Esta arquitetura clássica permitiu, através das várias gerações de computadores, um aumento de desempenho de várias ordens de magnitude nos sistemas de computação.

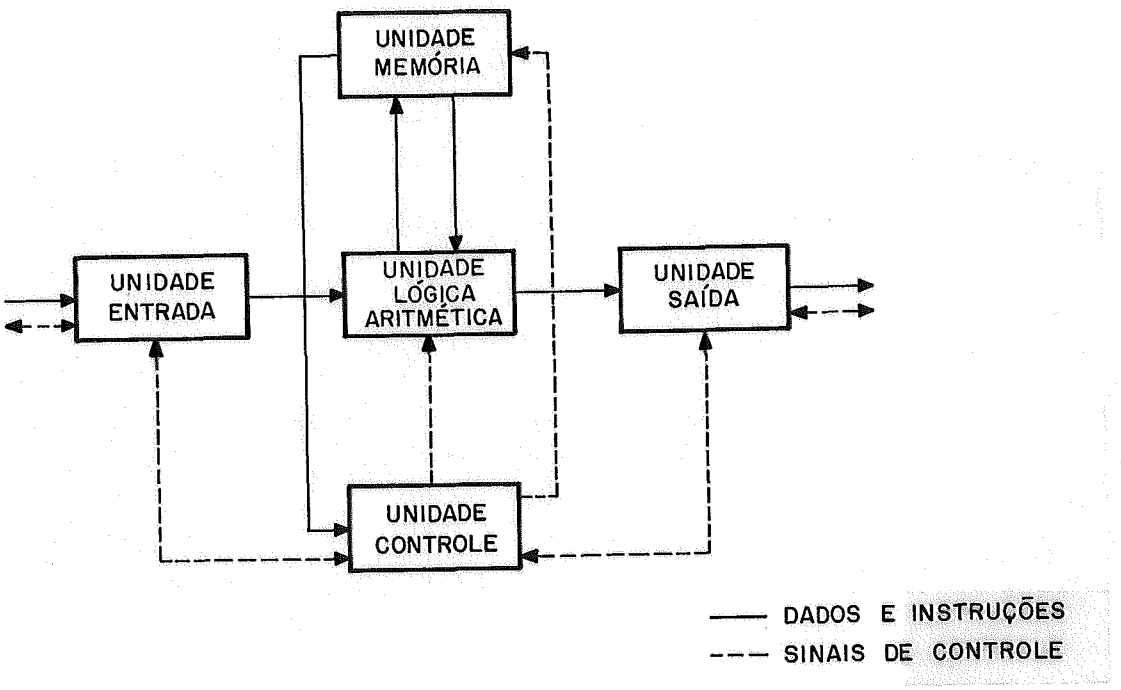


Figura II.1 - Unidades básicas da arquitetura de VON NEUMAN.

Todavia, essa arquitetura clássica não satisfaz as necessidades de processamento exigidas pelas aplicações mais complexas. Por esta razão, alguns tipos de arquitetura foram desenvolvidos como alternativas à arquitetura de VON NEUMAN, permitindo processamento em paralelo. Estes tipos de arquiteturas, por exemplo o da Figura (II.2), envolvem vários processadores, uma ou mais memórias compartilhadas e/ou privativas, e um mecanismo de interconexão rápido entre estes módulos.

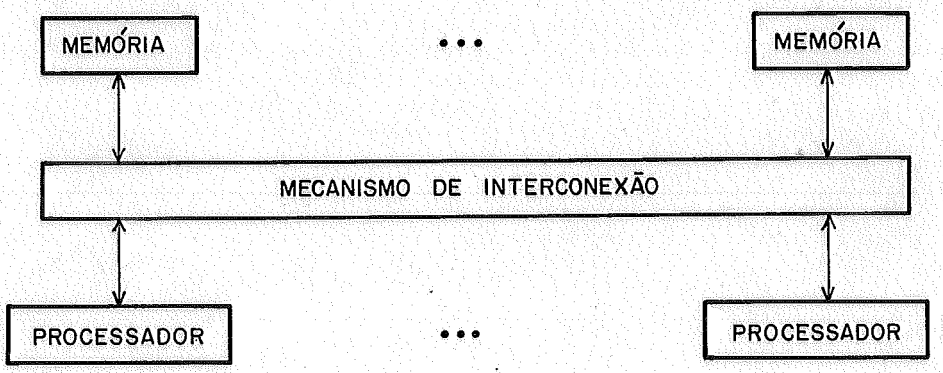


Figura II.2 - Arquitetura para processamento paralelo.

Essas arquiteturas podem ser divididas, segundo HAYNES et alii (9), em: Processadores Vetoriais ("Array Processors"), Computadores a Fluxo de Dados ("Data Flow Computers"), Unidades Funcionais Multiplas de Finalidade Especial ("Multiple Special-Purpose Functional Units"), Processadores Associativos ("Associative Processors"), Máquinas de Linguagem de Programação Funcional ("Functional Programming Language Machines") e Múltiplos Processadores ("Multiple Processors"). Estas arquiteturas possuem características em comum, e procuram responder às principais dificuldades de implementação de processamento paralelo.

Dentre estas arquiteturas, a Múltiplos Processadores é descrita na seção seguinte, por ser a arquitetura com a qual Clepsidra se identifica. A arquitetura Multiplos Processadores, como discutido em Haynes et alii (9), é mais flexível do que as outras, embora necessite de um controle mais complexo. É flexível devido à possibilidade de

alocação nos Operadores de Tarefas de tamanhos distintos, com tempos de execução distintos, permitindo comunicação entre estas Tarefas de forma assíncrona. Além disto, permite reconfiguração simples, através da facilidade de redistribuição da carga de trabalho. É uma arquitetura que necessita de um controle complexo devido principalmente, as comunicações entre Tarefas ocorrerem de forma assíncrona. Posteriormente, a arquitetura Clepsidra é descrita.

II.2 - ARQUITETURA MÚLTIPLOS PROCESSADORES

A arquitetura Multiplos Processadores é considerada aqui como sendo a conexão de dois ou mais processadores. Estes processadores são capazes de executar instruções independentes e trocar informações através de um mecanismo de interconexão.

Esse tipo de arquitetura é apropriada para,entre outras, as seguintes aplicações:

- Aplicações que demandam alta confiabilidade (facilidade de introdução de mecanismos de tolerância a falhas);

- Aplicações voltadas para o controle de processos, seja em tempo real ou não (possibilidade de dedicação de processadores a processos distintos).

As características dessa arquitetura são descritas a seguir, objetivando conceituá-las, mostrar as motivações e considerações do seu projeto. Para obter uma descrição mais detalhada, sugerimos a leitura dos trabalhos: SIEWIOREK et alii (10), FULLER et alii (11), ROBERTS (12), FATHI et KRIEGER (13) e SILVERMAN et alii (14), os quais forneceram subsídios para as seções seguintes.

II.2.1 - CONCEITUAÇÃO

A conceituação é feita quanto ao modo de processamento e a forma de conexão entre os processadores.

II.2.1.1 - MODO DE PROCESSAMENTO

Pela Taxionomia de FLYNN (6), a arquitetura Multiplos Processadores se classifica como "Multiple Instruction Multiple Data". Esta classificação é a mais global, entre as definidas por FLYNN (6), podendo ter diferentes processadores, onde cada um possui o seu próprio controle, permitindo processamento paralelo.

II.2.1.2 - FORMAS DE CONEXÃO

As formas de conexão são: Fortemente Conectada, Fracamente Conectada e Redes, cujas características, entre outras, são descritas abaixo.

a) Fortemente Conectada

- Os processadores podem acessar uma memória compartilhada, através da qual se comunicam, podendo também ter sua própria memória;

- Os processadores operam sob um esquema de controle estrito;

- Existe autonomia quanto ao tempo de processamento dedicado a cada tarefa por qualquer processador;

- Há necessidade de sincronizar os processadores que são cooperantes durante uma execução.

b) Fracamente conectada

- Os processadores são autonomos, podendo ser geograficamente dispersos;

- A comunicação é feita através de troca de mensagens, utilizando protocolo bastante rígido;

- O meio físico para a comunicação é implementado através

de interfaces para linhas seriais de alta velocidade.

c) Redes

- Sistemas de computação autônomos e completos com todos os subsistemas necessários para execução local, interconectados para compartilhar informações;

- Sistemas de computação dedicados à Tarefas, alocadas conforme as capacidades de processamento destes sistemas;

- Os processos são distribuídos de forma estática, devido à característica anterior;

- A comunicação se restringe mais a troca de dados, às vezes são trocados comandos, que por sua vez podem requisitar dados.

II.2.2 - MOTIVAÇÕES

Entre as várias motivações para projetos de arquitetura Multiplos Processadores, são destacadas algumas, abaixo descritas:

a) Custo/Desempenho

O avanço da tecnologia de integração de circuitos, como a técnica "Very Large Scale Integration", possibilitou o desenvolvimento de processadores monolíticos de baixo custo. Isto, tornou atrativo a conexão de vários destes processadores, de forma a se obter um sistema de computação de alto desempenho.

b) Performance

O aumento da performance, considerada como sendo o número de operações executadas por unidade de tempo, aparece naturalmente, pois vários processadores operam simultâneamente.

É conveniente salientar que o aumento do número de processa

dores, além do necessário, em um sistema de computação, pode levar a uma situação de degradação da performance. Esta degradação é causada pelo aumento de conflitos que ocorrem durante os acessos feitos pelos processadores aos recursos compartilhados. Além disto, também pelo aumento da quantidade de informação de controle a ser gerenciada, a qual não constitui processamento útil ("overhead").

c) Modularidade

A característica de modularidade ocorre na arquitetura Multiplos Processadores através da possibilidade de isolamento de partes do processamento em cada processador. Esta característica torna fácil a agregação de mais recursos, permitindo suprir ao aumento de demanda.

d) Tolerância a Falhas/Degradação Suave

A arquitetura Multiplos Processadores, como discutido em FATHI (13), facilita a adição de mecanismos de tolerância a falhas. Por exemplo, uma característica forte é permitir aos processadores confrontarem os resultados (processo de votação).

Também, nesse tipo de arquitetura, quando um dos processadores deixa de funcionar o computador pode continuar trabalhando após a realocação da sua carga de trabalho, embora com performance menor, caracterizando uma degradação suave no processamento.

II.2.3 - CONSIDERAÇÕES SOBRE ARQUITETURA MÚLTIPLOS PROCESSADORES

Para o projeto de arquitetura tipo Multiplos Processadores, devem ser feitas considerações, basicamente, a respeito da introdução da concorrência, da diminuição dos conflitos de acesso a recursos compartilhados, e para se evitar bloqueio perpétuo ("deadlock").

Para tanto, especial atenção deve ser dedicada quanto:

- à forma de particionar um programa em Tarefas, de maneira a permitir o maior grau de concorrência na execução daquele programa;

- à alocação das Tarefas aos Processadores, que deve observar o potencial de execução do computador, objetivando o maior grau possível de cooperação entre elas, permitindo aumentar a performance da execução como um todo;

- à alocação dos recursos físicos às Tarefas, devendo ser feita de maneira a tratar estes recursos de forma lógica, permitindo o acesso a eles por qualquer Tarefa em execução, não importando em qual processador a Tarefa esteja alocada;

- às decisões a serem tomadas quando da existência de conflitos de acesso a recursos compartilhados, que devem observar o potencial de execução do computador, juntamente com as prioridades de execução das Tarefas envolvidas no conflito naquele instante, para poder realizar o tratamento mais adequado;

- ao controle das execuções das Tarefas, que cabe ao sistema operacional do computador. Este último deve se relacionar com as Tarefas através de eventos que traduzam os estados de execução delas e os dos recursos requisitados.

II.3 - ARQUITETURA DO COMPUTADOR CLEPSIDRA

O objetivo da arquitetura Clepsidra, apresentada em ROBERT FILHO (15), mostrada na figura (II.3), é propor uma organização baseada em microprocessadores, trabalhando cooperativamente, visando obter um computador de alta performance. Esta arquitetura foi projetada inicialmente para uso em fluxo de dados ("data flow").

Fluxo de dados implica numa alta taxa de troca de informações entre os operadores do sistema (microprocessadores). Foi então, escolhida uma malha reticulada como suporte físico das vias de comunicação entre os operadores.

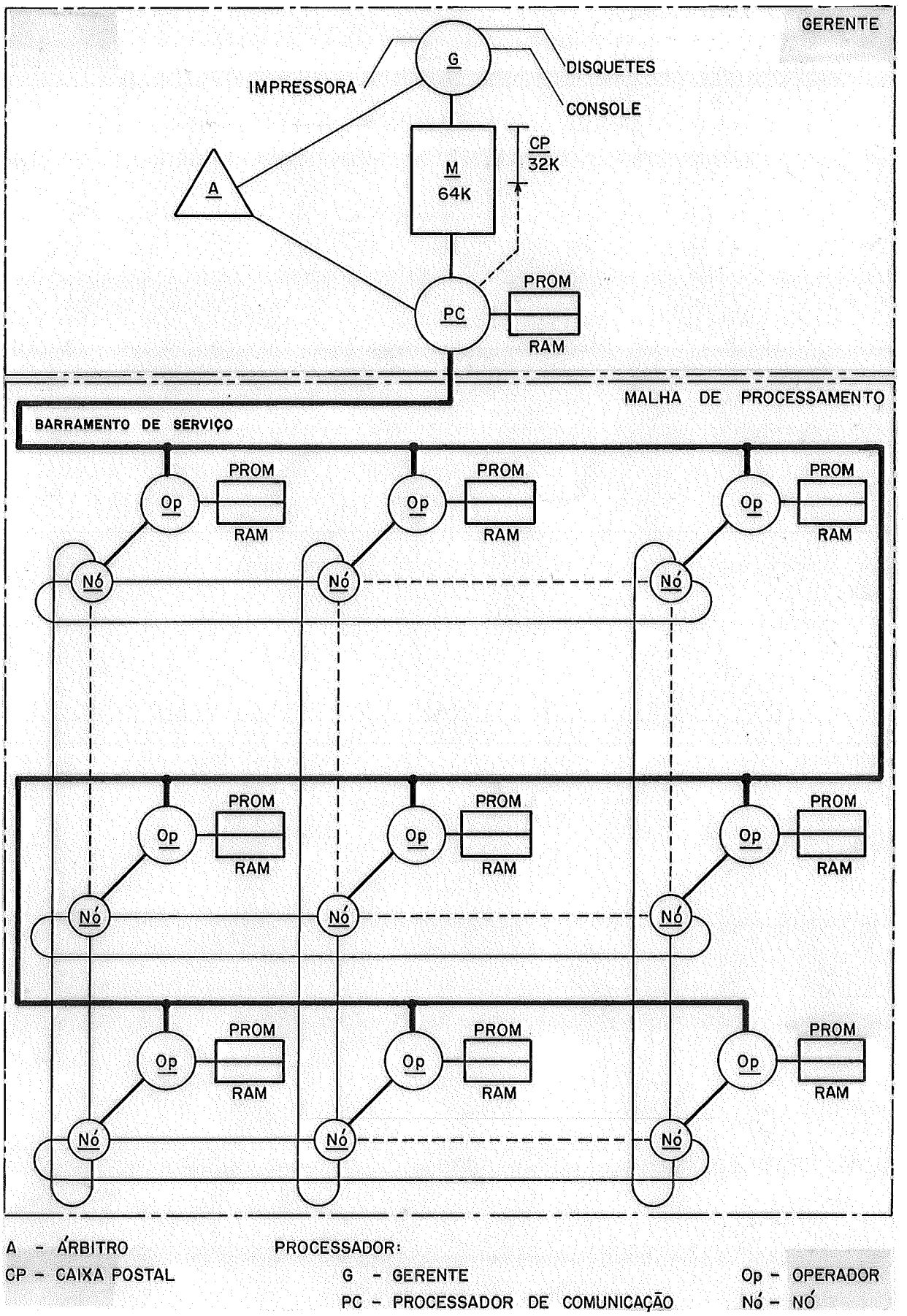


Figura II.3 - Arquitetura Clepsidra.

Em cada cruzamento do retículo foi colocado um microprocessador de comunicação chamado Nó. A cada Nó está ligado um microprocessador de cálculo: o Operador.

Esse conjunto de microprocessadores, organizados em retículo, recebeu o nome de Malha de Processamento (MP).

Os programas a serem executados podem ter mais Tarefas do que o número total de Operadores do sistema. Como a Tarefa é a unidade de processamento do Operador, decorre a necessidade de um elemento capaz de gerenciar a aplicação das Tarefas existentes nos Operadores disponíveis, tomando para si a qualidade de mestre e deixando para a MP a qualidade de escravo do sistema. Este elemento é o Gerente, que se comunica ainda com o ambiente externo recebendo programas e dados, e expedindo resultados via terminal de video, disco e impressora. Para se comunicar com os Operadores, o Gerente dispõe de um Barramento de Serviço ligado a todos eles e que não faz parte do retículo.

O Gerente é fisicamente constituído de dois microprocessadores. Um deles é o Gerente propriamente dito. O outro é o Processador de Comunicação.

A arquitetura Clepsidra é uma arquitetura Multiplos Processadores seção (II.2). Quanto ao modo de processamento, segundo a Taxionomia de FLYNN (7) ela é uma arquitetura "Multiple Instruction Multiple Data". A razão é que os Operadores trabalham assincronamente, executando Tarefas que podem ser diferentes entre si quanto ao código e ou conjunto de dados. Quanto às formas de conexão apresentadas na seção (II.2.2), ela não é seguramente uma Rede. Encaixa-se pouco em Fracamente Conectada e encaixa-se mais, porém ainda parcialmente, em Fortemente Conectada.

II.3.1 - MALHA DE PROCESSAMENTO

A Malha de Processamento (MP) tem a forma de um retículo de n X n Nós e seus Operadores associados, com os extremos interligados, como mostrado na figura (II.3).

O microprocessador Nó, situado em cada cruzamento do reticu

lo, é o encarregado da propagação de mensagens na MP. O Nó tem por função controlar o tráfego que por ele passa, dirigindo as mensagens aos destinos expressos nos seus campos de endereço. As mensagens produzidas por seu Operador associado são inseridas no retículo e as que são destinadas ao seu Operador associado são retiradas do retículo e a ele enviadas.

O microprocessador Operador tem por função executar Tarefas. As Tarefas são carregadas no Operador através de um diálogo com o Processador de Comunicação (PC), via Barramento de Serviço. Durante sua execução, uma Tarefa pode produzir mensagens destinadas a outras Tarefas alocadas em outros Operadores. Estas mensagens são enviadas através do retículo. As Tarefas também podem produzir mensagens destinadas ao Gerente. Estas mensagens são enviadas ao PC via Barramento de Serviço.

II.3.2 - GERENTE

A função básica do Gerente na arquitetura Clepsidra é gerar a macroordenação das Tarefas, que compõe os programas em execução em um dado instante.

A macroordenação consiste em, respeitando o grafo de sequenciamento entre as Tarefas, gerado por um processo de compilação, alocá-las nos Operadores, de maneira a obter a maior taxa de execução útil possível.

Como já descrito, o Gerente é realizado fisicamente por dois microprocessadores. O primeiro é o Gerente propriamente dito, que se ocupa da entrada e saída com os periféricos e da macroordenação das Tarefas. O segundo é o Processador de Comunicação (PC) que se ocupa da comunicação com os Operadores através do Barramento de Serviço. O PC carrega o código das Tarefas e os dados respectivos nos Operadores, e durante a execução das Tarefas, pode enviar e receber mensagens contendo dados, resultados ou controles gerados pelos Operadores.

O PC e o Gerente se comunicam através de memória compartilhada, dotada de uma organização do tipo de Caixa Postal.

II.3.3 - FORMAS DE COMUNICAÇÃO

A arquitetura Clepsidra figura (II.3) comporta as seguintes formas de comunicação entre os seus elementos constituintes:

II.3.3.1 - COMUNICAÇÃO NÓ/NÓ E NÓ/OPERADOR

Os Nós são providos de cinco barramentos. Um deles é destinado à comunicação com o Operador associado. Os outro quatro são iguais entre si e ligam o Nó aos seus quatro vizinhos: da esquerda, da direita, do alto e do baixo.

Os Nós trabalham de forma síncrona, num ciclo de duas fases de mesma duração: recepção/roteamento de mensagens e transmissão de mensagens, mostrado na figura (II.4). Quando um Nó está na fase de recepção/roteamento, todos os seus vizinhos estão na fase de transmissão. Finda a fase de recepção/roteamento o Nó passa para a fase de transmissão e todos os vizinhos para a fase de recepção/roteamento.

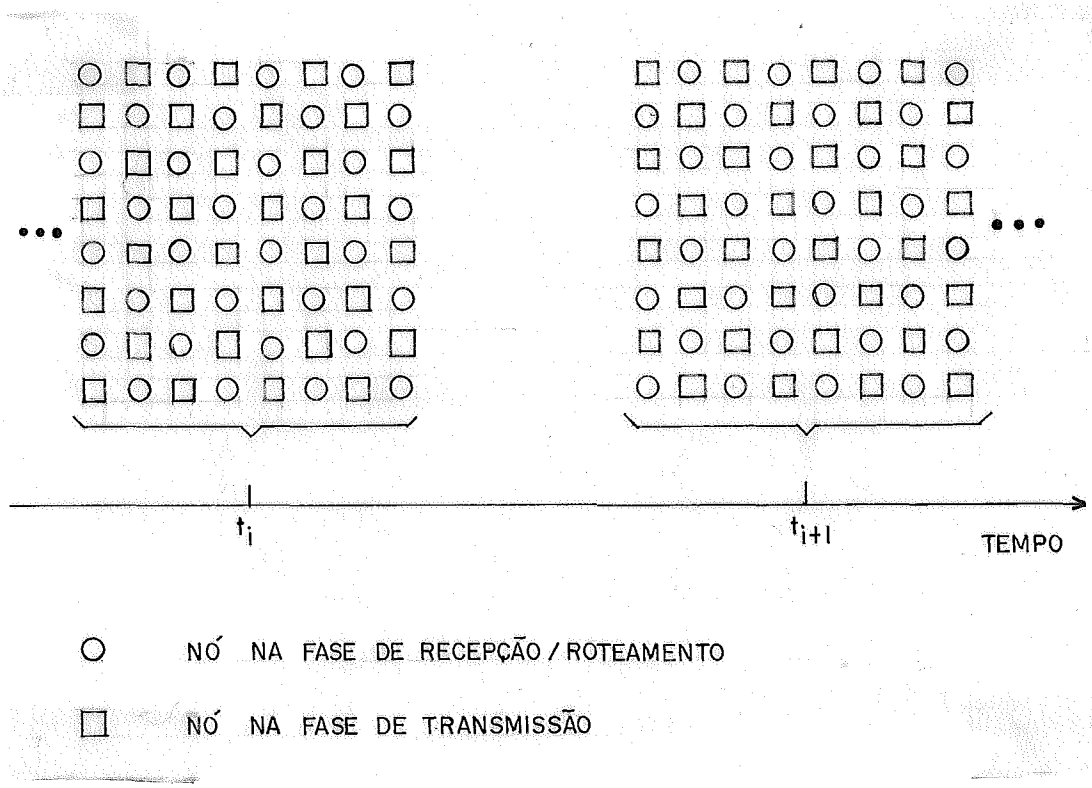


Figura II.4 - Fases de Comunicação do Nós.

Vale a pena lembrar que o Nó é capaz de interromper o Operador associado. Este último apenas pode sinalizar através de um "flag" um pedido de comunicação com o Nó.

II.3.3.2 - COMUNICAÇÃO OPERADOR/PROCESSADOR DE COMUNICAÇÃO

O Processador de Comunicação (PC) se interliga com os Operadores através de um Barramento de Serviço. O PC, quando deseja iniciar uma comunicação com um determinado Operador, interrompe-o, obrigando-o a iniciar um protocolo de comunicação. O Operador quando deseja iniciar uma comunicação com o PC, ativa um bit de "flag", que é consultado pelo PC durante a varredura destinada a verificar quais Operadores desejam iniciar comunicação.

O Barramento de Serviço dispõe de oito fios para transporte de dados num sentido, oito fios para transporte no sentido inverso e 2X m fios para controle, onde m representa o número de Operadores da máquina. Cada Operador está ligado ao PC através de dois fios particulares de controle. Um serve para que o PC interrompa o Operador e o outro transporta o "flag" enviado pelo Operador ao PC.

II.3.3.3 - COMUNICAÇÃO PROCESSADOR DE COMUNICAÇÃO/GERENTE

O Processador de Comunicação (PC) e o Gerente se comunicam através de memória compartilhada, dotada de uma organização do tipo Caixa Postal. O PC e o Gerente utilizam um barramento único para acesso à memória compartilhada. O acesso ao barramento único é gerenciado por um árbitro de barramento (descrito na seção seguinte).

II.3.4 - "HARDWARE" DO CLEPSIDRA

O "hardware" Clepsidra é composto de dois blocos maiores: o Microcomputador Gerente e a Malha de Processamento. Na implementação atual, nenhum dos dois dispõe de relógio em tempo real.

a) Gerente

O Microcomputador Gerente é realizado por dois microprocessa

dores, o Gerente propriamente dito e o Processador de Comunicação (PC). Ambos foram construídos usando a família MOTOROLA (16).

O Gerente é um microcomputador de configuração convencional: dois acionadores de disco flexível de oito polegadas, terminal de vídeo, impressora, 64K "bytes" de memória. Uma característica deste microprocessador é a existência de uma árbitro em "hardware", que aloca o barramento do microprocessador, pelo período de um ciclo de máquina a cada vez (um micro segundo ou mais se houver extensão de ciclo), a um dos demandadores de barramento: PC, Gerente, ADM (disco) e Memória (refrescamento).

O PC foi adicionado ao Gerente com a função de aliviar-lhe a carga de trabalho. O PC compartilha o barramento do Gerente através do uso do árbitro acima mencionado. Este compartilhamento de barramento dá-lhe acesso à memória do Gerente, todavia limitado aos 32K "bytes" iniciais pelo mecanismo de endereçamento do próprio PC. O PC dispõe ainda de uma memória interna, não submetida à arbitragem, onde estão os seus programas de controle e a área de trabalho. O acesso do PC à memória compartilhada, diminui a velocidade de processamento do Gerente.

Por outro lado, o PC dispõe de um Barramento de Serviço ao qual estão ligados todos os Operadores. Este barramento foi implementado com interfaces paralelas MC6821, descritas em MOTOROLA (16). Para evitar reprogramação constante quanto ao sentido dos dados (emissão ou recepção) foram utilizados dois portos de oito bits cada: um para emissão e outro para recepção de dados. Do ponto de vista do controle, o PC dispõe de uma saída e uma entrada de MC6821 para cada Operador a que está conectado. A saída serve para interromper o Operador e a entrada serve para receber um "flag" do Operador, esta última no sentido de pedir uma interrupção.

b) A MALHA DE PROCESSAMENTO

A Malha de Processamento pode conter até oito por oito Nós e seus Operadores associados. O protótipo construído contém quatro Nós e quatro Operadores, configuração mínima para demonstrar todos os mecanismos de comunicação.

O Nó foi realizado com os circuitos da família Signetics

8X300, descrito em SIGNETICS (17). Ele dispõe de uma memória EPROM de 2K "bytes", 512 "bytes" de RAM e cinco portos (8T32) para comunicação com os quatro Nós vizinhos e com o Operador associado. O 8X300 é um microprocessador próprio para comunicação de dados. Ele trabalha num ciclo de 250 ns. A cada ciclo, ele é capaz de receber um dado de um porto, tratá-lo (deslocamento, rotação, incremento, decremento) e enviá-lo através de um outro porto. O 8X300 não dispõe de mecanismo de interrupção.

O Operador foi construído com base na família do microprocessador MC6809, descrito em MOTOROLA (16). Ele dispõe de uma memória EPROM de 2K "bytes" e de uma memória RAM de 2K "bytes". O Operador se comunica com o PC e com o seu Nó associado. Em ambos os casos, esta comunicação é feita através da interface MC6821 com um porto de oito bits para emissão e outro porto de oito bits para a recepção de dados (para evitar reprogramação frequente da interface). O Operador pode ser interrompido pelo PC e também pelo Nó associado. O Operador no entanto, só pode sinalizar através de "flag", tanto para o Nó como para o PC, a existência de alguma comunicação a ser iniciada.

CAPÍTULO III

O SISTEMA DE MULTIPROCESSAMENTO CLEPSIDRA (SMC)

III.1 - INTRODUÇÃO

A idéia de particionar um programa, ou seja, segmentá-lo em procedimentos e dados pertinentes, denominados Tarefas, que possam ser executadas concorrentemente, é bastante conveniente para a otimização do uso dos recursos de um sistema de computação. Um sistema que suporta a execução de um conjuntos destas Tarefas é denominado Multi-tarefas.

Essa capacidade de execução concorrente, proporcionada por um sistema Multi-tarefas, implementada num sistema de computação Múltiplos Processadores é conhecido como Multiprocessamento. Um programa para ser executado em Multiprocessamento, deve ser dividido em um certo número de Tarefas, para as quais se identifica uma certa ordem de execução, sequencial entre algumas, paralela entre outras.

O SMC tem por objetivo implementar um sistema Multi-tarefas no computador Clepsidra de arquitetura não convencional, aproveitando apenas os recursos dele atualmente disponíveis.

III.2 - CONCEPÇÃO DO SMC

O desenvolvimento de Sistemas Operacionais utilizando o conceito de estrutura hierárquica, discutido em DIJKSTRA (16) e depois em MADNICK (17), produz Sistemas Operacionais bem estruturados.

Em arquitetura Múltiplos Processadores a simplicidade de identificação de suas partes e das funções que estas devem realizar, facilita a identificação dos módulos que compõem o Sistema Operacional.

Como primeiro passo para a identificação dos módulos que compõem o SMC, ele foi dividido em níveis, aderentes à arquitetura do computador Clepsidra.

A arquitetura do computador Clepsidra apresenta, nitidamente, uma relação de mestre X escravo, como visto na seção (II.3), no conjunto de seus processadores.

Aproveitando esta relação, o SMC provê um controle hierarquizado, dividido em dois níveis, um para o escalonamento e o controle do processamento de Tarefas, e o outro para a execução das Tarefas.

Ainda, dentro dos módulos é mantida a estruturação hierárquica, que permite modularidade, de modo a tornar fácil a inclusão, retirada ou alteração de seus procedimentos. Isto permite modificações amenas no SMC, quando, por exemplo, da agregação de mais Operadores ou de outras características.

O SMC, naturalmente, provê também um protocolo de comunicação entre os diversos módulos de programas que o compõem, através de envio e recepção de comandos assíncronos, os quais representam os pedidos de atividades a serem executadas.

III.3 - ESTRUTURA DO SMC

Os dois níveis hierárquicos se identificam pelas duas partes básicas do computador Clepsidra definidas em CINTRA (20), isto é, o Controle de Comunicação Externa e Interna, identificado como Gerente na seção (II.3.2), e a Malha de Processamento.

Assim, a função de mestre no SMC é exercida pelo Gerente (primeiro nível), que é constituída por dois módulos. O primeiro módulo, denominado Sistema Operacional Gerente (SOG), reside no Gerente. O segundo módulo, denominado Monitor de Processador de Comunicação (MPC), reside no Processador de Comunicação. Neste nível se concentra a maior parte do código do SMC.

O SOG é responsável pelo controle da comunicação externa, e o MPC pelo controle de comunicação interna, e ambos pela supervisão de execução das Tarefas desenvolvidas nos Operadores da Malha de Processamento.

A função de escravo no SMC é exercida pela Malha de Processamento (MP) (segundo nível), constituída de dois módulos. O primeiro módulo

é o Núcleo do Operador (NOp), que é o responsável pelo atendimento local à demanda proveniente da execução da Tarefa alocada. Em cada Operador existe uma cópia idêntica do NOp. O segundo módulo, denominado Rotina de Comunicação do Nó (RCNó), é o responsável pelo controle da comunicação entre Operadores. Em cada Nó existe uma cópia idêntica da RCNó.

A figura (III.1) mostra a localização dos quatro módulos do SMC na arquitetura Clepsidra implementada.

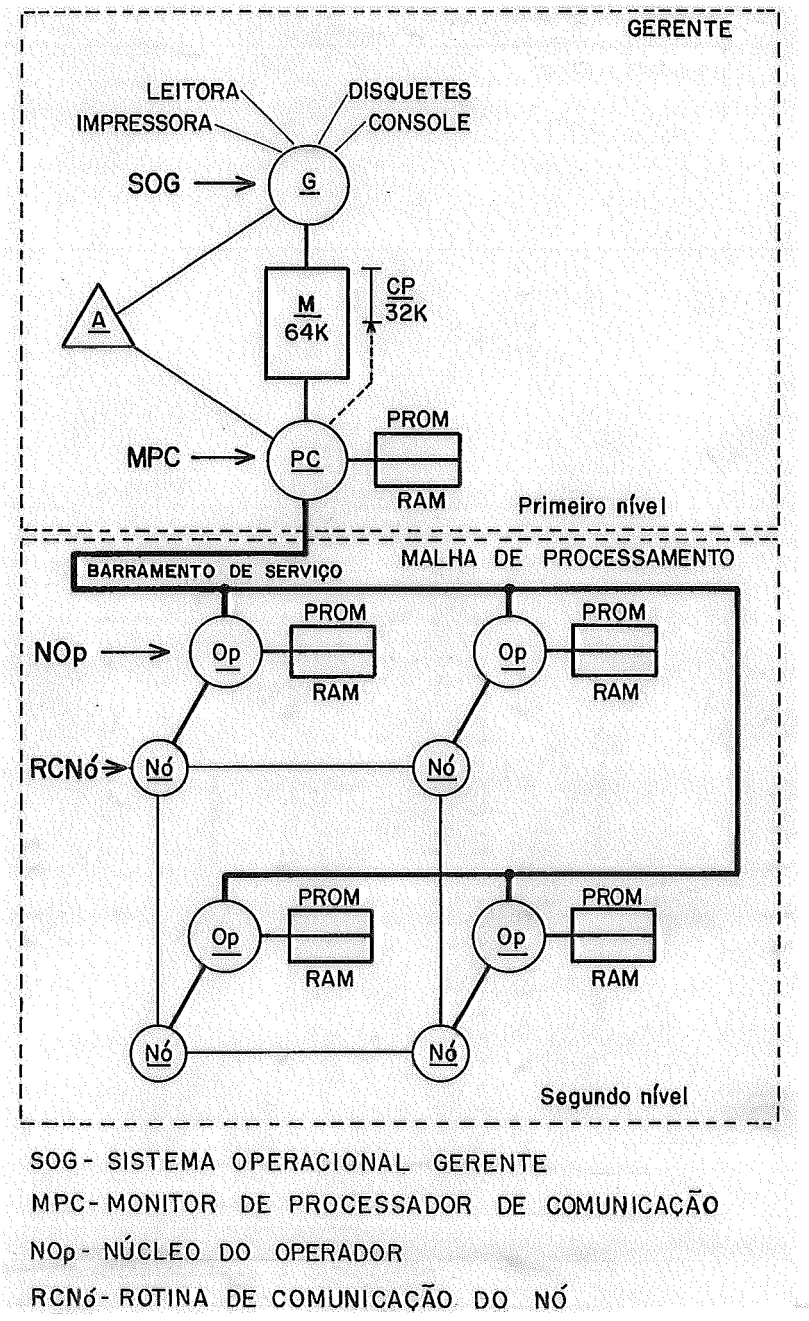


Figura III.1 - Localização dos módulos do SMC na arquitetura Clepsidra.

Dessa forma, os quatro módulos que compõem o SMC utilizam as características inerentes à arquitetura Clepsidra (mestre X escravo), aproveitando-as naturalmente para a sua implementação. Estes quatro módulos possibilitam um ambiente para que Tarefas possam ser executadas em paralelo e cooperem entre si. Assim, estes módulos devem prover: inicialização e carregamento de Tarefas, alocação de recursos para as Tarefas, detecção e recuperação de falhas.

III.3.1 - PRIMEIRO NÍVEL

Como já visto, o Primeiro Nível é composto de dois módulos: o Sistema Operacional Gerente e o Monitor do Processador de Comunicação.

a) Sistema Operacional Gerente (SOG)

A finalidade do SOG é controlar a comunicação com o meio externo, isto é, controlar a console, a impressora, os discos flexíveis, e também gerenciar as informações de controle de execução das Tarefas. Para tanto, o SOG interage com o Monitor do Processador de Comunicação (MPC), enviando-lhe e recebendo códigos e dados, através de Caixa Postal (depósito e retirada de informações), conforme mostrado na figura (III.2).

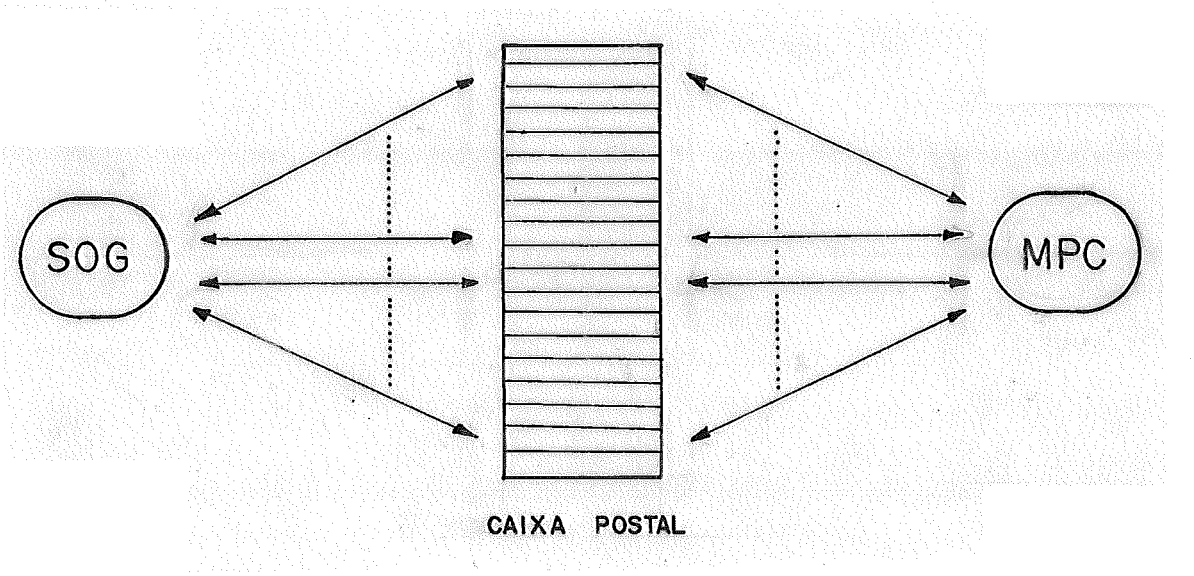


Figura III.2 - Estrutura da Caixa Postal.

A demanda originada no ambiente da Malha de Processamento (MP), em função da execução das Tarefas, é atendida pelo SOG. Conforme o pedido, o SOG designa um de seus procedimentos, para realizar o atendimento. Este procedimento vai em seguida requerer os recursos do Gerente para realizar a sua função, os quais são controlados por um escalonador.

Cada um desses procedimentos possui uma função específica, um não interfere diretamente no outro, mas todos cooperam pelo suporte ao controle de processamento e a alocação de Tarefas nos Operadores.

De uma forma mais explícita, esses procedimentos gerenciam os Programas, isto é, as Tarefas que os constituem. Este gerenciamento vai desde a entrada das Tarefas no SMC, o escalonamento e o carregamento delas na MP, o atendimento da demanda delas durante a execução, o tratamento à identificação e recuperação de falhas, até a impressão dos resultados.

A execução do algoritmo de escolha de programa a ser executado e, naturalmente, da escolha de quais Tarefas a serem carregadas nos Operadores, constitue-se no escalonamento a longo prazo. Os escalonamentos a médio e a curto prazo não existem, pois a Tarefa uma vez carregada só deixa o Operador quando termina ou quando há falha de execução.

b) Monitor do Processador de Comunicação (MPC)

O MPC tem por finalidade controlar a comunicação com os Operadores. Para isto, atende os pedidos das Tarefas alocadas nos Operadores, depositando-os na Caixa Postal. Estas informações são retiradas pelo SOG para o atendimento ser feito por um de seus procedimentos, permitindo a continuidade de execução das Tarefas.

O MPC é constituído, basicamente, por dois procedimentos, os quais são executados de forma alternada. O primeiro, verifica pedidos de interrupções vindos dos Operadores através de linhas específicas do "hardware". O MPC comunica com um NOp por vez. Após o início da comunicação, ela não é interrompida enquanto não tiverem sido trocadas todas as mensagens previstas para aquela comunicação. O segundo verifica pedidos vindos do SOG através da varredura da Caixa Postal. Em ambos os casos, o pedido é atendido logo que detectado.

O MPC exerce também uma função de auxiliar do SOG, procurando aliviar a carga de trabalho deste último. Ele filtra os pedidos vindos dos NOps ou do SOG, verificando a consistência deles. Para isto, o MPC mantém uma tabela que contém informações do estado de execução dos Operadores, através da qual ele supervisiona o processamento ocorrido neles. Assim, o MPC pode interferir nos casos de inconsistência, como por exemplo, quando o NOp pede que seja enviado dados ao SOG, sendo que o estado corrente do NOp é de entrada de dados e não de saída de dados.

Quando detectado esse tipo de inconsistência, visto como falha de execução de Tarefa (detecção de falha), o MPC interrompe a execução das Tarefas do programa, avisando ao SOG do fato ocorrido.

III.3.2 - SEGUNDO NÍVEL

Da mesma forma que no Primeiro Nível, o Segundo Nível é composto por dois módulos: Núcleo do Operador (NOp) e Rotina de Comunicação do Nó (RCNÓ), os quais são constituídos de procedimentos, sempre estruturados de forma a permitir mudanças amenas.

a) Núcleo do Operador (NOp)

O NOp é constituído, basicamente, dos procedimentos que requisitam e aceitam interrupção do MPC e do RCNÓ, que realizam a alocação e retirada de Tarefas no Operador, que atendem as necessidades de execução da Tarefa alocada, ou seja, comunicação com o SOG ou com as outras Tarefas, e que fazem o acompanhamento da execução da Tarefa alocada. Este acompanhamento é feito à toda comunicação, produzida ou requisitada, verificando a consistência dos códigos que fluem durante a comunicação com o estado do Operador e da Malha de Processamento.

b) Rotina de Comunicação do Nó (RCNÓ)

A função da RCNÓ é de encaminhar informações provenientes, ou do NOp ou de outra RCNÓ vizinha, na direção e sentido do destino especificado pelo endereço delas.

Nem sempre, o caminho tomado é o mais curto para o destino da mensagem, pois a melhor via pode estar ocupada. Quando isto ocorre, a RCNÓ procura encaminhar a mensagem por uma direção e sentido que mais se aproximem dos originais.

A RCNÓ possue duas fases de trabalho, a fase de recepção/roteamento de mensagem e a de transmissão de mensagem, que se alternam (seção II.3.3.1).

III.4 - O SISTEMA MULTI-TAREFAS IMPLEMENTADO PELO SMC

A implementação de um sistema Multi-tarefas, em arquitetura Múltiplos Processadores, deve aproveitar as características desta arquitetura. No caso da arquitetura Clepsidra, aquelas descritas na seção(II.3).

Para tanto, nesta seção é descrito como foi equacionado o problema de distribuição dos programas a serem executados nos Operadores, indo desde a decomposição de programas em Tarefas, a inicialização e o escalonamento delas para a execução, a alocação delas nos Operadores, a alocação de recursos físicos ou lógicos (periféricos e dados) para atendimento das necessidades ocorridas durante a execução delas, e também a detecção e a recuperação de falhas.

Essas atividades, em geral, implicam na cooperação de diversos módulos do SMC.

A inicialização para a execução está a cargo do SOG. A alocação é decidida pelo SOG e auxiliada pelo MPC e o NOp. O pedido de alocação de recursos, físicos ou lógicos, feito pelas Tarefas é atendido sempre sob o controle do SOG, depois de encaminhado pelo NOp e MPC. Para alguns casos de recursos lógicos, o atendimento é feito pelo NOp, depois de encaminhado pela RCNÓ. E a detecção de falhas e o posterior tratamento delas são feitas a nível do NOp, MPC e SOG.

III.4.1 - DECOMPOSIÇÃO DE PROGRAMAS

Durante a execução de um número de Tarefas, pertencentes ao mesmo programa, existe a possibilidade de interdependência de dados, devi

do à necessidade de comunicação e sincronização entre elas. Isto torna necessário estabelecer uma ordem de execução, onde procura-se obter o maior grau de paralelismo.

Por isso, é preciso uma fase de detecção de paralelismo, que não é automática, ficando a cargo do programador, que pode ser feita através do uso de Redes de Petri.

O modelamento por Redes de Petri permite decompor um algoritmo em partições que possam ser executadas concorrentemente, apresentando os detalhes de relacionamento entre estas partições. Desta forma, havendo necessidade, pode-se redimensionar as Tarefas, verificar a ordem de execução delas, e identificar quais podem ser executadas simultaneamente.

O relacionamento entre as Tarefas, no SMC, ocorre somente por dependência de informações, e não por dependência de tempo, pois as Tarefas não são capazes de lidar com a variável tempo. Quanto à dependência de dados, uma Tarefa pode operar totalmente independente, denominada independente, ou de forma relacionada, denominada relacionada, através de cooperação com outras Tarefas.

A literatura sobre Redes de Petri é bastante extensa quanto às técnicas de decomposição, como pode ser visto em, entre outros, PETERSON (21), TOULOTTE et alii (22). Neste trabalho, não se pretende entrar em detalhes sobre esta técnica de modelagem de sistemas, apenas mostra-se uma aplicação no Capítulo V.

Uma característica importante no Sistema Multi-tarefas é a troca de mensagens, forma pela qual se realiza o envio e a recepção de informações entre as Tarefas. Esta troca de mensagens é realizada por Primitivas de Comunicação e Sincronização, descritas na seção (III.4.1.2).

Assim, um programa estando a nível de algoritmo ou modelado por qualquer técnica de modelamento, para poder ser executado sob o SMC passa pelas seguintes fases, executadas manualmente por um programador:

1. É modelado em Redes de Petri;
2. São identificadas as partes que podem ser executadas concorrentemente;
3. As partições são codificadas em linguagem executável pelos Operadores, tornando-se pequenos programas;
4. São introduzidas nestes pequenos programas as Primitivas de Comunicação e Sincronização necessárias. A partir deste ponto, eles tornam-se Tarefas.

Quando as Tarefas pertencem ao mesmo Programa, estas são denominadas de Tarefas Irmãs.

III.4.1.1 - ÁREA DE TRABALHO DE TAREFAS

Em cada Operador, existe uma área reservada de memória para uso da Tarefa alocada. O tamanho e a localização desta área, denominada Área de Trabalho da Tarefa, é de prévio conhecimento do programador.

A Área de Trabalho da Tarefa é dividida em duas regiões: Área de Alocação e Área de Execução, mostradas na figura (III.3).

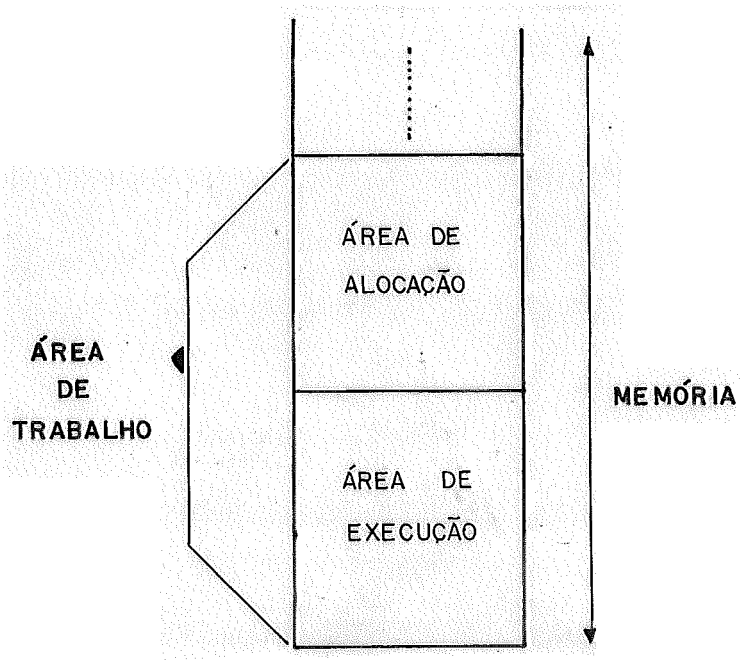


Figura III.3 - Área de Trabalho da Tarefa

A área de Alocação corresponde a uma região da memória RAM de tamanho fixo, localizada logo após o Núcleo do Operador e indo até o início da Área de Execução. Esta última, subsequente a Área de Alocação, vai até o fim da memória RAM disponível.

III.4.1.2 - PRIMITIVAS DE COMUNICAÇÃO E SINCRONIZAÇÃO DE TAREFAS

O SMC oferece Primitivas de Comunicação e Sincronização para a comunicação entre Tarefas/Tarefas e Tarefas/SOG. Estas Primitivas proporcionam a programação de Tarefas de forma estruturada, e também permitem que haja universalidade para a execução delas, isto é, qualquer que seja o Operador alocado, este provê o mesmo suporte de comunicação, tornando transparente para o programador a alocação delas.

O uso destas Primitivas unifica o tratamento da comunicação entre Tarefas e a entrada e saida de dados delas.

A execução das Tarefas ocorre nos Operadores da Malha de Processamento, as quais comunicam entre si através da rede "crossbar", controlada pelos Nós associados aos Operadores. Devido a esta característica da arquitetura Clepsidra, foi adotada a técnica de troca de mensagens para a comunicação entre Tarefas.

Na construção das Primitivas de Comunicação e Sincronização, procurou-se manter a simplicidade, para não sobrecarregar a comunicação na Malha de Processamento. Portanto, foram adotadas três primitivas, suficientes para o tipo de comunicação desejada entre as Tarefas:

Envia (itf, itd); Recebe (itf);

Recebecondicional (itf, param).

Onde os parâmetros significam: itf - identificador tarefa fonte itd - identificador tarefa destino e param - "flag" de teste que indica a presença de mensagem. A Primitiva Envia corresponde ao envio de uma mensagem. As Primitivas Recebe e Recebecondicional correspondem ao pedido de uma mensagem.

O tamanho de uma mensagem é fixo (71 "bytes", correspondendo a 64 "bytes" de dados úteis e 7 "bytes" de controle. Os 64 "bytes" de dados úteis residem na pilha do Operador, cujo fim é dado por um

apontador de pilha e o início é o valor do apontador menos 63.

Dessa forma, a toda mensagem que a Tarefa deseja enviar, ela deve, primeiro, preencher a pilha com os dados, em seguida invocar a primitiva. Esta primitiva, em função dos seus parâmetros, monta a mensagem e a introduz no retículo.

Para a recepção de mensagem, a Tarefa invoca a primitiva Recebe, a qual verifica se a mensagem requerida já está presente. Uma vez estando presente, a mensagem é copiada pelo Núcleo do Operador de um arquivo de entrada dele para a pilha de Área de Execução da Tarefa, cujo início é o endereço apontado pelo apontador de pilha.

Assim, o programador deve prever na lógica da Tarefa, a cada entrada e saída de mensagens, o tratamento a ser feito antes da invocação da primitiva Envia e o tratamento a ser feito após a invocação da primitiva Recebe.

O fato de se ter optado por suportar troca de mensagens de tamanho fixo, levou a uma diminuição da flexibilidade de programação da Tarefa, mas em contrapartida, simplificou a implementação das primitivas pelo SMC.

Para a confecção de mensagens, duas questões são imediatas, segundo GENTLEMAN (23): quanto à sintaxe e quanto à semântica.

a) Sintaxe da mensagem

A mensagem possue tamanho único, sendo caracterizada por: endereço do Operador em que a Tarefa destino está alocada (ver seção IV.3.1.1), ordem de sua produção, identificações das Tarefas destino e fonte, identificação do programa fonte, conjunto de dados úteis e a informação da paridade da mensagem, mostrada na figura (III.4).

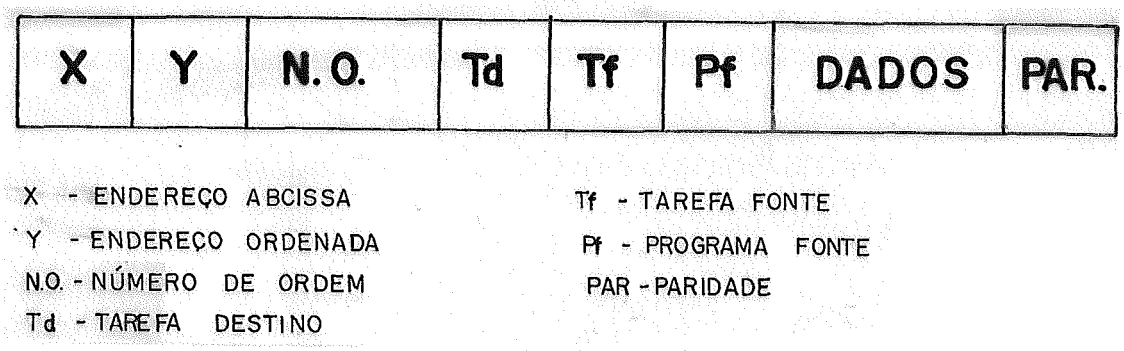


Figura III.4 - Campos de Mensagem

O motivo de se prover um campo da mensagem para conter a ordem de produção dela é devido à característica de operação da Malha de Processamento. Na Malha de Processamento, quando uma mensagem não consegue a melhor via para chegar ao destino por estar ocupada, esta é desviada por outra via disponível. Isto pode fazer com que, quando houver duas mensagens para o mesmo destino, produzidas em instantes de tempos distintos, a segunda possa chegar antes da primeira.

Nesse caso, se o Núcleo do Operador fornecer à Tarefa a mensagem inapropriada, pode comprometer a execução das Tarefas envolvidas. Portanto, urge a necessidade de se evitar esta possível falha. Assim, o Núcleo do Operador enumera as mensagens produzidas e os pedidos de mensagens feitas, para que possam ser confrontados os números de ordem e consequentemente atribuída à Tarefa a mensagem apropriada. Para esta enumeração, os Núcleos dos Operadores mantém seis contadores, três para a produção, um para cada Operador destino, atualizados a cada execução da Primitiva Envia, e os outros três para a recepção, também um para cada Operador emissor, atualizados a cada execução da Primitiva Recebe. O Núcleo do Operador identifica qual Operador que emitiu a mensagem, através da confrontação da identidade da Tarefa expedidora da mensagem (Tarefa fonte) com o estado de alocação da Malha de Processamento, dado pela tabela (IV.2).

b) Semântica da mensagem

Em relação à semântica, três aspectos precisam ser abordados: sincronismo, canais de comunicação e tratamento de falhas.

Quanto ao sincronismo, as primitivas podem ser síncronas (blo

queadas) ou assíncronas (não bloqueadas). Uma primitiva é considerada síncrona se ela causa um bloqueio na execução da Tarefa que a contém. Isto ocorre quando a primitiva não pode ser executada integralmente devido à falta de alguma condição. Uma primitiva é considerada assíncrona em caso contrário.

Para a escolha de qualquer dos dois tipos de primitivas, deve-se levar em conta os seguintes pontos:

- o ideal, em sistemas com troca de mensagens, é utilizar primitivas não bloqueadas, que permitem um maior grau de paralelismo. Por outro lado, esta é uma alternativa bastante complexa para se implementar;

- o outro extremo, consiste em primitivas bloqueadas, que tiram parte do paralelismo, tornando sequenciais as execuções das Tarefas durante a comunicação.

Para o SMC, adotou-se para a primitiva de recepção de mensagens dois tipos: bloqueado e não-bloqueado; e para o envio optou-se por bloquear a Tarefa após o envio de n mensagens para uma determinada Tarefa Irmã.

Para o recebimento de mensagens, o bloqueamento é até certo ponto natural, pois geralmente até o recebimento da mensagem, a Tarefa não tem nada a fazer. Além disso, a programação da Tarefa desta forma para o recebimento de mensagens, é mais simples devido ao recebimento da mensagem poder ser tratado de forma determinística. Pois de outra forma, tornaria mais complexa, tanto o tratamento da recepção da mensagem, bem como a lógica de programação da Tarefa. Porém, existe situação em que o não recebimento da mensagem não afeta o curso normal de uma execução. Neste caso, para a primitiva de recepção, o caso não bloqueado (condicional) vem satisfazer esta situação.

Para o envio de mensagens, o limite de até um número n é uma forma de permitir um certo grau de paralelismo entre as Tarefas, onde n é o tamanho do arquivo do Operador para recebimento de mensagens de um outro Operador qualquer. Em cada Operador existem três arquivos, sendo que cada um é utilizado para armazenar mensagens de um determinado Opera

dor da Malha de Processamento. Assim, o Núcleo do Operador emissor bloqueia a Tarefa a cada n mensagens enviadas para uma mesma Tarefa, desbloqueando-a quando receber do Núcleo do Operador hospedeiro da Tarefa destino um sinal de que pode liberá-la. Isso ocorre quando a Tarefa destino tenha consumido pelo menos n/2 mensagens do arquivo do Núcleo do Operador emissor.

Com relação aos canais de comunicação, o SMC não provê a especificação deles por parte da Tarefa, pois a comunicação na Malha de Processamento não permite determinar qual via será tomada, devido ao caracter totalmente aleatório de comunicação existente na Malha de Processamento.

Quanto ao tratamento de falhas, o SMC para toda mensagem recebida realiza uma verificação da validade da mensagem. Esta verificação consiste em, realizar o teste de paridade, após receber 71 "bytes" (este número é fixo para toda mensagem).

III.4.2 - INICIALIZAÇÃO DE TAREFAS

A cada nova Tarefa, é criada uma estrutura de dados, denominada Bloco Descritor de Tarefa, para o acompanhamento de sua execução, a qual desaparece no seu término. Este acompanhamento vai desde a entrada da Tarefa no SMC até a impressão dos dados produzidos por ela.

No escalonamento dos programas, a cada Tarefa é preenchido o respectivo Bloco Descritor de Tarefa, com as informações: o seu programa de origem, a sua identificação e a sua prioridade quanto à alocação, em relação às Tarefas Irmãs.

Além disso, é feita uma verificação do tamanho da Tarefa confrontando-o com o tamanho da Área de Alocação, caso ultrapasse, a Tarefa e suas irmãs são declaradas inválidas. Este fato é relatado através de impressão, indicando as Tarefas inválidas e o motivo, sendo retiradas do SMC.

III.4.3 - CRITÉRIO DE ALOCAÇÃO DE TAREFAS

A partir do modelamento do programa por Redes de Petri, é estabelecido a precedência e simultaneidade de execução entre Tarefas Irmãs, chamada de macroordenação.

O SOG tem em disco o código de cada uma das Tarefas, seus respectivos dados iniciais, quando for o caso, bem como os grafos de sequenciamento (um para cada programa), determinado pelo programador, a partir da macroordenação, expressada pelas prioridades das Tarefas. A precedência é indicada por valores decrescentes de prioridades e a simultaneidade por valores iguais.

O SOG inicia a alocação dos programas de acordo com a ordem de armazenamento deles, através das suas Tarefas de maior prioridade, até a ocupação de todos os Operadores disponíveis, usando a Tabela de Estado dos Operadores, mostrada na seção (IV.2.1).

A medida em que as Tarefas vão terminando, elas são substituídas por outras do mesmo programa, de acordo com a prioridade delas. Isto até que todas as Tarefas do programa terminem. Existindo Operador livre, sem que haja mais Tarefas dos programas já em execução para serem alocadas, inicia-se a execução de um novo programa, através da alocação de suas Tarefas mais prioritárias.

A partir do início da execução das Tarefas, todo o fluxo do processamento está sob a iniciativa delas, mas sob o controle do SMC, atendendo a demanda delas.

III.4.4 - ATENDIMENTO AOS PEDIDOS DE RECURSOS FEITOS PELAS TAREFAS

Durante a execução das Tarefas, elas podem gerar mensagens, pedindo dados ou fornecendo resultados.

O tratamento dessa demanda consiste em atender a necessidade de fornecer mais dados para as Tarefas continuarem o processamento (Primitiva Recebe), bem como de enviar dados à alguma Tarefa Irmã ou para a impressão de resultados (Primitiva Envia).

No caso de pedir dados, dois procedimentos podem ser originados: ou o pedido é para o Gerente, que vai buscá-lo no disco, ou é para outra Tarefa Irmã. No último caso, a Tarefa pode estar alocada ou não. Quando alocada, o Núcleo do Operador verifica se já recebeu aquela mensagem. Se recebeu, repassa-a à Tarefa, se não recebeu, bloqueia a Tarefa, espera pela mensagem específica, para depois entregá-la à Tarefa, retomando a execução dela. Quando não alocada, a Tarefa requisitante ficará bloqueada até a Tarefa que irá produzir a mensagem ser alocada e gerar a mensagem requisitada. Para enviar dados, o tratamento da primitiva Envia segue aquele descrito na seção (III.4.1.2). O SMC não provê multiprogramação a nível de Operador durante a entrada e saída de dados.

Além desses, o atendimento é feito também quando uma Tarefa termina o seu processamento de modo normal ou anormal.

III.4.5 - DETECÇÃO DE FALHAS

A arquitetura Clepsidra não provê nenhum mecanismo de detecção de falhas explicitamente. Assim, toda a detecção de falhas é feita pelo SMC. Para isto, o SMC aproveita das posições estratégicas dos seus módulos, isto é, intermediários na comunicação entre Tarefas/Tarefas e Tarefas/SOG, exercidas pelo Monitor do Processador de Comunicação e pelos Núcleos dos Operadores. O Monitor do Processador de Comunicação possue a capacidade de verificar a consistência dos comandos que por ele fluem, e caso detecte inconsistência, interrompe o fluxo e avisa o Sistema Operacional Gerente da falha, cabendo a este último a decisão da atitude a ser tomada.

Outra forma de detecção, é feita a nível do Núcleo do Operador. Este trabalha de forma previsível, correspondendo a um autômato. Portanto, a partir de um estado de processamento, o NOp possue os próximos estados já estabelecidos, e caso haja alguma influência para que mude a um estado não previsto, isto é imediatamente detectado pelo Núcleo do Operador, o qual descontinua a execução da Tarefa, avisando ao Sistema Operacional Gerente o fato ocorrido.

Além desses mecanismos de detecção, também os pedidos por parte de Tarefas que não possam ser atendidos pelo SOG, como por exemplo,

pedido de mais dados, no caso inexistentes, é considerado falha de execução de Tarefas.

III.4.6 - RECUPERAÇÃO DE FALHAS

Em sistemas que possuem mecanismos de tolerância a falhas, após a falha ter sido detectada, a recuperação dela é naturalmente ativada.

Para a recuperação de falha, como discutido em ANDERSON et LEE (24), duas técnicas podem ser aplicadas. Uma delas procura retornar a um estado anterior de execução confiável, denominada "Backward Error Recovery". A outra procura, através do estado atual da execução, produzir um novo estado livre de falha e é denominada "Forward Error Recovery".

O SMC não possue condições para prover nenhuma dessas técnicas. Pois, para a técnica "Backward Error Recovery" é necessário haver um mecanismo que interrompa a execução de tempo em tempo para o armazenamento de estados intermediários da execução. Isto permite, quando for detectada uma falha e após a análise dela, retomar a execução a partir de um destes estados de execução intermediários, considerado naturalmente o mais apropriado. Este mecanismo, o computador Clepsidra não provê (relógio de tempo real).

A técnica "Forward Error Recovery" necessita de um conhecimento prévio do comportamento dos programas a serem executados. Como o SMC visa suportar programas genéricos, isto torna impossível a implementação desta técnia a nível do SMC. No entanto, o programador pode vir a implementar esta técnica, a nível de Tarefas, ficando totalmente a cargo delas a recuperação, não influenciando o SMC.

Portanto, a recuperação de falhas no SMC consiste num tratamento de exceção simples, ou seja, o critério adotado é simplesmente de descontinuar a Tarefa em que foi detectada a falha, bem como da(s) sua(s) Tarefas(s) Irmã(s), retirando o programa do SMC, sendo relatado este fato e a sua causa, via impressão.

CAPÍTULO IV

ESPECIFICAÇÃO DO SISTEMA DE MULTIPROCESSAMENTO CLEPSIDRA (SMC)

IV.1 - INTRODUÇÃO

A implantação do SMC observou as considerações feitas no Capítulo III, procurando especificá-las dentro das condições e estruturas físicas oferecidas pela arquitetura original do computador Clepsidra.

Dentre as condições tecidas no Capítulo III, a estrutura hierárquica tornou-se a característica mais importante, pois esta permitiu à confecção do SMC simples. Respeitando esta estrutura hierárquica, todo o controle de alocação de Tarefas aos Operadores e o posterior atendimento de suas necessidades, durante as suas execuções, ficou no primeiro nível, e o controle direto das execuções delas ficou no segundo nível.

As seções seguintes descrevem o fluxo das Tarefas desde a entrada destas no sistema até as suas execuções, incluindo a forma de relacionamento entre elas. Esta descrição é feita de acordo com os dois níveis do SMC, através dos módulos que os implantam.

IV.2 - PRIMEIRO NÍVEL

O primeiro nível é constituído fisicamente por dois microprocessadores, os quais possuem, já na arquitetura original, a qualidade de mestre, sendo os responsáveis pelo controle da execução de Tarefas na Malha de Processamento. No primeiro microprocessador, chamado Gerente, reside o módulo Sistema Operacional Gerente. No segundo microprocessador, chamado Processador de Comunicação, reside o módulo Monitor do Processador de Comunicação. Estes módulos são descritos nas duas seções seguintes.

IV.2.1 - SISTEMA OPERACIONAL GERENTE (SOG)

O SOG tem por objetivo controlar a execução e dar suporte à demanda ocorrida durante a execução das Tarefas, suprindo as necessidades de entrada e saída delas.

O SOG é um programa escrito em linguagem de programação Pascal sequencial, que é executado usando a p-máquina do Sistema Pascal U. C. S. D. II.0, descrito em UCSD (25). A maior parte do SMC está compreendida no SOG, sendo composto de Procedimentos e do Escalonador de Procedimentos, mostrados nas duas seções seguintes. Após é mostrado, através da figura (IV.4), como ocorre a evolução da execução de Tarefas a nível do SOG.

IV.2.1.1 - ESCALONADOR

O Escalonador de Procedimentos do SOG é constituído de um algoritmo de decisão para o controle da execução deles. Além disto, o Escalonador é quem faz a verificação da consistência dos pedidos provenientes da Malha de Processamento, via Monitor do Processador de Comunicação.

Para tanto, o Escalonador mantém a tabela (IV.1), que indica o estado de execução dos Núcleos dos Operadores (NOps), com respeito à interação entre o SOG e os NOps. Por exemplo, esta tabela mostra, num dado instante, que os Operadores 1 e 2 estão ocupados com as Tarefas 1 e 2, ambas do programa 10, o Operador 3 está ocupado com a Tarefa 3 do programa 9 e o Operador 4 encontra-se disponível. Esta tabela é atualizada a cada alocação/retirada de Tarefa.

Operador	1	2	3	4
programa	10	10	9	ø
Tarefa	1	2	3	ø
estado	1	1	4	ø

Tabela IV.1 - Tabela do estado da Malha de Processamento.

Para cada Procedimento do SOG, existe um Bloco Descritor de Procedimento (BDP), que é composto das informações mostradas abaixo:

```
CONST
        max  priori procedimento = 3;
```

```
TYPE
      apontador bdp = ↑bdp;
      bdp = RECORD
            nome procedimento: (inicializar, carrega tarefa, entrada da
                                dos, saída dados, término, rolamento, im
                                pressão);
            prioridade: 1.. max priori procedimento;
            estado: (inativo, executando, pronto, suspenso);
            prox bdp: apontador bdp
           END;
```

O controle da execução dos Procedimentos é feito através de alocação dos BDP em estados, os quais representam a situação de execução para cada Procedimento, cujo diagrama é mostrado na figura (IV.1).

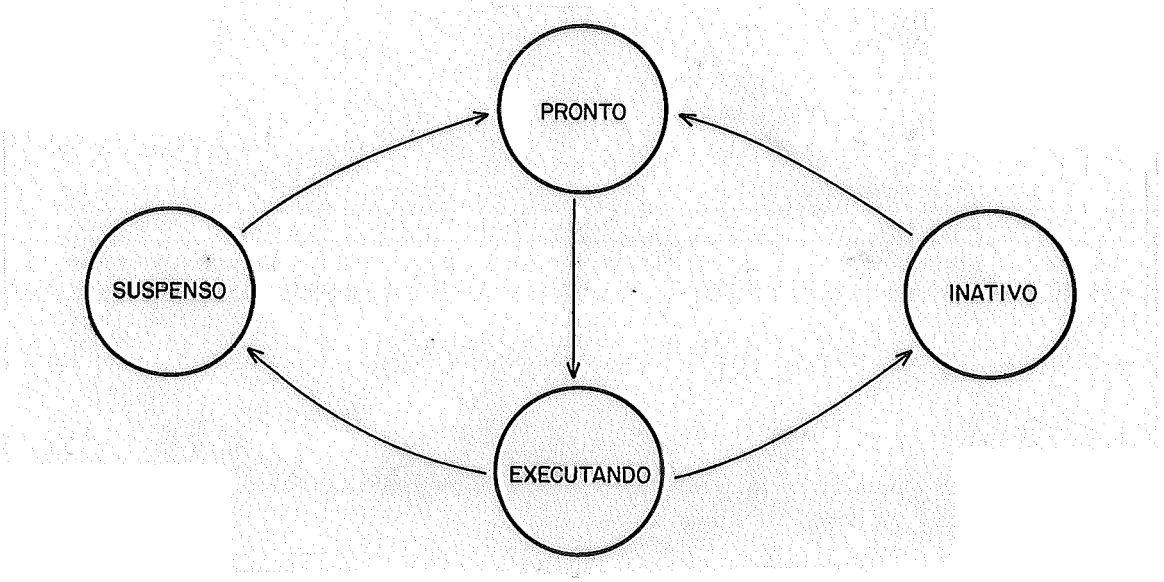


Figura IV.1 - Diagrama de estados dos Procedimentos.

Esses estados são:

PRONTO - O Procedimento pode iniciar a execução;

EXECUTANDO - O Procedimento está sendo executado, podendo requerer qualquer recurso físico do Gerente;

SUSPENSO - O Procedimento teve a sua execução interrompida devido à

falta de algum recurso (físico ou lógico);

INATIVO - A execução do Procedimento não está sendo exigida.

Esses estados são organizados sob uma estrutura de dados do tipo fila, como é mostrado abaixo:

```
TYPE
     filas procedimentos = (inativo, suspenso, pronto, executando);
VAR
     topo filas procedimentos: ARRAY [filas procedimentos]
                                   OF apontador bdp;
```

O algoritmo de decisão, para o escalonamento de Procedimentos, corresponde à constante verificação de um conjunto de requisitos que devem ser satisfeitos, tornando possível aos Procedimentos serem comutados entre esses estados. Para serem considerados aptos à execução, ou seja, prontos, os Procedimentos devem satisfazer as condições mostradas na Tabela IV.2. A indicação de BDT em filas, na coluna "Condições necessárias", é descrita na seção seguinte, e pode ser vista na figura (IV.4).

Procedimentos:	Condições necessárias:
Inicializar	existe Tarefa e existe Bloco Descritor de Tarefa disponível
Carrega Tarefa	Tarefa pronta (BDT na fila 1) e Operador disponível
Entrada Dados	Tarefa requerendo mais dados (BDT na fila 3)
Saída Dados	Tarefa requerendo o armazenamento dados produzidos (BDT na fila 4)
Término	Tarefa executada (BDT na fila 5)
Rolamento	Tarefa com dados para serem impressos

	(BDT na fila 6)
Impressão	Tarefa com dados já preparados para serem impressos (BDT na fila 7)

Tabela IV.2 - Condições que tornam os Procedimentos aptos à execução.

Assim, sempre que a execução de um Procedimento para, ou por término normal, é colocado na fila de inativo, ou por falta de algum recurso (físico ou lógico), é colocado na fila de suspenso, o controle da execução retorna ao Escalonador. Este refaz o escalonamento em função das condições correntes (figura IV.2), principalmente da demanda proveniente do processamento ocorrido na Malha de Processamento, e depois coloca em execução o Procedimento mais prioritário. Esta figura mostra a facilidade de inclusão de Procedimentos, por exemplo, basta prever mais uma fila de BDT (descrito na seção seguinte), caracterizando a condição necessária para a ativação do Procedimento novo (Procedimento i).

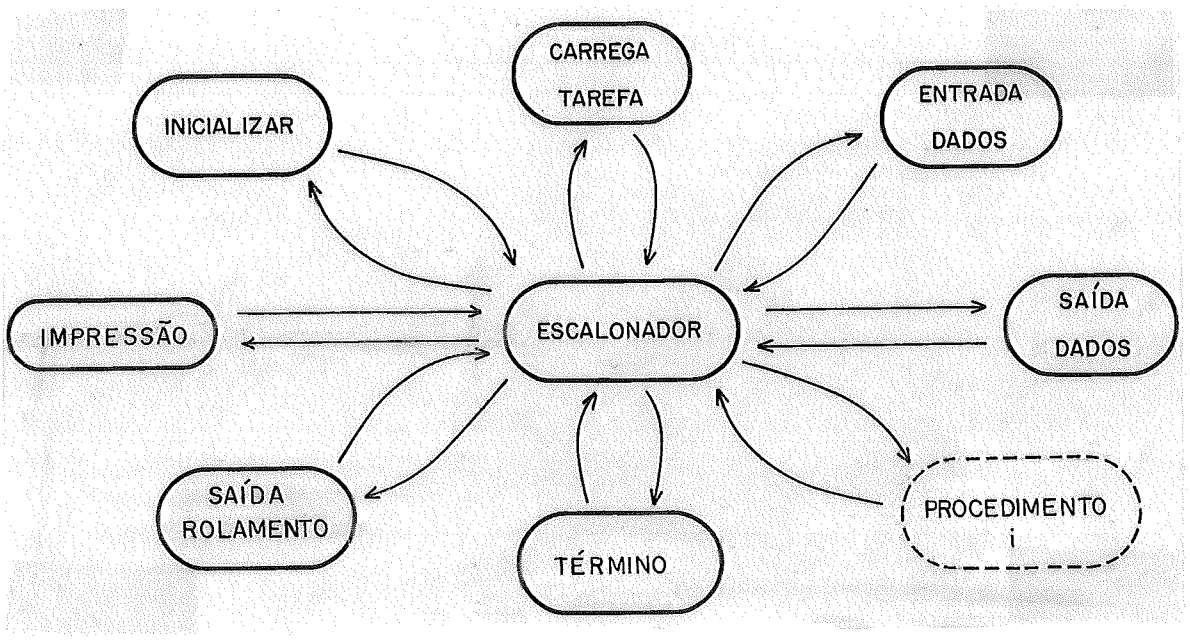


Figura IV.2 - Escalonador dos Procedimentos do SOG.

IV.2.1.2 - PROCEDIMENTOS DO SOG

O controle e o suporte, para a execução de Tarefas, são feitos através de uma estrutura que permite acompanhar todas as situações em

que elas podem passar, isto é, Inicializar, Carrega Tarefa, Entrada Dados, Saída Dados, Término, Rolamento (SPOOL) e Impressão. Estas situações caracterizam as atividades a serem realizadas para cada Tarefa. Elas dão nomes aos procedimentos que realizam estas atividades.

Para cada situação, que representa o estado em que a Tarefa pode estar a nível do SOG, existe um tratamento adequado que é feito por procedimento específico do SOG. Este tratamento é feito utilizando um conjunto de dados referentes à Tarefa, denominado Bloco Descritor da Tarefa (BDT).

A cada Tarefa corresponde um BDT, no qual são colocadas informações oriundas do cabeçalho dela, que constituem-se nas mostradas abaixo:

```
TYPE
      apontador bdt: ↑bdt;
      bdt = RECORD
              nome prog;
              nome tar: STRING [2];
              nro priori: STRING [1];
              estado: (executando, entrada dados, saída dados, término);
              erro,
              arq dado aberto,
              arq impr aberto,
              arq tar aberto: BOOLEAN;
              código erro,
              operador,
              nro caixa postal: INTEGER;
              prox bdt: apontador bdt
            END;
```

O BDT é criado pelo Procedimento Inicializar, sendo manipulado pelos Procedimentos Carrega Tarefa, Entrada Dados, Saída Dados, Término, Rolamento e Impressão, sendo que, neste último o BDT é tornado disponível para a próxima Tarefa. Assim uma Tarefa passa por vários estados, conforme é mostrado no diagrama da figura (IV.3). Os estados que possuem dois círculos, como executando, entrada dados, saída dados e término, são aque

les pelo qual a Tarefa também passa a nível dos módulos MPC e NOp. As mudanças de estado de execução de Tarefas possíveis, a nível do SOG, são acompanhadas pelo Escalonador, permitindo desta forma ao SOG detectar tentativas de mudanças de estados indevidas.

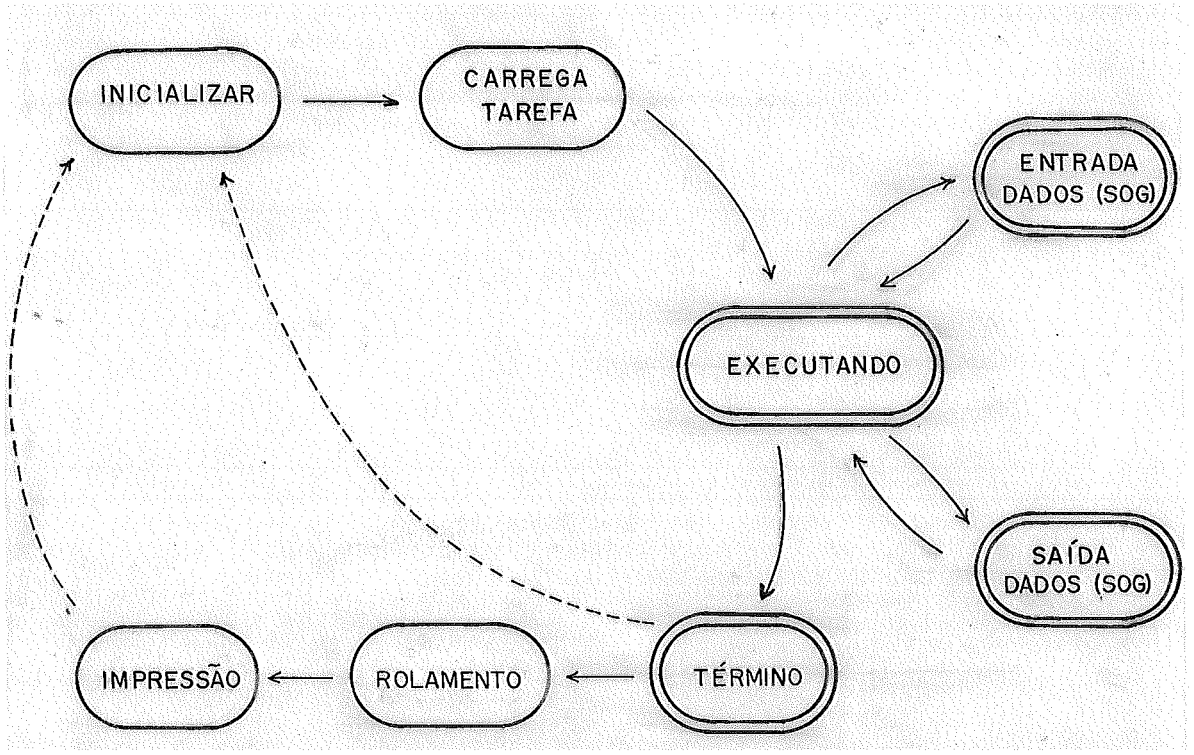


Figura IV.3 - Diagrama de estados de Tarefas, a nível do SOG.

Essa manipulação é organizada sob uma estrutura de dados tipo fila, como mostrado abaixo:

```
CONST
      max fila bdt = 8;
```

```
VAR
      topo fila bdt: ARRAY [1.. max fila bdt]
                     OF apontador bdt;
```

Para cada Procedimento do SOG existe uma fila de BDTs e a alocação de um BDT numa determinada fila de um Procedimento, consiste na forma de indicar a necessidade daquele BDT ser tratado por aquele Procedimento.

Após o tratamento de um BDT por um Procedimento, este BDT é alocado à fila de outro Procedimento conforme a necessidade do BDT.

A descrição dos tratamentos é feita a seguir de acordo com a ordem estabelecida pelo SMC de suporte à execução das Tarefas, visando o atendimento durante à execução até o término dela, com a posterior impressão dos resultados. Esta descrição está dividida em três partes, sendo feita por Procedimentos, na ordem natural da evolução da execução de uma Tarefa.

a) Procedimento referente à inicialização e preparação de execução de Tarefas (prioridade de execução dois)

. Procedimento Inicializar

Para cada programa, são verificadas as características das respectivas Tarefas, constituindo-se nas informações que são colocadas nos BDTs, isto é, o Procedimento Inicializar tem por objetivo criar um BDT a cada Tarefa nova, preenchendo os campos dele, observando o sequenciamento indicado no cabeçalho de cada Tarefa.

Este Procedimento verifica se o tamanho da Tarefa é maior do que o permitido (maior do que a Área de Alocação), caso seja, esta Tarefa e suas Tarefas Irmãs são retiradas do SMC, sendo relatado este fato por meio de impressão.

Com a criação do BDT a Tarefa está pronta para ser executada, sendo colocada na fila de Tarefas prontas, cujo Procedimento encarregado de tratar é o Carrega Tarefa. Após, o Procedimento Inicializar é tornado

inativo, permanecendo à espera do próximo escalonamento.

b) Procedimentos encarregados de interagir com a Malha de Processamento (prioridade de execução um)

Esses Procedimentos utilizam para a comunicação com o Processador de Comunicação uma região de memória compartilhada, denominada Caixa Postal (CxP), que é descrita no Apêndice A.

. Procedimento Carrega Tarefa

O Procedimento Carrega Tarefa possui a incumbência de, uma vez existindo Operador disponível, carregar as instruções da Tarefa mais prioritária no Operador disponível. Cada Operador corresponde a uma única Caixa Postal. Assim, quando uma Tarefa está sendo carregada através da Caixa Postal i, significa que ela é para ser executada no Operador i.

Para tanto, este Procedimento utiliza rotinas programadas em "assembler" para poder acessar a CxP referente ao Operador disponível, que são: ESCR CCP (nro CCP, ni) e ESCR CxP INFO (inic CxP, fim CxP, nro linhas).

Após um primeiro carregamento (preenchimento de uma CxP), o Procedimento Carrega Tarefa verifica se carregou ou não todas as instruções da Tarefa, se não carregou, o Procedimento Carrega Tarefa é suspenso automaticamente, esperando ser ativado no próximo escalonamento. Esta interrupção da execução dele se faz necessária para aumentar o paralelismo do SMC, pois durante este intervalo de tempo o Monitor do Processador de Comunicação estará carregando o código no Núcleo do Operador, podendo, enquanto isso o SOG atender outras necessidades do processamento. Assim, o carregamento é feito passo a passo.

Esse Procedimento atualiza a Tabela do Estado da Malha de Processamento, marcando o Operador ocupado como não disponível.

Após o Procedimento Carrega Tarefa terminar de carregar a Tarefa e ter ativado a execução da Tarefa no Operador, este é tornado inativo. Sendo que o BDT fica alocado à fila de Tarefas em execução até ocorrer

alguma necessidade de entrada ou saída de dados, ou término normal ou anormal de execução. No caso de entrada e saída de dados, o BDT vai para a fila específica até ser atendida a necessidade, retornando após o atendimento para a fila de Tarefas em execução, ocorrendo o mesmo para a situação de término.

. Procedimento Entrada Dados

O Procedimento Entrada Dados atende à necessidade da Tarefa de ter mais dados para continuar a sua execução, enviando-lhe novos dados.

Esse Procedimento utiliza duas rotinas programadas em "assembler" para preencher a CxP referente a Tarefa com mais dados, que são ESCR CxP INFO (inic CxP, fim CxP, nro linhas) e ESCR CCP (nro CCP, ed).

Após o carregamento de novos dados na CxP específica, o BDT respectivo retorna à fila de Tarefas em execução, esperando pelo próximo evento. Quando o MPC pede dados e não há mais dados para serem carregados, este Procedimento informa ao MPC do ocorrido, que por sua vez descontinua a Tarefa, bem como as suas Tarefas Irmãs, além de introduzir no BDT o código de erro (51 - inexistência de dados). Em seguida a uma dessas atividades, esse Procedimento é tornado inativo.

. Procedimento Saída Dados

O Procedimento Saída Dados atende à necessidade de saída de dados produzidos (resultados) durante a execução da Tarefa, armazenando-os em disco, para posterior impressão, quando do término de execução dela.

Para retirar os dados da CxP, esse Procedimento utiliza duas rotinas programadas em "assembler": LE CxP INFO (inic CxP, fim CxP) e ESCR CCP (nro CCP, disp).

Após o armazenamento dos dados produzidos pela Tarefa em área reservada do disco desta Tarefa, o BDT respectivo retorna à fila de Tarefas em execução, esperando pelo próximo evento. Em seguida o Procedimento Saída Dados vai para o estado dos procedimentos inativos.

. Procedimento Término

Uma vez um BDT estando na fila de Tarefas terminadas, o Procedimento Término verifica se aquela Tarefa terminou de forma normal ou anormal. No caso de ter terminada normalmente, o Procedimento Término verifica se a Tarefa produziu dados, em caso de ter produzido, o BDT é alocado à fila de Tarefas para Rolamento (SPOOL), caso contrário o BDT é tornado disponível, sendo alocado na fila de BDTs disponíveis (Procedimento Inicializar), para posterior alocação e consequente preenchimento de dados úteis à execução referentes à nova Tarefa. Este Procedimento atualiza a Tabela do Estado da Malha de Processamento, marcando a disponibilidade de Operador. Após, o Procedimento Término vai para o estado dos procedimentos inativos.

c) Procedimentos referentes à saída do programa do SMC (prioridade de execução três)

Os Procedimentos Rolamento e Impressão utilizam um conjunto de "buffers", cuja estrutura de manipulação é descrita no Apêndice C.

. Procedimento Rolamento (SPOOL)

O Procedimento Rolamento tem por finalidade a leitura de dados produzidos por Tarefas, que estão armazenados em área do disco destinada à impressão, colocando-os em "buffers" numa forma final para a impressão, isto é, todas as informações necessárias para identificar de qual programa e Tarefa pertencem os dados impressos e os resultados.

O preenchimento de "buffers" é feito enquanto houver "buffer" disponível ou terminar o conjunto de dados para impressão. Se faltar "buffer", o Procedimento Rolamento é suspenso. Se terminar o conjunto de dados, ele é tornado inativo. No caso de se colocar todos os dados produzidos em "buffers", isto significa que está pronto para ser impresso, portanto o BDT é alocado à fila de Tarefas para impressão.

. Procedimento Impressão

O Procedimento Impressão é encarregado de imprimir o cabeçalho da impressão, que contém indicações de qual programa e Tarefa perten

cem os dados a serem impressos, os dados produzidos e o final da impressão, indicando o término desta impressão.

Se houve erro de execução, esta situação é detectada e impresso o código do erro ocorrido. Podendo ocorrer os seguintes erros:

51 - dados inexistentes para a Tarefa i (detectado pelo SOG, marcando a Tarefa i com o código de erro)

52 - Tarefa i querendo se comunicar com Tarefa já executada (detectado pelo NOp, marcando a Tarefa i com o código de erro)

Os erros seguintes dizem respeito aos pedidos não aceitos, em função do estado de execução da Tarefa. Estes erros podem ser detectados a nível de três módulos: SOG, MPC e NOp. Os pedidos que são aceitos são mostrados na tabela (IV.3). Estes erros são descritos abaixo, sendo caracterizados em função do pedido não aceito:

53 - novas instruções não aceitas

54 - últimas instruções não aceitas

55 - descontinua Tarefa não aceita

56 - entrada dados não aceita

O código de erro é transmitido para o SOG, acompanhado do estado corrente da Tarefa e o módulo em que foi detectado. Vale notar que pode existir uma defasagem de estado corrente entre os módulos, além dos módulos possuirem alguns estados de Tarefa em comum e outros estados não.

Após o término da impressão, o BDT é tornado disponível, sendo colocado na fila de BDTs disponíveis para posterior alocação e preenchimento de dados referentes à nova Tarefa para ser executada (Procedimento Inicializar).

IV.2.1.3 - SEQUÊNCIA DE EXECUÇÃO DE TAREFAS

Para ilustrar a sequência de execução de Tarefas, a figura (IV.4) mostra desde a entrada de Tarefa no SMC até a impressão dos resultados por ela produzidos. Esta sequência de atividades é realizada pelos diversos Procedimentos descritos na seção (IV.2.1.2).

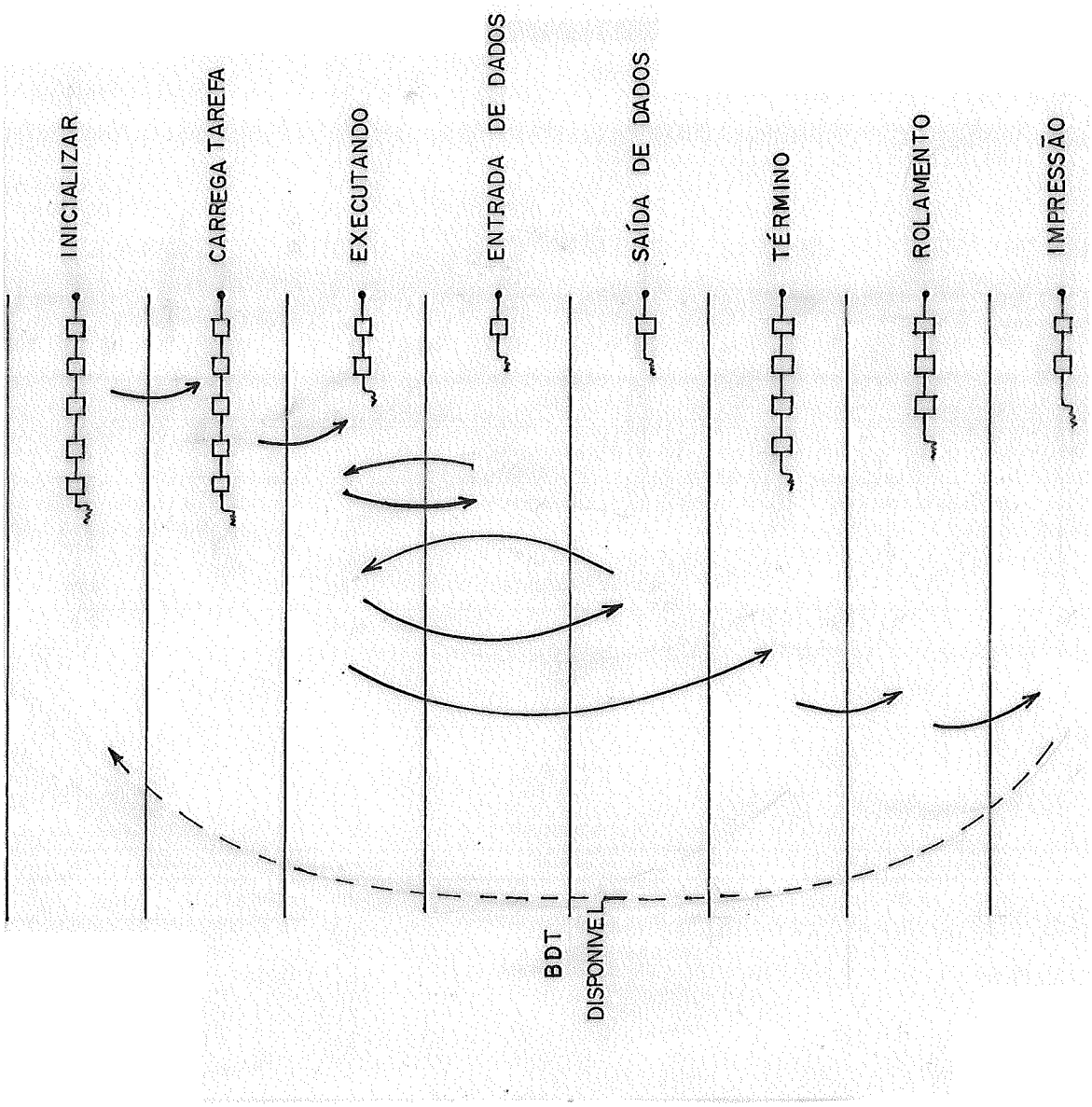


Figura IV.4 - Sequência de execução de Tarefas.

IV.2.2 - MONITOR DO PROCESSADOR DE COMUNICAÇÃO (MPC)

O MPC tem por objetivo realizar a comunicação com os Núcleos dos Operadores (NOp), enviando-lhes instruções de Tarefas e dados e recebendo dados produzidos por elas. Desta forma, o MPC supervisiona diretamente a execução das Tarefas. As informações enviadas aos Operadores provêm doSOG, e as recebidas são encaminhadas ao SOG. O MPC se comunica com o SOG através de Caixas Postais - CxP, explicado no Apêndice A, e com os NOps através do Barramento de Serviço, descrito no Apêndice B.

O MPC foi programado usando a linguagem de programação MPL - Motorola Programming Language, descrita em MPL (26).

Para realizar essas atividades de comunicação, o MPC realiza duas atividades, as quais codificadas consistem nos procedimentos Verifica Pedido de Interrupção (VPI) dos Operadores e Atende Pedido das Caixas Postais (APCxP). Cada um destes procedimentos interage com o respectivo meio de comunicação externo: Barramento de Serviço e Caixas Postais. Estes procedimentos são executados sequencialmente de forma alternada.

Além disso, o MPC por ser um intermediário na comunicação entre as Tarefas e o SOG pode servir de verificador da execução das Tarefas.

Para tanto, o MPC mantém uma tabela idêntica à tabela (IV.1), que indica o estado de processamento dos Núcleos dos Operadores, com respeito à interação entre os Núcleos dos Operadores e o MPC.

Assim, através dessa tabela, à todo comando novo, seja proveniente dos Núcleos dos Operadores ou do SOG, o MPC confronta-o com o estado de execução do Núcleo do Operador, confirmando a consistência do atendimento do pedido feito.

Quando é detectada a inconsistência, o MPC gera uma mensagem de erro e a envia ao SOG, cuja interpretação é de descontinuar a execução das Tarefas Irmãs, imprimindo o fato. Os erros possíveis de serem detectados são aqueles descritos na seção (IV.2.1.2).

O MPC faz o acompanhamento da mudança dos estados de execução de Tarefas possíveis através do diagrama de estado mostrado na figura (IV.5), que permite detectar tentativas indevidas de mudanças de estado.

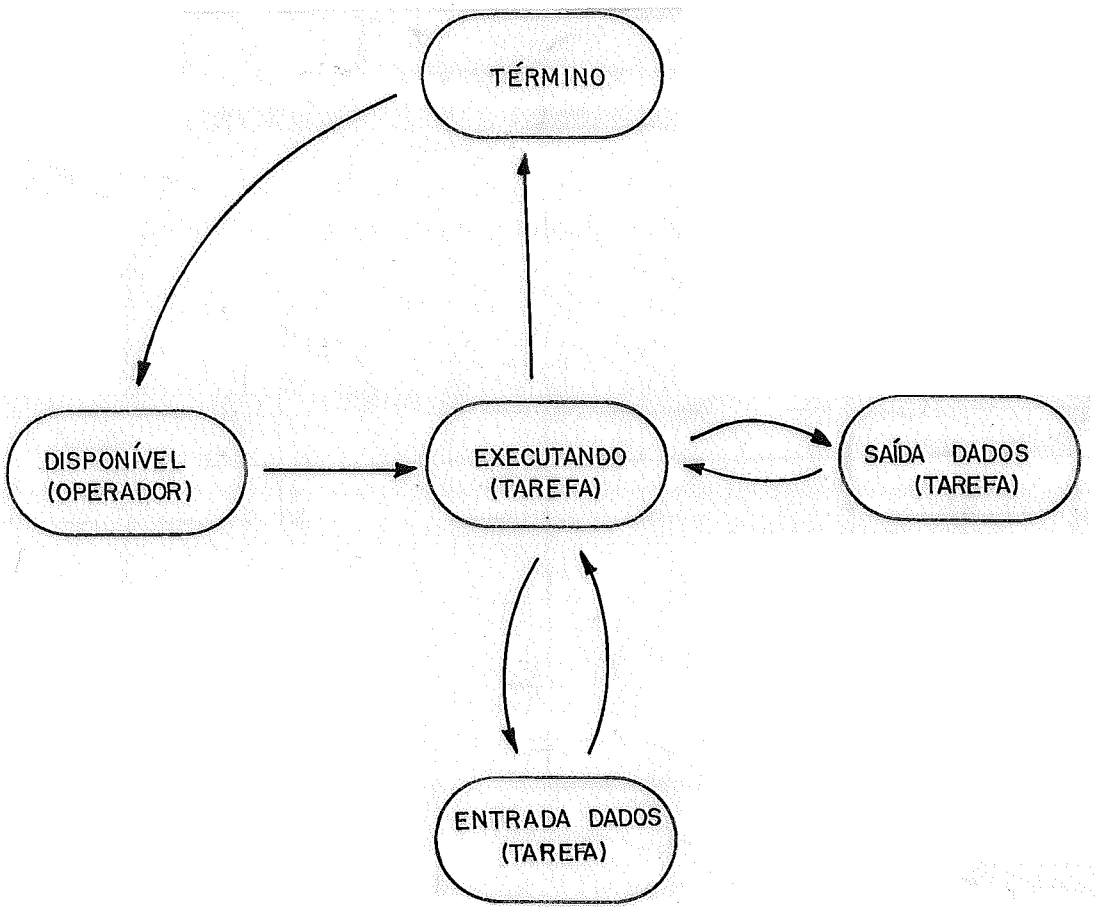


Figura IV.5 - Diagrama de estados de Tarefa, a nível do MPC.

IV.2.2.1 - VERIFICA PEDIDO DE INTERRUPÇÃO (VPI)

O procedimento VPI é encarregado de controlar o Barramento de Serviço, através do qual ele se comunica com os Núcleos dos Operadores. Esta comunicação é feita através de varredura das linhas do Barramento de Serviço, que indicam se há requisição de algum Operador de comunicação. A comunicação via Barramento de Serviço é descrito no Apêndice B.

IV.2.2.2 - ATENDE PEDIDO DA CAIXA POSTAL (APCxP)

A atividade APCxP se caracteriza por observar em sequência os quatro Controle de Caixa Postal - CCxP, de onde provêm o comando de interação com o SOG. Os comandos que o SOG pode depositar indicam os seguintes casos: novas instruções, últimas instruções, descontinua tarefa e entrada dados. O acesso aos CCxP são realizados como é mostrado no Apêndice A.

IV.3 - SEGUNDO NÍVEL

O segundo nível é realizado pela Malha de Processamento, que é composta de microprocessadores Operadores e seus respectivos microprocessadores Nós. Em cada Operador reside uma cópia do Núcleo do Operador, e em cada Nó reside uma cópia da Rotina de Comunicação do Nó.

IV.3.1 - NÚCLEO DO OPERADOR (NOp)

O NOp é o módulo do SMC encarregado de suportar a execução da Tarefa no Operador hospedeiro. Este suporte é feito de maneira a atender os pedidos de entrada e saída da Tarefa, quando em execução, e também fazer a alocação e retirada de Tarefa.

Para este suporte, o NOp mantém uma tabela semelhante à tabela (IV.1). Esta tabela é utilizada para localizar os Operadores hospedeiros das Tarefas destinos, quando do envio ou da recepção de uma mensagem.

O NOp, também, exerce uma função de gerenciamento da execução da Tarefa, podendo detectar erro de execução dela. Para esta atividade, o NOp utiliza a tabela (IV.3) que contém, dependendo do estado corrente, os possíveis motivos de interrupção aceitos (início de comunicação) e o respectivo tratamento, seja proveniente do MPC ou da RCNÓ do Nó associado. Desta forma, o NOp confronta o motivo da interrupção com os possíveis, naturalmente dentro do estado corrente, estabelecendo daí a atitude a ser tomada.

Estado do NOp		Interrupção proveniente de:		Consequência
Código	Interpretação	Código	Interpretação	
Ø	Disponível		- MPC -	
		1	novas instruções da Tarefa	Recebe e aguarda
		2	últimas instruções da Tarefa (ativa a execução)	Recebe e ativa a execução
			- RCNÓ -	
			nenhuma interrupção é aceita	

Estado do NOp		Interrupção proveniente de:		Consequência
Código	Interpretação	Código	Interpretação	
1	Executando a Tarefa	3	- MPC - descontinua a Tarefa	Recebe e realiza o pedido
		5	- RCNÓ - entrada de dados	Recebe e gerencia o pedido
2	Entrada de Dados	3	- MPC - descontinua a Tarefa	Recebe e realiza o pedido
		4	entrada de dados	Recebe e gerencia o pedido
		11	- RCNÓ - entrada de dados	Recebe e gerencia o pedido
3	Saída de Dados	3	- MPC - descontiua a Tarefa	Recebe e realiza o pedido
		11	- RCNÓ - entrada de dados	Recebe e gerencia o pedido
4	Erro Execução		- MPC - nenhuma interrupção é aceita	
			- RCNÓ - nenhuma interrupção é aceita	
5	Término		- MPC - nenhuma interrupção é aceita	
		4	- RCNÓ - entrada de dados	Recebe, identifica a Tarefa fonte e avisa o SOG do erro ocorrido

Tabela IV.3 - Motivos de interrupção aceitos pelo NOp.

O Nop também realiza um outro gerenciamento de execução de Tarefa. Este gerenciamento é o de acompanhar a comutação de estado de execução dela. Quando é feito uma tentativa indevida, o NOp detecta e interrompe a execução, indicando erro (código 57).

Os estados em que a Tarefa pode passar são mostrados na figura (IV.6), cujos arcos indicam o sentido das transições.

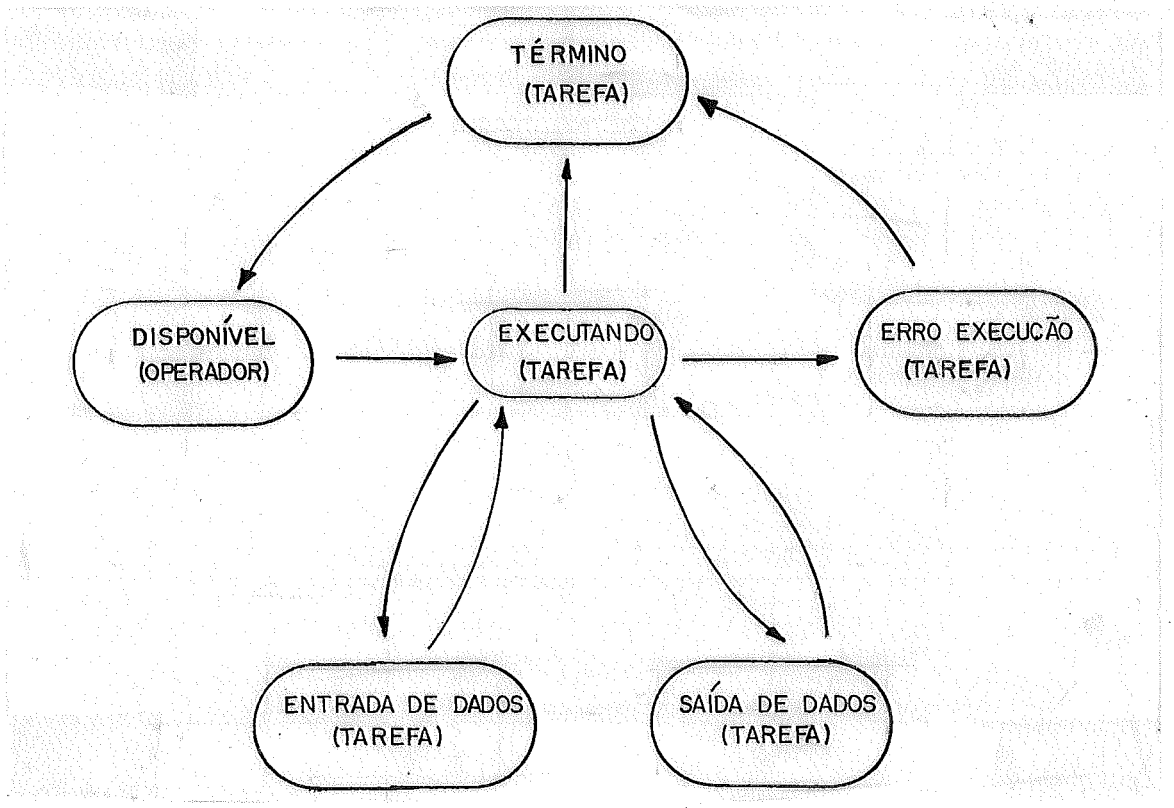


Figura IV.6 - Diagrama de estados da Tarefa, nível do NOp.

Caso o motivo da interrupção não seja nenhum dos previstos, mostrados na Tabela (IV.3), o NOp simplesmente ignora os outros pedidos.

IV.3.1.1 - ATENDE PEDIDO DE COMUNICAÇÃO DAS TAREFAS

A unidade de entrada e saída da Tarefa corresponde a um conjunto de dados, apontado por um apontador de pilha, que é colocado ou retirado da pilha da Área de Execução da Tarefa. Quando ocorre a comunicação, este conjunto de dados mais as informações de controle para transmissão e gerenciamento dela, caracterizam uma mensagem.

Os pedidos de entrada e saída, caracterizados pela invocação das Primitivas de Comunicação e Sincronização, são executados imediatamente pelo NOp, após terem sido invocadas pelas Tarefas. As primitivas Envia, Recebe e Recebecondicional são descritas abaixo:

a) Primitiva Envia

A invocação da Primitiva Envia pela Tarefa ocorre quando esta produziu dados e deseja enviá-los ou a alguma Tarefa Irmã como cooperação ao processamento, ou para o SOG para ser impresso. Isto faz com que a Tarefa passe do estado de executando para o estado saída de dados.

A identificação do destino a ser enviada a mensagem é feita através dos parâmetros contidos na sua invocação, correspondendo a: itf - identificador da Tarefa fonte, itd - identificador da tarefa destino.

A cada produção de uma mensagem, é associada a ela um número que representa a sua ordem de produção. Para toda mensagem enviada para um determinado Operador, um contador de mensagens (nmen) é atualizado. A cada envio de n mensagens, o NOp bloqueia a Tarefa, esperando por uma mensagem de liberação da Tarefa, proveniente daquele Operador, cujo contador de mensagens local é igual a n. À toda chegada de mensagem de liberação, o contador nmen é atualizado para nmen - n/2, estando a Tarefa bloqueada ou não. Se nmen - n/2 é menor do que zero, então nmen é atualizado para zero.

Essa primitiva utiliza dois parâmetros, o primeiro identifica a Tarefa fonte, acumulador A, o segundo a Tarefa destino, acumulador B.

b) Primitiva Recebe

A invocação da Primitiva Recebe, por parte da Tarefa, faz que o NOp, primeiro, passe do estado executando para o estado entrada de dados, depois verifica se já está presente a mensagem requisitada. Para tanto, ele faz uma busca no arquivo de mensagens reservado ao Operador em que deve estar alocada a Tarefa que produz esta mensagem. A localização do Operador é dada através da consulta à Tabela IV.1, que indica o estado de alocação da Malha de Processamento.

Caso a mensagem não esteja armazenada no arquivo de mensagens procurado, o NOp bloqueia a execução da Tarefa hóspede. Isto faz com que o NOp permaneça no estado de entrada de dados até receber a mensagem apropriada, ou quando houver algum motivo para mudança deste estado (falha de execução).

Esses arquivos de mensagens são organizados sob uma estrutura de dados do tipo fila, cujas mensagens neles contidas são identificados por programa e Tarefa, e ordenadas de acordo com o seu número de ordem (produção).

A cada mensagem recebida o NOp verifica o contador do arquivo de mensagem específico, uma vez ele indo além de n/2, o NOp, a partir deste ponto, passa acompanhar o contador de forma mais cuidadosa (ativa um "flag"). Este acompanhamento serve para, uma vez o contador ficar aquém de n/2 e o "flag" estiver ativado, o NOp desativa o "flag" e envia uma mensagem de liberação para a Tarefa, cujo Operador hospedeiro usa aquele arquivo de mensagens. O Operador é localizado através da Tabela do Estado da Malha de Processamento, idêntica à tabela (IV.1).

Como o formato da mensagem não prevê campo para esta indicação, a forma de se contornar é colocando o valor zero no campo que indica a ordem de produção da mensagem. Pois não existe mensagem de ordem zero.

Essa primitiva utiliza um parâmetro que indica a Tarefa fonte, o acumulador A.

c) Primitiva Recebecondicional

A Primitiva Recebecondicional realiza atividade semelhante a Primitiva Recebe, com exeção de que se não houver a mensagem, os contadores não são atualizados e o parâmetro que indica a presença de mensagem volta com valor zero. Se existe a mensagem, o comportamento é idêntico à Primitiva Recebe e o parâmetro volta com o valor um.

Essa primitiva utiliza dois parâmetros, um para indicar a Tarefa fonte, acumulador A, e o outro para passar o endereço em que deve ser colocado a indicação de presença ou não de mensagem, acumulador B.

IV.3.1.2 - ALOCAÇÃO E RETIRADA DE TAREFAS

A alocação ocorre desde a chegada do comando indicando novas instruções da Tarefa (comando 1) até a chegada do comando últimas instruções da Tarefa (comando 2).

Quando a Tarefa termina a sua execução normalmente, isto é, a última instrução executada é Chama-NOp, então o Núcleo do Operador passa para o estado Término e informa esta situação ao MPC, após passa para o estado Disponível.

Outro motivo de se retirar uma Tarefa, neste caso de maneira forçada, é quando ocorre falha de execução, como nos casos descritos na seção (IV.2.1.2),ou quando durante a cooperação entre as Tarefas (troca de mensagens) ocorre um dos casos abaixo descritos. Estes casos são entendidos como erro de execução da Tarefa, fazendo com que o programa seja descontinuado e retirado do SMC.

- chega mensagem para a Tarefa que já não está mais alocada (código 58);

- quer enviar mensagem para Tarefa que não está mais alocada na Malha de Processamento (código 59);

- quer receber mensagem de uma tarefa que não está mais alocada na Malha de Processamento (código 60).

IV.3.2 - ROTINA DE COMUNICAÇÃO DO NÓ (RCNÓ)

A RCNÓ possue duas fases de trabalho. Uma de Recepção/Roteamento da mensagem e outra de Transmissão da mensagem. Em ambas as fases, basicamente, a RCNÓ trabalha utilizando da mensagem somente a informação que contém o endereço. A RCNÓ não foi programada, sendo que, basicamente, o seu algoritmo é descrito abaixo:

- O endereço do destinatário é constituído de dois sub-campos: X e Y. O sub-campo X contém a distância, segundo um eixo X, entre o emissor e o receptor. De modo semelhante, o sub-campo Y contém a distân

cia entre o emissor e o receptor segundo um eixo Y. A distância é contada em número de nós do retículo a serem atravessados. A distância pode ser positiva ou negativa, segundo se vai para a direita ou esquerda ou para cima ou para baixo conforme os eixos X e Y.

- A medida em que as mensagens progridem na malha, uma RCNÓ decrementa o valor absoluto do sub-campo X ou do sub-campo Y do endereço, segundo o eixo em que a mensagem foi enviada, ao sair do Nó. Desta forma, ao atingir o destino, o campo de endereço da mensagem contém X = 0 e Y = 0.

- Por vezes, quando a via ou as vias desejadas pela mensagem estão ocupadas, a RCNÓ a desvia, tomando todavia a providência prévia de modificar o endereço, de modo a refletir este desvio.

As mensagens são enviadas fisicamente "byte" a "byte" através de um barramento unidirecional. Entre dois Nós vizinhos dois barramentos os interligam, um barramento transmite mensagens de um Nó a outro, e o outro barramento transmite mensagens no sentido inverso.

IV.3.2.1 - FASE DE RECEPÇÃO/ROTEAMENTO DE MENSAGENS

Na fase de recepção, a RCNÓ percorre circularmente os barramentos de comunicação com as vizinhas e com o Núcleo do Operador (NOp) associado. Com cada vizinho, ela executa um protocolo de comunicação e em caso de existência de uma mensagem, recebe-a inteira (71 "bytes") antes de passar à vizinha seguinte. No caso do Nop, o procedimento é o mesmo, salvo que apenas um "byte" é recebido a cada Fase de Recepção, em razão da baixa velocidade de processamento do microprocessador Operador comparada à do microprocessador Nó. Deste modo, uma mensagem entrará ou sairá da rede em 71 fases de recepção. Este procedimento permite que a rede trabalhe em alta velocidade de transmissão, que diminui somente nos contatos com os microprocessadores Operadores.

Em seguida, a RCNÓ que está em recepção executa um algoritmo de roteamento. Ela terá então até cinco mensagens armazenadas em memória, que terão de partir pelos cinco barramentos de saída, no próximo ciclo de transmissão. A RCNÓ não faz estocagem temporária de mensagens, pois

isto tornaria o sistema passível de bloqueio perpétuo "deadlock". O algoritmo procura enviar o máximo possível de mensagens em direções e sentidos que as aproximem dos destinos. As que não puderem ser enviadas desta forma são desviadas segundo os barramentos disponíveis. Para tanto, a RCNÓ modifica convenientemente os seus endereços, preservando o destino original.

IV.3.2.2 - FASE DE TRANSMISSÃO

Na fase de transmissão, a RCNÓ percorre circularmente os barramentos de comunicação com as RCNÓs vizinhas e com o NOp associado. Com cada RCNÓ vizinha ele executa um protocolo de comunicação e em caso de existência de uma mensagem, transmite-a inteira antes de passar à vizinha seguinte. De forma análoga à recepção, um único "byte" é enviado ao NOp por Fase de Transmissão, em caso de existência de mensagem a ele destinada.

CAPÍTULO V

EXEMPLO

V.I - INTRODUÇÃO

Neste capítulo é mostrada uma aplicação de um programa a ser executado sob o Sistema de Multiprocessamento Clepsidra (SMC). Neste sentido, é descrito a decomposição de um programa em Tarefas, o escalonamento delas nos Operadores e o relacionamento delas durante a execução, seguidos dos respectivos tratamentos a serem feitos pelo SMC.

Para a decomposição de um programa, procedimento não automático realizado fora do ambiente do SMC, existem técnicas gráficas que possibilitam obter partições de um programa, as quais podem ser executadas paralelamente. Dentre elas, duas técnicas podem ser usadas: Redes de Petri e Grafos de Precedência. Sendo que, o modelamento por Redes de Petri, segundo MACKELVY et AGRAWAL (27), apresenta mais detalhes do relacionamento entre as partições, o que facilita a introdução das Primitivas de Comunicação e Sincronização, caracterizando-as como Tarefas, conforme descrita na seção (III.4.1). Assim, optou-se pelo uso de Redes de Petri para a realização deste exemplo.

A aplicação se refere a um sistema de controle de malha fechada, cujo exemplo foi tirado de BOTTURA (28), número P2.33, transcrito a baixo:

"Na figura abaixo apresentamos um modelo bastante simples de um sistema de controle com realimentação, utilizado por um estudante para controlar suas notas. Este modelo foi sugerido pelo Professor O.J.M. Smith, da Universidade da Califórnia, em Berkeley. Alguns valores típicos dos parâmetros são $k_1 = 1$, $\tau_1 = 1$ mês, $\tau_2 = 0{,}5$ mês. O esforço para eliminar atividades extracurriculares é refletido na constante do ganho k_2, para o qual um estudante, com excesso de autoconfiança, poderia ter $k_2 = 0{,}5$". A figura (V.1) caracteriza o modelo.

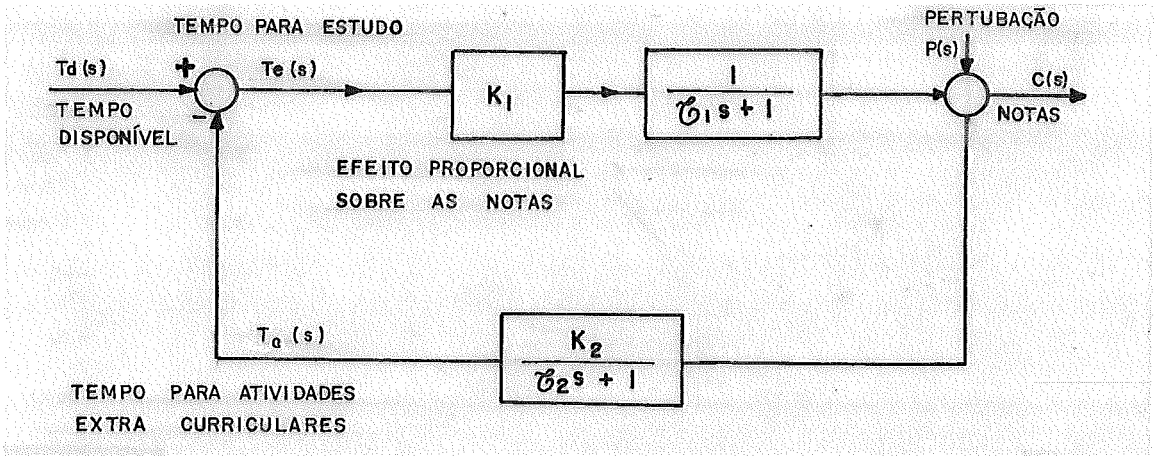


Figura V.1 - Sistema de Controle de Malha Fechada.

Para o desenvolvimento do exemplo, primeiro são invertidas as duas funções transformadas de Laplace, de forma a obter funções no domínio do tempo, devido a amostragem ser feita no domínio do tempo.

$\tau_1 = 1 \qquad k_1 = 1$

$$k_1 \cdot \frac{1}{\tau_1 + s} = \frac{k_1}{1.s + 1} = \frac{1}{s + 1} \quad \Rightarrow \quad \varepsilon^{-at}\ ;\ a = 1 \quad \Rightarrow \quad \varepsilon^{-t}\ //$$

$k_2 = 0{,}5 \qquad \tau_2 = 0{,}5$

$$\frac{k_2}{\tau_2 s + 1} = \frac{1/2}{1/2s + 1} = \frac{1}{s + 2} \quad \Rightarrow \quad \varepsilon^{-at};\ a = 2 \quad \Rightarrow \quad \varepsilon^{-2t}\ //$$

A entrada do modelo Td (disponível) é obtida a partir de um gráfico qualquer. Por exemplo, o da figura (V.2), onde na ordenada tem-se Td (disponível) e na abcissa Tt (transcorrido). Neste gráfico, é suposto o desenvolvimento de atividades por um estudante durante quinze dias. O Td, dado em horas, é amostrado em intervalos de 24 horas. Isto garante, que as variações significativas num período de quinze dias (360 horas) sejam preservadas de forma realística.

Como o interesse aqui é verificar a evolução da execução do cálculo do modelo sob o SMC, considera-se a amostragem feita como válida,

dentro deste ponto de vista.

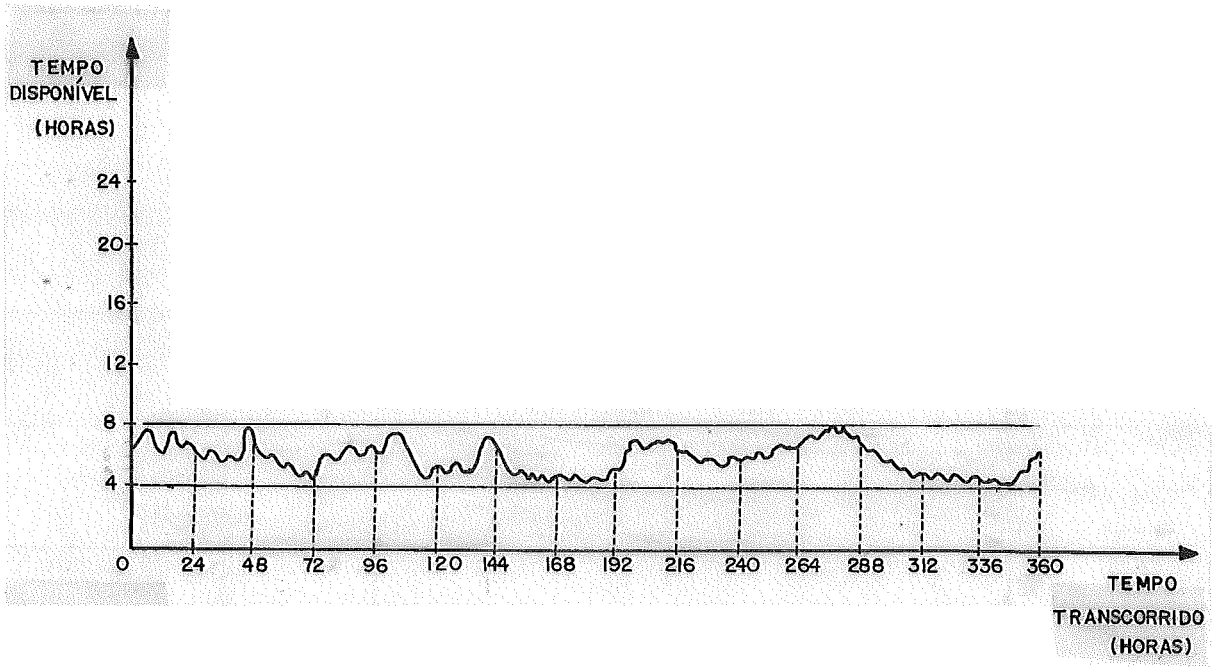


Figura V.2 - Gráfico de tempo disponível X tempo transcorrido.

O tempo Td obtido, a partir do gráfico, é mostrado na tabela (V.1).

t (transcorrido)	Td (s)	t (transcorrido)	Td (s)
24	6	192	5
48	7	216	7
72	4	240	6
96	6	264	7
120	5	288	7
144	7	312	5
168	4	336	4
		360	6

Tabela V.1 - Td (disponível) obtido a partir do gráfico da figura (V.2).

V.2 - DECOMPOSIÇÃO DO MODELO EM TAREFAS

O modelamento em Redes de Petri foi feito manualmente e resultou na rede mostrada na figura (V.3), onde as transições representam as

operações a serem feitas e os lugares representam os dados necessários para que a operação seja realizada. Como o computador Clepsidra possue na atual configuração quatro Operadores, a rede obtida possue quatro transições, de modo a obter um maior grau de paralelismo. Assim, a transição t_1 representa o cálculo do tempo para o estudo, t_2 o cálculo do efeito proporcional sobre as notas, t_3 representa o cálculo das notas e t_4 representa o cálculo do tempo para atividades extra curriculares. A transição t_1 incorpora também um mecanismo que permite reapresentar o valor antigo da realimentação enquanto o valor novo não é calculado por t_4.

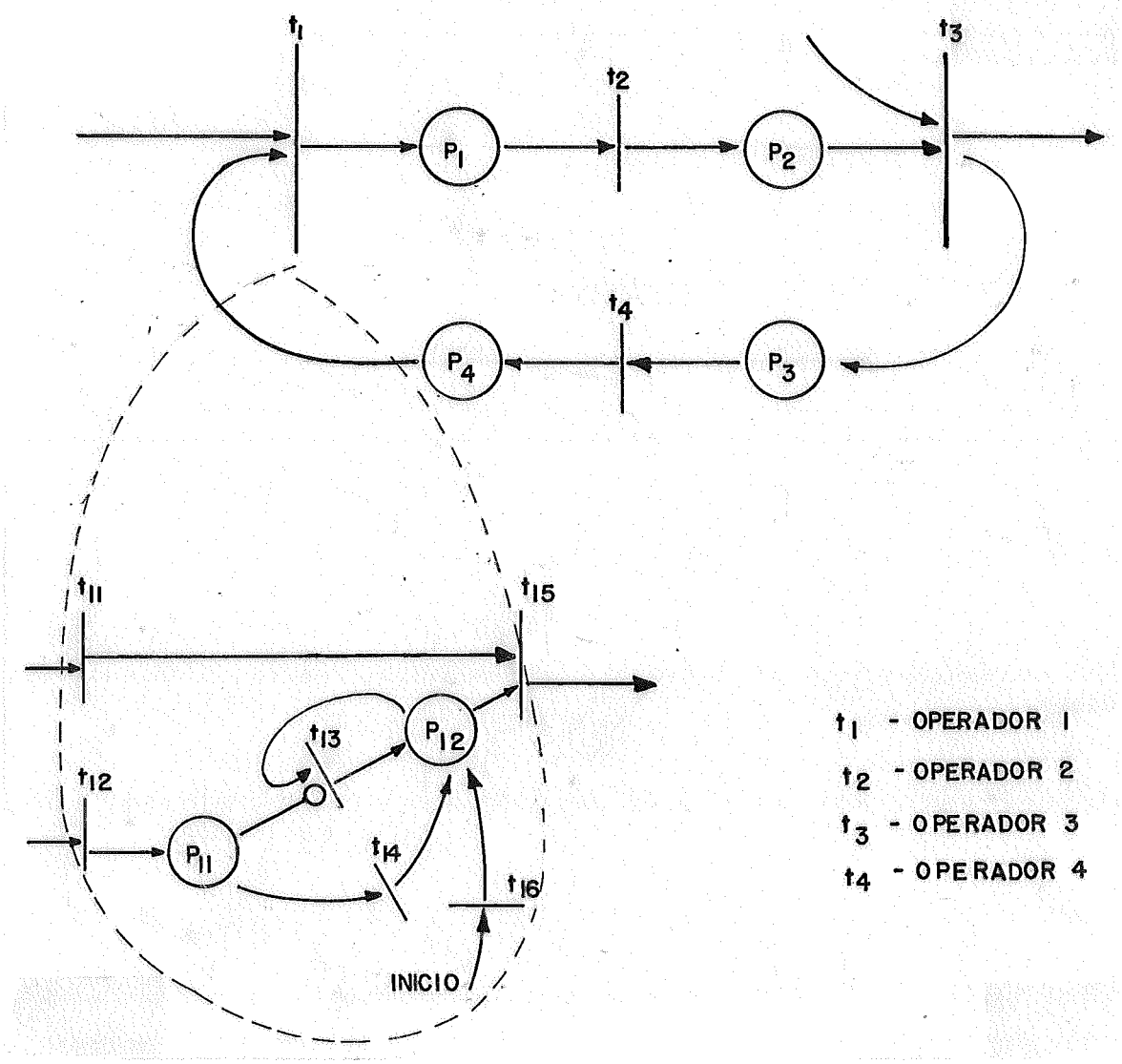


Figura V.3 - Rede de Petri para o sistema de controle de malha fechada.

Após a decomposição, as partições são programadas. A transi

ção t_1 apresenta uma transição de início, que consiste em zerar toda a área de trabalho, pois no primeiro instante do cálculo de t_1, a transição t_4 ainda não enviou resultados. Neste caso, t_1 utiliza os últimos dados vindos de t_4, no início zeros, para tanto, t_1 utiliza a primitiva Recebecondicional, a qual permite, no caso de ter recebido uma mensagem de t_4, usar os novos dados vindos de t_4, caso contrário a transição t_1 usa os dados antigos. A transição t_1 utiliza também a primitiva Recebe para receber dados do Gerente e a primitiva Envia para enviar os resultados à transição t_2. A transição t_1, uma vez programada em "assembler" 6809, descrito em MOTOROLA (29), é inserida nela as Primitivas de Comunicação transformando-se na Tarefa 1, mostrada abaixo:

```
*       Inicializa área de trabalho com zeros
*       Init é uma rotina do NOp
        JSR             INIT
*
*       Inicializa os registros indexadores X e Y para trabalho
*       INIC-AREA-EXEC - início da Área de Execução da Tarefa
*
        LDX             INIC-AREA-EXEC
        LEAY            64,X
        LEAU            64,Y
*
*       Pede dados ao Gerente
*
*       Recebe (G) - Gerente é representado por Ø
*
V1      LDA             #Ø
        JSR             RECEBE
*
*       Pede dados da Tarefa 4, através do Recebecondicional
*       O endereço = 1000 é onde a primitiva vai indicar, através de 1 ou Ø,
*       a presença ou não de mensagem nova. No caso desta Tarefa, não é ne
*       cessário fazer o teste, devido a sua lógica.
*
        LDA             #4
        LDB             =1000
        JSR             Recebecondicional
*
*       Cálculo de t1
*
        LDB             #64
V2      LDA             B,X
        SUBA            B,Y
        PSHA
        DECB
        BNE                 Término de cálculo para 64 dados
        BRA             V2
*
*       Envia dados da Tarefa 1 para a Tarefa 2
```

```
*
        LDA             #4
        LDB             #1
        JSR             ENVIA
        BRA             V1
        END
```

A transição t_2 programada é mostrada abaixo. Ela usa a primitiva Recebe para receber os resultados de t_1 e a primitiva Envia para enviar dados à transição t_3.

```
*
*       Inicializa o registro de índice X e o apontador de pilha U
*
        LDX             INIC-AREA-EXEC
        LEAU            64,X
*
*       Pede dados da Tarefa 1
*       A Tarefa 1 é representada pelo número 1
*
V1      LDA             #1
        JSR             RECEBE
*
*       Cálculo de t2
*
        LDB             #64
V2      LDA             B,X
*
*       Para o cálculo da exponencial, a Tarefa faz uso de uma subrotina do
*       NOp, cujo parâmetro é o registro A, retornando nele o resultado
*
        NEGA
        JSR             EXP
        PSHA
        DECB
        BNE                     Término de cálculo para 64 dados
        BRA             V2
*
*       Envia dados da Tarefa 2 para a Tarefa 3
*
        LDA             #2
        LDB             #3
        JSR             ENVIA
        BRA             V1
        END
```

A transição t_3 usa quatro primitivas, duas primitivas Recebe e duas primitivas Envia. Uma das primitivas Recebe é para receber dados do Gerente, que consiste nos dados de pertubação em degrau unitário e a outra é para receber dados da Tarefa 2, ambas são bloqueadas. Uma primitiva Envia é usada para enviar resultados a serem impressos (Gerente) e a outra

primitiva Envia é usada para enviar dados para a Tarefa 4. O programa da Tarefa 3 é mostrado abaixo:

```
*
*       Inicializa os registros de índice X e Y
*
        LDX             INIC-AREA-EXEC
        LEAY            64,X
*
*       Pede dados do Gerente - Ø
*
V1      LDA             =# Ø
        JSR             RECEBE
*
*       Pede dados da Tarefa 2
*
        LDA             =# 2
        JSR             RECEBE
*
*       Cálculo de t3
*
        LDB             =# 64
V2      LDA             B,X
        ADDA            B,Y
        PSHA
        DECB
        BNE
        BRA             V2      Término de cálculo para 64 dados
*
*       Envia resultados para o Gerente Ø
*
        LDA             =# 3
        LDB             =# Ø
        JSR             ENVIA
*
*       Envia dados para a Tarefa 4
*
        LEAU            64,Y
        LDA             =# 3
        LDB             =# 4
        JSR             ENVIA
        BRA             V1
        END
```

A transição t_4 recebe dados de t_3 e envia resultados para t_1. Para isto, ela usa uma primitiva Recebe, para receber dados da Tarefa 3 e uma primitiva Envia para enviar resultados para a Tarefa 4. O programa da Tarefa 4 é mostrado abaixo:

```
*
*       Inicializa o registro de índice X e o apontador de pilha U
*
        LDX             INIC-AREA-EXEC
        LEAU            64,X
*
*       Pede dados da Tarefa 3
*
V1      LDA             =# 3
        JSR             RECEBE
*
*       Cálculo de t4
*
        LDB             =# 64
V2      LDA             B,X
        ADDA            B,X
*
*       Cálculo da exponencial
*
        NEGA
        JSR             EXP
        PSHA
        DECB
        BNE                     Teste de término de cálculo para 64 dados
        BRA             V2
*
*       Envia resultados para a Tarefa 1
*
        LDA             =# 4
        LDB             =# 1
        JSR             ENVIA
        BRA             V1
        END
```

V.3 - RELACIONAMENTO ENTRE AS TAREFAS

O relacionamento é mostrado através da figura (V.4), que especifica qual Tarefa interage com qual, através da invocação de Primitivas de Comunicação e Sincronização.

O NOp realiza o tratamento de toda invocação, por parte das Tarefas, das Primitivas de Comunicação e Sincronização e de rotinas fornecias pelo NOp, por exemplo, INIT e EXP.

À toda invocação, seja de primitiva ou de rotina do NOp, o NOp sempre salva o contexto da Tarefa, executa a invocação feita, restaura o contexto da Tarefa e retoma a sua execução. É necessário ter cuidado, por parte do programador, sobre o carregamento dos parâmetros das primitivas ou das rotinas.

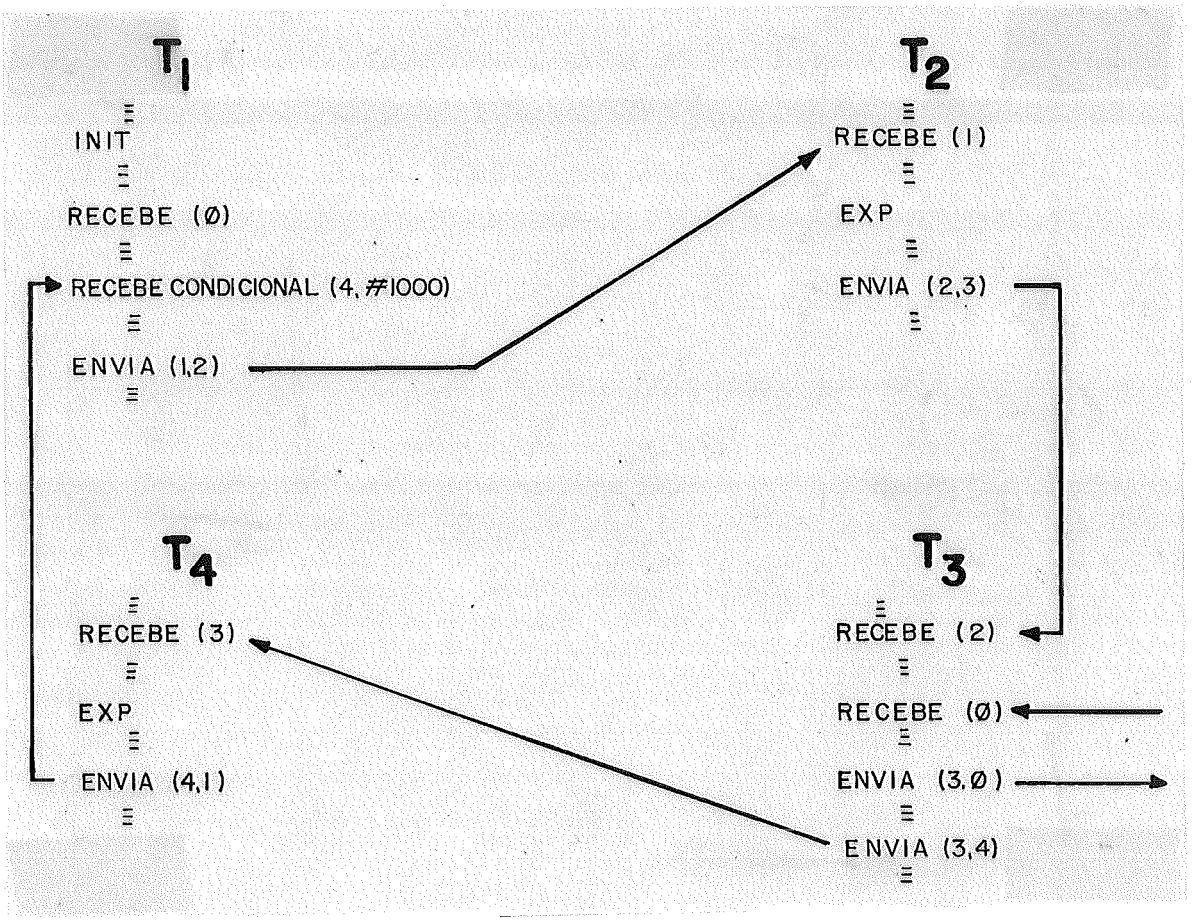


Figura V.4 - Relacionamento entre as Tarefas

As invocações das primitivas implicam em mudança de estado da Tarefa, que é gerenciado pelo NOp. Isto permite ao NOp fazer o acompanhamento da evolução da execução da Tarefa e detectar situações indevidas, por exemplo, entrada de dados a uma Tarefa quando esta não a pediu.

À toda geração de mensagem é introduzida nela um número que representa a sua ordem de produção, este número é para ser confrontado pelo NOp destino com o número do pedido feito pela Tarefa hóspede dele. Isto permite a entrega de mensagens às Tarefas de forma apropriada.

CAPÍTULO VI

CONCLUSÃO

Este trabalho propõe um Sistema Operacional simples, denominado Sistema de Multiprocessamento Clepsidra (SMC), que realiza processamento paralelo de Tarefas, cujo sequenciamento é dirigido por um processador especial ("hardware X software"), a partir de um grafo de macroordenação, gerado num estágio anterior por um programador. Este sistema foi projetado para um computador, como descrito em ROBERT FILHO (5), originalmente idealizado para processamento a Fluxo de Dados ("Data Flow"), denominado Clepsidra.

O SMC consiste num sistema de multiprocessamento, que suporta a execução de programas particionados em Tarefas, as quais correspondem, a grosso modo, ao tamanho de uma subrotina. Este foi o nível de partição escolhido para definir a unidade de paralelismo no SMC.

Nos sistemas a Fluxo de Dados, o sequenciamento de execução é governado automaticamente pelos resultados produzidos na execução das unidades de paralelismo de programa, que por sua vez vão servir como dados de entrada de outras unidades de paralelismo de programa para disparar as suas execuções, até cessarem naturalmente estas atividades.

Desta forma, num sistema a Fluxo de Dados não existe um gerenciamento centralizado. No caso do computador Clepsidra, ele foi construído a base de microprocessadores sob a forma de retículo, contendo também um processador Gerente que semeia o programa na Malha de Processamento, expandindo de forma automática (Fluxo de Dados), até que a atividade cesse e o Gerente recolha os resultados.

O fato deste computador ter sido construído a base de microprocessador, o qual não é especializado em comunicação, faz com que a unidade de paralelismo deva ser bem maior do que uma instrução de máquina, de maneira a que o custo do recebimento dos dados e da expedição dos resultados seja razoável em relação ao processamento útil, apesar de diminuir o paralelismo no computador Clepsidra.

A idéia básica que levou ao desenvolvimento do SMC foi ten

tar diminuir a taxa de comunicação entre as unidades de paralelismo, transferindo para o Gerente o sequenciamento da execução das unidades de paralelismo (entrada e saída de dados), que no caso de Fluxo de Dados ocorre automaticamente a partir de troca de mensagens entre as unidades de paralelismo.

O SMC é dividido em dois níveis. O primeiro nível é o responsável pelo escalonamento, alocação e controle de execução de Tarefas. Ele é composto de dois módulos: Sistema Operacional Gerente e Monitor do Processador de Comunicação, que residem no microcomputador Gerente. O segundo nível é o encarregado de realizar a execução das Tarefas. Ele é composto de dois módulos: Núcleo do Operador e Rotina de Comunicação do Nó. Uma cópia do Núcleo do Operador reside em cada um dos Operadores, da mesma forma, uma cópia da Rotina de Comunicação do Nó reside em cada um dos Nós. O inter-relacionamento entre os dois níveis se dá através de troca de comandos de forma assíncrona. Do primeiro nível para o segundo nível podem fluir os comandos: novas instruções, últimas instruções, descontinua tarefa e entrada de dados. Do segundo nível para o primeiro nível podem fluir os comandos: pede dados, resultados, término normal e término anormal.

A execução típica de um programa passa pelas fases seguintes:

- O programador particiona um programa (é sugerido o uso de Redes de Petri) e posteriormente caracteriza estas partições em Tarefas, para isto são introduzidas manualmente as Primitivas de Comunicação e Sincronização necessárias e após é preparado um cabeçalho com informações sobre o sequenciamento de alocação para cada Tarefa;

- Após, estas Tarefas são armazenadas em disco para posterior manuseio pelo Sistema Operacional Gerente. O Sistema Operacional Gerente põe em execução os programas de acordo com a sua ordem de armazenamento. Dentro de cada programa, a alocação de Tarefas nos Operadores é feita de acordo com a informação da macroordenação contida nos respectivos cabeçalhos, em forma de prioridade, sempre que houver Operador disponível;

- Durante a execução uma Tarefa pode requerer ou produzir dados, os quais são recebidos ou transmitidos sob a forma de mensagens, tanto

para as Tarefas Irmãs, bem como para o Sistema Operacional Gerente (impressão). A execução prossegue até o término normal ou por falha de execução dela ou de uma de suas Tarefas Irmãs (término anormal);

- O programa termina quando termina a execução da última de suas Tarefas. Em seguida os resultados produzidos pela Tarefa são impressos.

Um exemplo é descrito no Capítulo V, mostrando a decomposição da aplicação em partições, a caracterização em Tarefas e, através da figura (V.4), a forma de relacionamento estabelecida para estas Tarefas.

Alguns aspectos interessantes resultantes deste trabalho de pesquisa são:

- foi dada grande atenção ao problema de simplificação da comunicação. As mensagens são de tamanho fixo para uniformização de tratamento. Elas são filtradas nos diferentes módulos do SMC, só passando a nível hierárquico superior quando necessário, desta forma procura-se tomar as decisões a nível local, isto é, procura-se tratar a mensagem o mais próximo possível da sua origem;

- A arquitetura Clepsidra não dispõe de mecanismo de detecção de falhas. No entanto, o SMC provê tal mecanismo através da análise do estado do exercício das Tarefas. Esta análise é feita pelos módulos Sistema Operacional Gerente, Monitor do Processador de Comunicação e Núcleo do Operador, que verificam a consistência dos comandos que por eles fluem. Por exemplo, se o Monitor do Processador de Comunicação recebe um pedido de entrada de dados do Núcleo do Operador, e o último estado da Tarefa indicava entrada de dados ainda não completada, então é detectada uma falha de execução daquela Tarefa. O SMC é capaz de detectar falhas a esse nível não permitindo a proliferação de falhas. Quando ocorre falha, o critério adotado é de retirar o programa de execução (tratamento de exceção simples), devido à complexidade de recuperação na atual arquitetura e para o tipo de processamento que o SMC suporta;

- O SMC foi estruturado de forma conveniente a poder incorporar multiprogramação a nível de Operador. O fato de ter sido implementado detecção de falhas, faz com que um trabalho futuro conte com uma base para a instalação de outras características de tolerância a falha;

- a elaboração do SMC evidenciou a necessidade de se estudar o problema da partição automática de programas em Tarefas, bastante complexo, constituindo um campo fértil para trabalho. Uma das formas de abordar este trabalho é gerar uma arborecência contendo paralelismo maximal. Em seguida, um algoritmo de "recorte" subdivide a arborecência, cortando-a segundo linhas que rompe o menor número possível de arcos. Este tipo de recorte procura minimizar a comunicação entre as partições (Tarefas);

- no SMC, uma Tarefa ocupa um Operador desde o instante da alocação até o fim da execução estabelecida pelo Núcleo do Operador, não havendo portanto multiprogramação a nível de Operador. Este fato implica que o SMC é incapaz de executar programas que contenham mais Tarefas do que o número de Operadores da Malha de Processamento, pois desta forma pode ocorrer Bloqueio Perpétuo ("deadlock"). Assim, no SMC deixou-se de especificar uma característica importante para um sistema de multiprocessamento que se destina a uso geral. Esta decisão foi tomada em função de limitar o escopo da pesquisa, dada a dificuldade de especificação desta característica na arquitetura Clepsidra (por ex.: falta de relógio de tempo real).

Finalmente, vale a pena verificar se o objetivo inicial foi ou não atingido. O propósito inicial era a especificação e a implementação de um Sistema Operacional para processamento paralelo de finalidade geral, procurando desta forma contribuir para o conhecimento de Sistemas Operacionais desta classe, ainda bastante raros na literatura. O Sistema foi especificado, os algoritmos dos módulos SOG, MPC e NOp foram desenvolvidos e em seguida codificados e testados isoladamente, cujas listagens se encontram no Apêndice D. Todavia, não foi ainda possível testar funcionalmente o SMC, pois o "hardware" do computador Clepsidra não está completamente pronto.

REFERÊNCIAS BIBLIOGRÁFICAS

(1) ROSEN, S., "Electronic Computers. A Historical Survey", Computing Survey, Vol. 1, nº 1, pp. 7-36, (1969).

(2) DENNING, P. J., "Third Generation Computer System", Computing Survey, Vol. 3, nº 4, pp. 175-216, (1971).

(3) BEYERS, J. W.; DOHSE, L. J.; FUCETOLA, J. P.; KOCHIS, R. L.; LOB, C. G.; TAYLOR, G. L.; ZELLER, E. R., "A 32 bit VLSI CPU chip", IEEE Journal of Solid-State Circuits, SC 16(5), pp. 537-541, (1981).

(4) STONE, H. S., editor, Introduction to Computer Architecture, chapter 8, "Parallel Computers", Stone, H. S., pp. 318-374, The SRA Computer Science Series, (1975).

(5) FLORENTIN, J. S., "An Alternative Approach to Multiprocessor Design", Microprocessing and Microprogramming, Vol. 13, nº 1, pp. 11-19, (1984).

(6) ENSLOW JR, P. H., Multiprocessor and Parallel Processing, John Wiley and Sons, New York, (1974).

(7) FLYNN, M. J., "Some Computer Organizations and their Effectiveness", IEEE Transaction on Computers, C21(9), pp. 948-960, (1972).

(8) VON NEUMANN, J.; BURKS, A. W.; GOLDSTINE, H., "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", John von Neumann Collected Works, Vol. 5, Pergamon Press, (1963).

(9) HAYNES, L. S.; LAU, R. L.; SIEWIOREK, D. P.; MIZELL, D. W., "A Survey of Highly Parallel Computing", Computer, Vol 15, nº 1, pp. 9-24, (1982).

(10) SIEWIOREK, D. P.; BELL, C. G.; NEWELL, A., Computer Structures: Principles and Examples , McGraw-Hill Inc., (1982)

(11) FULLER, S. H.; OUSTERHOUT, J. K.; RASKIN, L.; RUBINFE, P. I.; SINDHU, P. J.; SWAN, R. J., "Multi-Microprocessors: An overview and Working Example", Proc. of IEEE, Vol. 6, nº 2, pp. 216-228, (1978).

(12) ROBERTS, L. G., "Multiple Computer Networks and Intercomputer Communication", Proc. ACM Symposium on Operating System Principles, (1967).

(13) FATHI, T. E.; KRIEGER, M., "Multiple Microprocessor Systems: What, Why and When", Computer, Vol. 16, nº 3, pp. 23-32, (1983).

(14) SILVERMAN, G.; STUNDEL, A.; LEHMAN, J., "The Modular Multiprocessor - A Model for Laboratory Design", IEEE Micro, Vol. 2, nº 2, pp. 51-62, (1982).

(15) ROBERT FILHO, E., Arquitetura de Processadores a Fluxo de Dados, Tese de Doctorat D'Etat, em preparação, Institut de Programmation, Université Pierre et Marie Curie, Paris IV, França, (1984).

(16) MOTOROLA, Motorola Microprocessor Data Manual, Motorola Semiconductor Products Inc., (1981).

(17) SIGNETICS, 8X300 Description and Specification, Signetics, (1978).

(18) DIJKSTRA, E. W., "The Structure of THE Multiprogramming System", Communication of ACM, vol. 11, nº 5, pp. 341-346, (1968).

(19) MADNICK, E. S.; DONOVAN, J. J., Operating Systems , McGraw-Hill Book Company, (1974).

(20) CINTRA, S. A. R., "Projeto de um Sistema de Multiprocessamento e Multiprogramação para o Computador Clepsidra", Anais do II Congresso da Sociedade Brasileira de Computação, pp. 329-332, (1982).

(21) PETERSON, J. L., Petri Net Theory and the Modeling of Systems, Prentice-Hall, Inc., (1981).

(22) TOULOTTE, J. M.; PARSY, J. P., "A Method for Decomposing Interpreted Petri Nets and Its Utilization", Digital Process, Vol. 5, nº 3-4, pp. 223-234, (1979).

(23) GENTLEMAN, W. M., "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept", Software - Practice and Experience, Vol. 11, pp. 435-466, (1981).

(24) ANDERSON, T.; LEE, P. A., Fault Tolerance Principlesand Practice, Prentice Hall International, (1981).

(25) UCSD, UCSD Pascal Manual, Softech Microsystems, (1980).

(26) MPL, MPL User's Guide, Motorola Semiconductor Products Inc., (1980).

(27) MACKELVEY, T. R.; AGRAWAL, D. P., "Design of Software for Distributed/Multiprocessor Systems", AFIPS Conference Proceedings, Vol. 51, (1982).

(28) BOTTURA, C. P., Princípios de Controle e Servomecanismos, Centro Acadêmico Bernardo Sayão, FEC - UNICAMP, (1980).

(29) MOTOROLA, MC6809-MC6809E Microprocessor Programming Manual, Motorola Semiconductor Products Inc., (1981).

APÊNDICE A

DESCRIÇÃO DA CAIXA POSTAL (CxP)

A CxP é uma área de memória compartilhada entre os processadores Gerente e Processador de Comunicação, servindo de meio de comunicação entre estes dois processadores.

O acesso à qualquer posição da CxP é mutuamente exclusivo a nível de ciclo de máquina, que é realizado através do dispositivo denominado Árbitro.

A configuração atual do computador Clepsidra possue quatro Operadores na Malha de Processamento, isto faz com que quatro CxPs sejam suficientes para a comunicação, cada uma alocada de forma estática a um Operador.

O acesso a cada CxP é controlado por um campo, denominado Controle de Caixa Postal (CCP), no qual são inseridos pelos dois processadores, códigos de comando que explicitam, além da permissão de acesso à CxP pelo processador, também o que deve ser feito com o conteúdo da Caixa Postal.

Assim,a área de comunicação entre o Gerente e o Processador de Comunicação é composta de quatro CxPs e quatro CCPs respectivos, como é mostrado na figura (A.1).

Como o fluxo de mensagem é determinado pelo comportamento de execução da Tarefa, basta haver um CCP para cada CxP, pois um módulo só toma alguma atitude se o comando for para ele, garantindo a exclusão da CxP. Este tipo de exclusão resolve para um processamento normal. Ocorre que, quando há falha detectada e uma vez o SOG sendo avisado, ele toma a atitude de descontinuar as Tarefas Irmãs daquela Tarefa. Portanto, torna-se necessário prover mais um campo, que serve para indicar se a execução de uma Tarefa está normal ou não.

Dessa forma, cada CCP é composto de dois campos, um indica o tipo de pedido feito (CCP/P), o outro se há falha ou não (CCP/F).

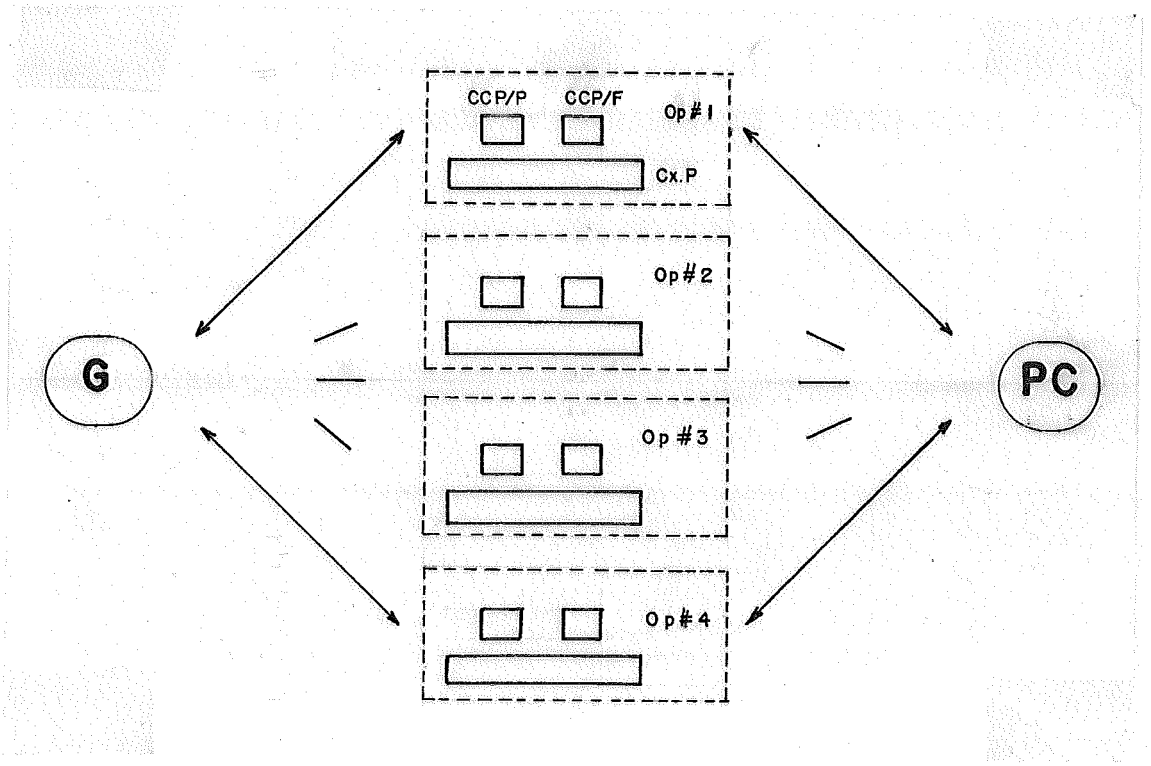
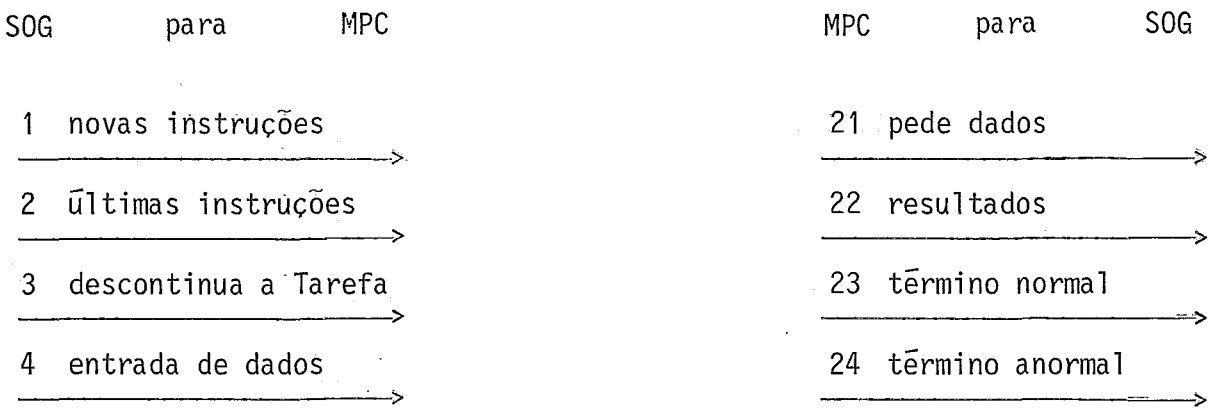


Figura A.1 - Caixas Postais e os respectivos Controles de Caixas Postais.

Assim os módulos SOG e MPC, para cada leitura, depósito ou retirada de informação, verificam primeiro o conteúdo de CCP/F (Ø - normal, 1 - falha). Se não há indicação de falha, então o CCP/P é verificado, onde cada pedido é específico, no sentido e na característica, portanto não há ambiguidade. Após o atendimento do pedido, o módulo que o atendeu coloca zero, indicando a disponibilidade da CxP. Se o CCP/P não conter zero, o módulo não pode acessar a CxP, passando a leitura para o próximo CCP/P, não ficando aguardando a liberação.

O fluxo dos comandos é mostrado abaixo, observando os dois sentidos de comunicação, cujos códigos de comandos colocados nos CCPs influenciam o sentido da comunicação. Portanto os códigos de comandos são exclusivos de cada módulo, SOG e MPC, com exceção do código de CxP disponível, que é zero, comum aos dois. Durante a comunicação, os módulos SOG ou o MPC podem detectar um código de comando (pedido) inconsistente em relação ao atual estado da Tarefa. Neste caso, ele podem inverter o sentido da comunicação, encaminhando o código de erro constatado.


SOG para MPC
1 novas instruções
2 últimas instruções
3 descontinua a Tarefa
4 entrada de dados
MPC para SOG
21 pede dados
22 resultados
23 término normal
24 término anormal

APÊNDICE B

COMUNICAÇÃO VIA BARRAMENTO DE SERVIÇO (BS)

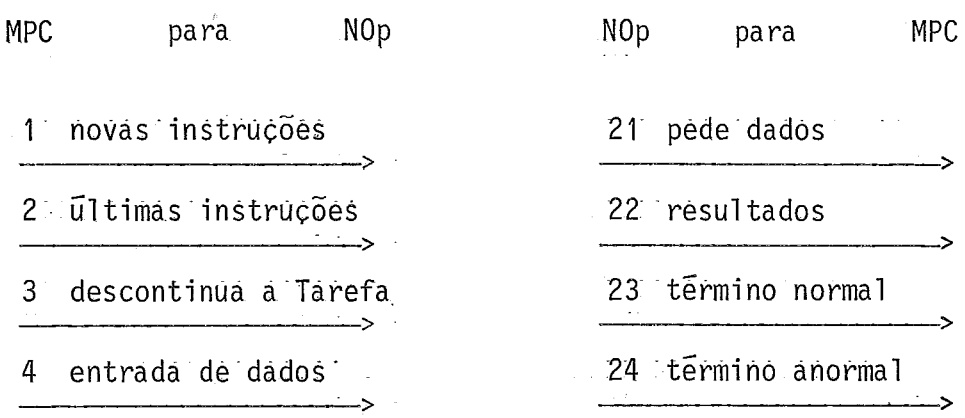
O BS é o meio de comunicação entre o Monitor do Processador de Comunicação (MPC) e os quatro Núcleos dos Operadores (NOp).

Por característica da arquitetura Clepsidra, via o BS, o MPC pode interromper qualquer NOp, enquanto que qualquer NOp somente pode pedir uma interrupção, pois o MPC possue o controle do BS.

Assim, o MPC ao iniciar uma comunicação primeiro coloca o código que deseja enviar a um NOp no BS, depois interrompe este NOp. O NOp ao receber a interrupção, que já era esperada ou não, lê o código no BS e verifica a consistência do pedido, atendendo-o ou não, de acordo com o estado de execução da Tarefa. No caso de vir informação (códigos ou dados) relativa ao comando do MPC, este envia um a um, a cada comutação da linha que o NOp usa para pedir interrupção, até completar a comunicação (novas instruções ou entrada de dados - 64 vezes; últimas instruções - o primeiro "byte" indica a quantidade; descontinua a Tarefa - o NOp realiza o pedido).

Para a comunicação no sentido NOp ao MPC, primeiro o NOp faz o pedido de interrupção, que o MPC atende através da colocação do código de aceite (código zero) no BS e em seguida interrompe o NOp pedinte, avisando-o para ler o BS (o NOp já aguardava). O NOp então coloca no BS o pedido, e em seguida, comuta a linha de pedido de interrupção. Se o pedido for de envio de resultados, a cada interrupção do MPC, o NOp coloca um dado no BS e comuta a linha de pedido de interrupção, isto até completar 64 envios.

Abaixo são mostrados os fluxos dos comandos, tanto no sentido do MPC para o NOp, bem como, no sentido do NOp para o MPC. Estes comandos são os mesmos que fluem pela Caixa Postal, mostrados no Apêndice A.


MPC para NOp
1 novas instruções
2 últimas instruções
3 descontinua a Tarefa
4 entrada de dados
NOp para MPC
21 pede dados
22 resultados
23 término normal
24 término anormal

APÊNDICE C

DESCRIÇÃO DA ESTRUTURA DE "BUFFERS"

Um "buffer" é um conjunto de posições de memória de tamanho fixo, que serve de área de armazenamento temporário de dados. No Sistema de Multiprocessamento Clepsidra existem oito "buffers", que são usados para fazer o Rolamento ("SPOOL"), podendo serem alocados a duas filas. A primeira destas filas indica a condição de disponibilidade dos "buffers", a segunda indica a condição de necessidade de serem impressos os dados contidos nos seus "buffers".

A alocação de "buffers" às filas é feita por dois procedimentos: Rolamento e Impressão. O Procedimento Rolamento usa os "buffers" alocados à fila 1, preenchendo-os e alocando-os à fila 2. O Procedimento Impressão imprime os dados contidos nos "buffers" alocados na fila 2, e depois aloca-os à fila 1. Esta comutação de "buffers" é mostrada na figura (C.1).

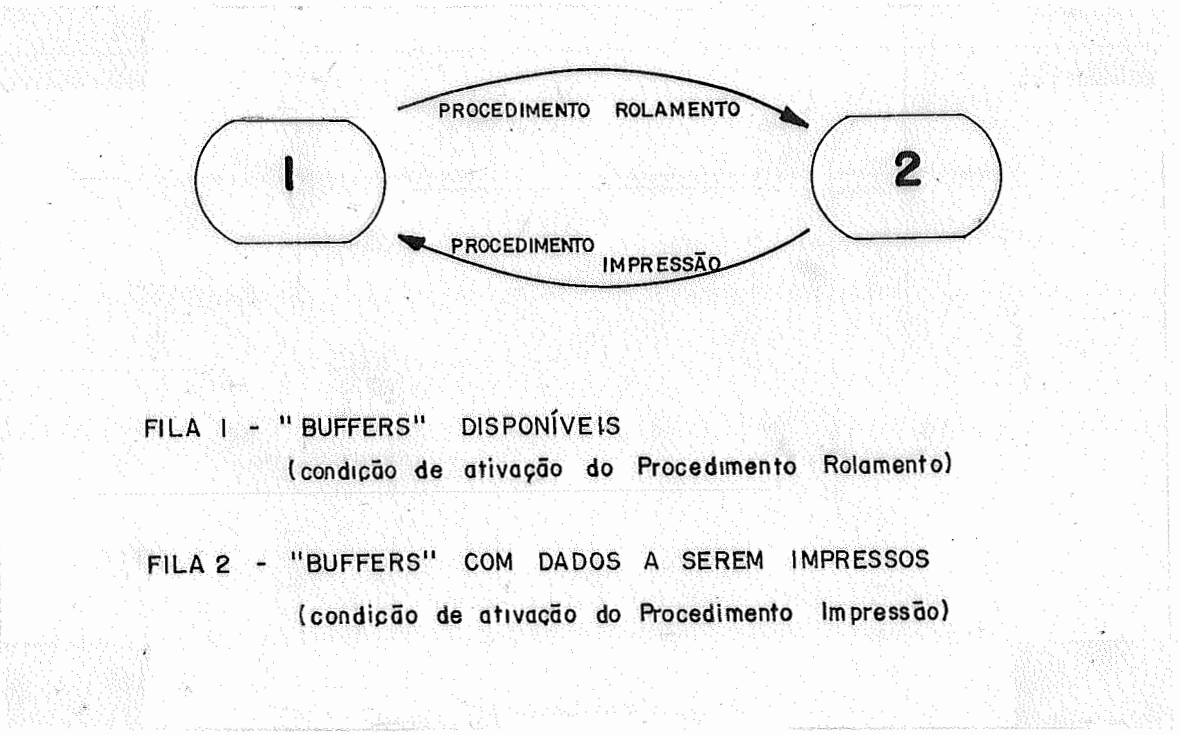


Figura C.1 - Estrutura de alocação de "buffers".

MÓDULO SISTEMA OPERACIONAL GERENTE

```
1=D      1 $%L PRINTER=$;
1=D      1 PROGRAM SO_GERENTE;
1=D      3
1=D      3 CONST
1=D      3       MAX_BUFFERS = 8;
1=D      3       MAX_LINHAS_BUFFERS = 64;
1=D      3       MAX_FI_BUFFERS = 2;
1=D      3       MAX_FI_BDT = 8;
1=D      3       DISP = 0; { CAIXA POSTAL DISPONIVEL }
1=D      3       NI = 1;   { NOVAS INSTRUCOES }
1=D      3       UI = 2;   { ULTIMAS INSTRUCOES }
1=D      3       DT = 3;   { DESCONTINUA TAREFA }
1=D      3       ED = 4;   { ENTRADA DADOS }
1=D      3
1=D      3
1=D      3 TYPE
1=D      3       LINHA = STRING [10];
1=D      3
1=D      3       APONT_BUFFER = ^BUFFER;
1=D      3       BUFFER = RECORD
1=D      3                  LINHAS : ARRAY [0..MAX_LINHAS_BUFFERS]
1=D      3                                 OF  LINHA;
1=D      3                  ULTIMO_BUFFER, NOVA_IMPRESSAO : BOOLEAN;
1=D      3                  PROX_BUFFER : APONT_BUFFER
1=D      3                END;
1=D      3
1=D      3
1=D      3       APONT_BD_TAREFA = ^BD_TAREFA;
1=D      3       BD_TAREFA = RECORD
1=D      3                     NOME_PROG,
1=D      3                     NOME_TAR : STRING [2];
1=D      3                     NRO_PRIOR : STRING [1];
1=D      3                     ESTADO : (EXECUTANDO, ENTR_DADOS,
1=D      3                               SAI_DADOS, TERMINOU);
1=D      3                     ERRO,
1=D      3                     ARQ_DADO_ABERTO,
1=D      3                     ARQ_TAR_ABERTO,
1=D      3                     ARQ_IMPR_ABERTO : BOOLEAN;
1=D      3                     CODIGO_ERRO,
1=D      3                     OPERADOR,
1=D      3                     NRO_CXP : INTEGER;
1=D      3                     PROX_BDT : APONT_BD_TAREFA
1=D      3                   END;
1=D      3
1=D      3
1=D      3 VAR
1=D      3     TEX_OP1_SU,
1=D      3     TEX_OP2_SU,
1=D      3     TEX_OP3_SU,
1=D      3     TEX_OP4_SU : TEXT;
1=D   1207     TOPO_BUFFER : ARRAY [1..MAX_FI_BUFFERS]
1=D   1207                       OF  APONT_BUFFER;
1=D   1209     TOPO_BDT : ARRAY [1..MAX_FI_BDT]  OF  APONT_BD_TAREFA;
1=D   1217     TERMINO_LE_DISCO, FIM_PROCESSAMENTO,
1=D   1217     OP_DISPONIVEL, EXISTE_TAREFA : BOOLEAN;
1=D   1221     INIC_CXP,
1=D   1221     FIM_CXP,
1=D   1221     NRO_TAREFA : INTEGER;
1=D   1224     ARQ_IMPR : STRING [10];
1=D   1230     ARQ_TARQ, ARQ_TAR, ARQ_DADOS : STRING [17];
1=D   1257     STRING_NRO_TAREFA : STRING [1];
1=D   1258     INFO :  INTEGER;
1=D   1259     I : INTEGER;
1=D   1260
```

```
 1#D  1260
 2#D     1 PROCEDURE  ESCR_CXP_INFO (INIC_CXP, FIM_CXP,
 2#D     1                           INFO : INTEGER); EXTERNAL;
 2#D     4
 3#D     1 PROCEDURE LE_CXP_INFO (INIC_CXP, FIM_CXP, INFO  : INTEGER);
 3#D     4                        EXTERNAL;
 3#D     4
 4#D     1 PROCEDURE ESCR_CCP (NRO_CXP, INFO : INTEGER); EXTERNAL;
 4#D     3
 5#D     1 PROCEDURE LE_CCP (NRO_CXP, INFO : INTEGER); EXTERNAL;
 5#D     3
 6#D     1 PROCEDURE LE_TECLADO; EXTERNAL;
 6#D     1
 7#D     1 PROCEDURE LE_DISCO (VAR TEXTO : TEXT; I : INTEGER);
 7#0     0 BEGIN
 7#1     0   WHILE  (NOT (EOF (TEXTO)))  AND  (I < MAX_LINHAS_BUFFERS) DO
 7#2    13   BEGIN
 7#3    13     READLN (TEXTO, INFO);
 7#3    28     I := I + 1
 7#2    29   END
 7#0    33 END;
 7#0    50
 8#D     1 PROCEDURE ESCR_DISCO (VAR TEXTO : TEXT);
 8#0     0 BEGIN
 8#1     0   I := 0;
 8#1     4   WHILE  (I < MAX_LINHAS_BUFFERS)  DO
 8#2    11   BEGIN
 8#3    11     WRITELN (TEXTO, INFO );
 8#3    27     I := I + 1
 8#2    30   END
 8#0    35 END;
 8#0    52
 9#D     1 PROCEDURE ESCREVE_CABECALHO;
 9#0     0 BEGIN
 9#1     0   WITH  TOPO_BDT ßß¢^  DO
 9#2    14   BEGIN
 9#3    14     WRITELN; WRITELN;
 9#3    30     WRITELN (' TAREFA SMMC.',NOME_TAR);
 9#3    73     WRITELN (' RESULTADOS:')
 9#2   105   END
 9#0   105 END;
 9#0   118
10#D     1 PROCEDURE ESCREVE_FIM_TAREFA;
10#0     0 BEGIN
10#1     0   WITH  TOPO_BDT ß9¢^  DO
10#2    14   BEGIN
10#3    14     WRITELN; WRITELN;
10#3    30     WRITELN (' FIM DA TAREFA SMMC.',NOME_TAR)
10#2    80   END
10#0    80 END;
10#0    92
11#D     1 PROCEDURE CRIA_BUFFERS (VAR AP_BUFFER : APONT_BUFFER);
11#D     2 VAR  NAP_BUFFER : APONT_BUFFER;
11#0     0 BEGIN
11#1     0   AP_BUFFER := NIL;
11#1     3   FOR I := 1 TO MAX_FI_BUFFERS  DO
11#2    17   BEGIN
11#3    17     NEW (NAP_BUFFER);
11#3    24     NAP_BUFFER^.PROX_BUFFER := AP_BUFFER;
11#3    31     AP_BUFFER := NAP_BUFFER
11#2    32   END
11#0    34 END;
11#0    50
12#D     1 PROCEDURE  CRIA_BD_TAREFA (VAR AP_BDT : APONT_BD_TAREFA);
12#D     2 VAR  NAP_BDT : APONT_BD_TAREFA;
12#0     0 BEGIN
```

```
12:1      0    AP_BDT := NIL;
12:1      3    FOR I := 1 TO MAX_FI_BDT DO
12:2     17    BEGIN
12:3     17      NEW (NAP_BDT);
12:3     22      NAP_BDT^.PROX_BDT := AP_BDT;
12:3     28      AP_BDT := NAP_BDT
12:2     29    END
12:0     31  END;
12:0     56
13:D      1  PROCEDURE PP_BUFFER (I, J : INTEGER);
13:D      3  VAR
13:D      3      P : APONT_BUFFER;
13:0      0  BEGIN
13:1      0    IF  TOPO_BUFFER ßIÇ = NIL
13:1     12      THEN BEGIN
13:3     16             WRITELN; WRITELN;
13:3     32             WRITELN (' ERRO * PP_BUFFER I = ',I,' J = ',J);
13:3    111             WRITELN ('        PROCESSAMENTO INTERROMPIDO *');
13:2    166           END
13:1    166      ELSE BEGIN
13:3    168             IF  TOPO_BUFFER ßJÇ = NIL
13:3    180               THEN BEGIN
13:5    184                      TOPO_BUFFER ßJÇ := TOPO_BUFFER ßIÇ;
13:5    208                      TOPO_BUFFER ßIÇ := TOPO_BUFFER ßIÇ^.
13:5    231                                           PROX_BUFFER;
13:5    235                      TOPO_BUFFER ßJÇ^.PROX_BUFFER := NIL
13:4    250                    END
13:3    252                 ELSE BEGIN
13:5    254                      P := TOPO_BUFFER ßJÇ;
13:5    268                      WHILE  P^.PROX_BUFFER <> NIL
13:5    272                        DO  P := P^.PROX_BUFFER;
13:5    284                      P^.PROX_BUFFER := TOPO_BUFFER ßIÇ;
13:5    301                      TOPO_BUFFER ßIÇ := TOPO_BUFFER ßIÇ^.
13:5    324                                           PROX_BUFFER;
13:5    328                      P^.PROX_BUFFER^.PROX_BUFFER := NIL
13:4    335                    END
13:2    337           END
13:0    337  END;
13:0    356
14:D      1  PROCEDURE  PP_BDT (I, J : INTEGER);
14:D      3  VAR
14:D      3      P : APONT_BD_TAREFA;
14:0      0  BEGIN
14:1      0    IF  TOPO_BDT ßIÇ = NIL
14:1     12      THEN BEGIN
14:3     16             WRITELN; WRITELN;
14:3     32             WRITELN (' ERRO * PP_BDT I = ',I,' J = ',J);
14:3    108             WRITELN ('     * PROCESSAMENTO INTERROMPIDO *');
14:2    162           END
14:1    162      ELSE BEGIN
14:3    164             IF  TOPO_BDT ßJÇ = NIL
14:3    176               THEN BEGIN
14:5    180                      TOPO_BDT ßJÇ := TOPO_BDT ßIÇ;
14:5    204                      TOPO_BDT ßIÇ := TOPO_BDT ßIÇ^.
14:5    227                                        PROX_BDT;
14:5    230                      TOPO_BDT ßJÇ^.PROX_BDT := NIL
14:4    244                    END
14:3    246                 ELSE BEGIN
14:5    248                      P := TOPO_BDT ßJÇ;
14:5    262                      WHILE  P^.PROX_BDT <> NIL
14:5    265                        DO  P := P^.PROX_BDT;
14:5    276                      P^.PROX_BDT := TOPO_BDT ßIÇ;
14:5    292                      TOPO_BDT ßIÇ := TOPO_BDT ßIÇ^.PROX_BDT;
14:5    318                      P^.PROX_BDT^.PROX_BDT := NIL
14:4    323                    END
14:2    325           END
```

```
14:0   325 END;
14:0   344
15:D     1 PROCEDURE INICIALIZAR;
15:D     1 VAR
15:D     1     TEXTO_INICIALIZAR : TEXT;
15:D   302     LINHA, LINHA_TEMPORARIA : STRING 810Ç;
15:D   314
16:D     1   PROCEDURE DELETA_LINHA_TEMPORARIA (K, L : INTEGER);
16:0     0   BEGIN
16:1     0     LINHA_TEMPORARIA := LINHA;
16:1    10     DELETE (LINHA_TEMPORARIA, 1, K);
16:1    19     L := LENGTH (LINHA_TEMPORARIA) - L;
16:1    29     DELETE (LINHA_TEMPORARIA, 2, L)
16:0    38   END;
16:0    50
15:0     0 BEGIN
15:1     0   NRO_TAREFA := NRO_TAREFA + 1;
15:1    20   DELETE (ARQ_TARQ, 5, 1);
15:1    28   STR (NRO_TAREFA, STRING_NRO_TAREFA);
15:1    41   INSERT (STRING_NRO_TAREFA, ARQ_TARQ, 5);
15:1    52 Ö%I-ç
15:1    52   RESET (TEXTO_INICIALIZAR, ARQ_TARQ);
15:1    62   IF  IORESULT <> 0
15:1    64     THEN BEGIN
15:3    68            EXISTE_TAREFA := FALSE;
15:3    72            CLOSE (TEXTO_INICIALIZAR, LOCK)
15:2    78          END
15:2    78 Ö%I+ç
15:1    78     ELSE WITH  TOPO_BDT Ö10Ç^  DO
15:3    95            BEGIN
15:3    95              Ö ATRIBUICAO DO NOME ç
15:4    95              DELETA_LINHA_TEMPORARIA (4, 1);
15:4    99              NOME_TAR := LINHA_TEMPORARIA;
15:4   107
15:4   107              Ö ATRIBUICAO DA PRIORIDADE ç
15:4   107              DELETA_LINHA_TEMPORARIA (6, 1);
15:4   111              NRO_PRIOR := LINHA_TEMPORARIA;
15:4   121
15:4   121              Ö ARQUIVO PARA IMPRESSAO NAO FOI ABERTO ç
15:4   121              ARQ_IMPR_ABERTO := FALSE;
15:4   128
15:4   128              Ö ARQUIVO DE DADOS NAO FOI ABERTO ç
15:4   128              ARQ_DADO_ABERTO := FALSE;
15:4   135
15:4   135              Ö ARQUIVO TAREFA NAO FOI ABERTO ç
15:4   135              ARQ_TAR_ABERTO := FALSE;
15:3   142            END;
15:1   142   PPBDT (10, 1)
15:0   144 END;
15:0   164
18:D     1 PROCEDURE  CARREGA_TAREFA;
18:D     1 VAR TEXTO_CAR_TAR : TEXT;
18:0     0 BEGIN
18:1     0   WITH  TOPO_BDT Ö1Ç^  DO
18:2    27   BEGIN
18:3    27     IF  NOT  ARQ_TAR_ABERTO
18:3    27       THEN BEGIN
18:5    34              INSERT (NOME_TAR, ARQ_TAR, 5);
18:5    45              ARQ_TAR_ABERTO := TRUE;
18:5    52              RESET(TEXTO_CAR_TAR, ARQ_TAR)
18:4    64            END;
18:3    64     LE_DISCO (TEXTO_CAR_TAR, 1);
18:3    71     ESCR_CXP_INFO (INIC_CXP, FIM_CXP, 1);
18:3    82     ESCR_CCP (NRO_CXP, NI);
18:3    90     IF  EOF (TEXTO_CAR_TAR)
18:3    97     THEN BEGIN
```

```
18:5    99                CLOSE (TEXTO_CAR_TAR, LOCK);
18:5   107                PPBDT (1, 2);
18:5   111                ESCR_CCP (NRO_CXP, DISP);
18:4   119              END
18:2   119      END
18:0   119 END;
18:0   138
19:D     1 PROCEDURE  ENTRADA_DADOS;
19:D     1 VAR  TEX_OP1_ED,
19:D     1      TEX_OP2_ED,
19:D     1      TEX_OP3_ED,
19:D     1      TEX_OP4_ED : TEXT;
19:0     0 BEGIN
19:1     0   WITH  TOPO_BDT 63Ç^  DO
19:2    66   BEGIN
19:3    66     IF  NOT  ARQ_DADO_ABERTO
19:3    66       THEN BEGIN
19:5    74              INSERT (NOME_TAR, ARQ_DADOS, 5);
19:5    85              ARQ_DADO_ABERTO := FALSE;
19:5    92              CASE  OPERADOR  OF
19:5    99                1 : RESET (TEX_OP1_ED, ARQ_DADOS);
19:5   114                2 : RESET (TEX_OP2_ED, ARQ_DADOS);
19:5   129                3 : RESET (TEX_OP3_ED, ARQ_DADOS);
19:5   144                4 : RESET (TEX_OP4_ED, ARQ_DADOS)
19:5   156              END;
19:4   174            END;
19:3   174      CASE  OPERADOR  OF
19:3   181        1 : LE_DISCO (TEX_OP1_ED, I);
19:3   191        2 : LE_DISCO (TEX_OP2_ED, I);
19:3   201        3 : LE_DISCO (TEX_OP3_ED, I);
19:3   211        4 : LE_DISCO (TEX_OP4_ED, I)
19:3   216      END;
19:3   236      ESCR_CXP_INFO (INIC_CXP, FIM_CXP, I);
19:3   247      ESCR_CCP (NRO_CXP, ED);
19:3   255      IF  TERMINO_LE_DISCO
19:3   255        THEN BEGIN
19:5   260              CASE  OPERADOR  OF
19:5   267                1 : CLOSE (TEX_OP1_ED, LOCK);
19:5   278                2 : CLOSE (TEX_OP2_ED, LOCK);
19:5   289                3 : CLOSE (TEX_OP3_ED, LOCK);
19:5   300                4 : CLOSE (TEX_OP4_ED, LOCK)
19:5   308              END;
19:5   326              ESCR_CCP (NRO_CXP, DISP)
19:4   332            END;
19:3   334      PPBDT (3, 2);
19:2   338    END
19:0   338 END;
19:0   378
19:0   378
20:D     1 PROCEDURE  SAIDA_DADOS;
20:0     0 BEGIN
20:1     0   WITH  TOPO_BDT 64Ç^  DO
20:2    14   BEGIN
20:3    14     IF  NOT  ARQ_IMPR_ABERTO
20:3    14       THEN BEGIN
20:5    19              INSERT (NOME_TAR, ARQ_IMPR, 5);
20:5    28              ARQ_IMPR_ABERTO := TRUE;
20:5    33              CASE  OPERADOR  OF
20:5    38                1 : REWRITE (TEX_OP1_SU, ARQ_IMPR);
20:5    53                2 : REWRITE (TEX_OP2_SU, ARQ_IMPR);
20:5    68                3 : REWRITE (TEX_OP3_SU, ARQ_IMPR);
20:5    83                4 : REWRITE (TEX_OP4_SU, ARQ_IMPR)
20:5    95              END;
20:4   112            END;
20:3   112      LE_CXP_INFO (INIC_CXP, FIM_CXP, I);
20:3   123      CASE  OPERADOR  OF
```

```
20:3   128          1 : ESCR_DISCO (TEX_OP1_SU);
20:3   135          2 : ESCR_DISCO (TEX_OP2_SU);
20:3   142          3 : ESCR_DISCO (TEX_OP3_SU);
20:3   149          4 : ESCR_DISCO (TEX_OP4_SU)
20:3   151        END;
20:3   170        ESCR_CCP (NRO_CXP, DISP);
20:3   176        PPBDT (4, 2);
20:2   180      END
20:0   180  END;
20:0   192
20:0   192
21:0     1  PROCEDURE  TERMINO;
21:0     0  BEGIN
21:1     0    WITH  TOPO_BDT 67Ç^  DO
21:2    14    BEGIN
21:3    14      IF  ERRO  THEN
21:4    19      BEGIN
21:5    19        INIC_CXP := OPERADOR * 64 + 1000;
21:5    31        FIM_CXP := INIC_CXP + 63;
21:5    39        LE_CXP_INFO (INIC_CXP, FIM_CXP, INFO);
21:5    50        CODIGO_ERRO := INFO
21:4    53      END
21:3    57                  ELSE
21:4    59      IF  ARQ_IMPR_ABERTO
21:4    59        THEN BEGIN
21:6    63               ARQ_IMPR_ABERTO := FALSE;
21:6    68               INSERT (NOME_TAR, ARQ_IMPR, 5);
21:6    77               CASE  OPERADOR  OF
21:6    82                 1 : CLOSE (TEX_OP1_SU, LOCK);
21:6    93                 2 : CLOSE (TEX_OP2_SU, LOCK);
21:6   104                 3 : CLOSE (TEX_OP3_SU, LOCK);
21:6   115                 4 : CLOSE (TEX_OP4_SU, LOCK)
21:6   123               END;
21:6   140               PPBDT (5, 6)
21:5   142             END
21:4   144        ELSE  PPBDT (5, 0)
21:2   148    END
21:0   150  END;
21:0   162
22:0     1  PROCEDURE  ROLAMENTO;
22:0     1  VAR TEXTO_SR : TEXT;
22:0     0  BEGIN
22:1     0    WITH  TOPO_BDT 68Ç^  DO
22:2    27    BEGIN
22:3    27      IF  NOT  ARQ_IMPR_ABERTO
22:3    27        THEN BEGIN
22:5    34               ARQ_IMPR_ABERTO := TRUE;
22:5    41               TOPO_BUFFER 81Ç^.NOVA_IMPRESSAO := TRUE;
22:5    58               INSERT (NOME_TAR, ARQ_IMPR, 5);
22:5    69               RESET (TEXTO_SR, ARQ_IMPR)
22:4    81             END;
22:3    81      WHILE (TOPO_BUFFER 81Ç <> NIL)  DO
22:4    97      BEGIN
22:5    97        LE_DISCO (TEXTO_SR,I);
22:5   104        IF EOF (TEXTO_SR)
22:5   111          THEN BEGIN
22:7   113                 CLOSE (TEXTO_SR, LOCK);
22:7   121                 PPBDT (6, 7);
22:7   125                 TOPO_BUFFER 81Ç^.ULTIMO_BUFFER := TRUE
22:6   140               END
22:5   142          ELSE  TOPO_BUFFER 81Ç^.ULTIMO_BUFFER := FALSE;
22:5   161        PPBUFFER (1, 2);
22:5   165        TOPO_BUFFER 81Ç^.NOVA_IMPRESSAO := FALSE
22:4   180      END;
22:2   184    END
22:0   184  END;
```

```
22:0   204
23:D     1 PROCEDURE  IMPRESSAO;
23:D     1 VAR
23:D     1    ACABOU : BOOLEAN;
23:0     0 BEGIN
23:1     0   ACABOU := FALSE;
23:1     3   WITH  TOPO_BUFFER @2Ç^  DO
23:2    17   BEGIN
23:3    17     WHILE  (TOPO_BUFFER @2Ç <> NIL )  AND  (NOT ACABOU)  DO
23:4    36     BEGIN
23:5    36       IF  NOVA_IMPRESSAO  THEN  ESCREVE_CABECALHO;
23:5    44       I := 1;
23:5    48       WHILE  (TOPO_BUFFER @2Ç^.LINHAS @IÇ <> 'DISP')
23:5    77                AND  (I < 65)  DO
23:6    85       BEGIN
23:7    85         WRITELN (TOPO_BUFFER @1Ç^.LINHAS @IÇ);
23:7   122         I := I + 1
23:6   125       END;
23:5   132       IF  TOPO_BUFFER @2Ç^.ULTIMO_BUFFER
23:5   144         THEN BEGIN
23:7   149                PPBDT (7, 8);
23:7   153                ACABOU := TRUE;
23:7   156                ESCREVE_FIM_TAREFA;
23:6   158              END;
23:5   158       PPBUFFER (2, 1)
23:4   160     END
23:2   162   END
23:0   164 END;
23:0   182
23:0   182
23:0   182 @ *              ESCALONADOR  DO  SO_GERENTE                  * Ç
23:0   182
23:0   182
24:D     1 PROCEDURE ESCALONADOR;
24:D     1
24:D     1 CONST
24:D     1      NRO_MAX_PROCEDIMENTOS = 7;
24:D     1      NRO_MAX_FI_PROCEDIMENTOS = 4;
24:D     1      MAX_PRIORIDADE = 5;
24:D     1 TYPE
24:D     1     PROCEDIMENTOS = (INICIALIZAR, CARREGA_TAREFA, ENTRADA_DADOS
24:D     1                      SAIDA_DADOS, TERMINO, ROLAMENTO,
24:D     1                      IMPRESSAO);
24:D     1
24:D     1     APONT_BD_PROCEDIMENTO = ^BD_PROCEDIMENTO;
24:D     1     BD_PROCEDIMENTO = RECORD
24:D     1                       NOME_PROCEDIMENTO : PROCEDIMENTOS;
24:D     1                       PRIORIDADE : 1..MAX_PRIORIDADE_PROCEDIMENTO;
24:D     1                       ESTADO : (PRONTO, INATIVO, EXECUTANDO,
24:D     1                                 SUSPENSO);
24:D     1                       PROX_BD_PROCEDIMENTO : APONT_BD_PROCEDIMENTO;
24:D     1                     END;
24:D     1
24:D     1     FILAS_PROCEDIMENTOS = (INAT, SUSP, PRONTOS,
24:D     1                            EXEC);
24:D     1
24:D     1 VAR
24:D     1    TOPO_FILA_PROCEDIMENTOS : ARRAY @FILAS_PROCEDIMENTOSÇ
24:D     1                              OF  APONT_BD_PROCEDIMENTO;
24:D     5    AP_ATUAL, AP_ANTERIOR : APONT_BD_PROCEDIMENTO;
24:D     7    FILA : FILAS_PROCEDIMENTOS;
24:D     8
24:D     8
25:D     1 PROCEDURE  RETIRA_PROCEDIMENTO (P, Q : APONT_BD_PROCEDIMENTOS;
25:D     3                                 FILA : FILAS_PROCEDIMENTOS);
25:0     0 BEGIN
```

```
25:1    0    IF  P = TOPO_FILA_PROCEDIMENTOS @FILA@;
25:1   10      THEN  TOPO_FILA_PROCEDIMENTOS @FILA@ := P^.PROX_BD_PROCEDIM
25:1   24      ELSE  IF P^.PROX_BD_PROCEDIMENTO = NIL
25:2   30              THEN @^.PROX_BD_PROCEDIMENTO := NIL
25:2   37              ELSE @^.PROX_BD_PROCEDIMENTO :=
25:3   44                      @^.PROX_BD_PROCEDIMENTO^.PROX_BD_PROCEDIM
25:0   46 END;
25:0   60
25:0   60
26:D    1 PROCEDURE  INSERE_PROCEDIMENTO (P : APONT_BD_PROCEDINENTO;
26:D    2                                 FILA : FILAS_PROCEDIMENTOS);
26:D    3 VAR
26:D    3    AP : APONT_BD_PROCEDIMENTO;
26:0    0 BEGIN
26:1    0   IF  TOPO_FILA_PROCEDIMENTOS @FILA@ = NIL
26:1   10     THEN TOPO_FILA_PROCEDIMENTOS @FILA@ := P
26:1   23     ELSE BEGIN
26:3   27            AP := TOPO_FILA_PROCEDIMENTOS @FILA@;
26:3   39            WHILE  AP^.PROX_BD_PROCEDIMENTO <> NIL
26:3   41              DO  AP := AP^.PROX_BD_PROCEDIMENTO;
26:3   51            AP^.PROX_BD_PROCEDIMENTO := P
26:2   54          END;
26:1   56   P^.PROX_BD_PROCEDIMENTO := NIL
26:0   59 END;
26:0   76
26:0   76
27:D    1 PROCEDURE  EXECUTA;
27:D    1 VAR
27:D    1    MAIOR_PRIORIDADE : INTEGER;
27:D    2    P1, P2, AP_ATUAL, AP_ANTERIOR : APONT_BD_PROCEDIMENTO;
27:0    0 BEGIN
27:1    0   MAIOR_PRIORIDADE := 0;
27:1    3   P1 := TOPO_FILA_PROCEDIMENTOS @PRONTOS@;
27:1   15   WHILE  P1 <> NIL  DO
27:2   20   BEGIN
27:3   20     IF  MAIOR_PRIORIDADE < P1^.PRIORIDADE  THEN
27:4   26     BEGIN
27:5   26       MAIOR_PRIORIDADE := P1^.PRIORIDADE;
27:5   30       AP_ATUAL := P1;
27:5   33       AP_ANTERIOR := P2
27:4   33     END;
27:3   36     P2 := P1;
27:3   39     P1 := P1^.PROX_BD_PROCEDIMENTO
27:2   40   END;
27:1   45   RETIRA_PROCEDIMENTO (AP_ATUAL, AP_ANTERIOR, PRONTOS);
27:1   50   INSERE_PROCEDIMENTO (AP_ATUAL, EXEC)
27:0   52 END;
27:0   68
28:D    1 PROCEDURE  SUSPENDE;
28:D    1 VAR
28:D    1    AP : APONT_BD_PROCEDIMENTO;
28:0    0 BEGIN
28:1    0   AP := TOPO_FILA_PROCEDIMENTOS @EXEC@;
28:1   12   TOPO_FILA_PROCEDIMENTOS @EXEC@ := NIL;
28:1   23   INSERE_PROCEDIMENTO (AP, SUSP)
28:0   25 END;
28:0   40
28:0   40
29:D    1 PROCEDURE  TERMINA;
29:D    1 VAR
29:D    1    AP : APONT_BD_PROCEDIMENTO;
29:0    0 BEGIN
29:1    0   AP := TOPO_FILA_PROCEDIMENTOS @EXEC@;
29:1   12   TOPO_FILA_PROCEDIMENTOS @EXEC@ := NIL;
29:1   23   INSERE_PROCEDIMENTO (AP, INAT)
29:0   25 END;
```

```
29:0    40
29:0    40
30:0     1 PROCEDURE  APRONTA (P, Q : APONT_BD_PROCEDIMENTO);
30:0     0 BEGIN
30:1     0   RETIRA_PROCEDIMENTO (P, Q, SUSP);
30:1     5   INSERE_PROCEDIMENTO (P, PRONTOS)
30:0     7 END;
30:0    22
30:0    22
31:0     1 PROCEDURE  INICIA (P, Q : APONT_BD_PROCEDIMENTO);
31:0     0 BEGIN
31:1     0   RETIRA_PROCEDIMENTO (P, Q, INAT);
31:1     5   INSERE_PROCEDIMENTO (P, PRONTOS)
31:0     7 END;
31:0    22
24:0    22 FUNCTION
32:0     3  CONDICAO_SATISFEITA (TIPO_PROCEDIMENTO : PROCEDIMENTOS) : BOOL
32:0     0 BEGIN
32:1     0   CASE  TIPO_PROCEDIMENTO  OF
32:1     3     INICIALIZAR       : IF  (TOPO_BDT @1Ç  <>  NIL)  AND
32:2    17                           (EXISTE_TAREFA)  THEN
32:3    23                             CONDICAO_SATISFEITA  :=  TRUE;
32:1    28     CARREGA_TAREFA   : IF  (TOPO_BDT @1Ç  <>  NIL)  AND
32:2    42                           (OP_DISPONIVEL)  THEN
32:3    48                             CONDICAO_SATISFEITA  :=  TRUE;
32:1    53     ENTRADA_DADOS    : IF  TOPO_BDT @3Ç  <>  NIL
32:2    65                           THEN  CONDICAO_SATISFEITA  :=  TRUE;
32:1    74     SAIDA_DADOS      : IF  TOPO_BDT @4Ç  <>  NIL
32:2    86                           THEN  CONDICAO_SATISFEITA  :=  TRUE;
32:1    95     TERMINO          : IF  TOPO_BDT @5Ç  <>  NIL
32:2   107                           THEN  CONDICAO_SATISFEITA  :=  TRUE;
32:1   116     ROLAMENTO        : IF  (TOPO_BDT @6Ç  <>  NIL)  AND
32:2   130                           (TOPO_BUFFER @1Ç  <>  NIL)  THEN
32:3   147                             CONDICAO_SATISFEITA  :=  TRUE;
32:1   152     IMPRESSAO        : IF  (TOPO_BDT @7Ç  <>  NIL)  AND
32:2   166                           (TOPO_BUFFER @1Ç  <>  NIL)  THEN
32:3   183                             CONDICAO_SATISFEITA  :=  TRUE
32:1   183   END
32:0   210 END;
32:0   226
32:0   226
32:0   226
32:0   226
24:0     0 BEGIN
24:1     0   FOR  FILA := INAT  TO  SUSP  DO
24:2    11   BEGIN
24:3    11     AP_ATUAL := TOPO_FILA_PROCEDIMENTOS @FILAÇ;
24:3    22     WHILE  AP_ATUAL  <>  NIL  DO
24:4    27     BEGIN
24:5    27       IF  CONDICAO_SATISFEITA (AP_ATUAL^.NOME_PROCEDIMENTO)
24:5    29         THEN CASE  FILA  OF
24:6    38                INAT  : INICIA (AP_ATUAL, AP_ANTERIOR);
24:6    44                SUSP : APRONTA (AP_ATUAL, AP_ANTERIOR)
24:6    46              END;
24:5    62       AP_ANTERIOR := AP_ATUAL;
24:5    65       AP_ATUAL := AP_ATUAL^.PROX_BD_PROCEDIMENTO
24:4    66     END
24:2    69   END;
24:1    78   IF  TOPO_FILA_PROCEDIMENTOS @EXECÇ^.ESTADO = INATIVO
24:1    88     THEN  TERMINA
24:1    92     ELSE  SUSPENDE;
24:1    98   EXECUTA;
24:1   100   { ATUALIZA_PEDIDO_CCP Ç
24:1   100   FOR I := 1 TO 4 DO
24:2   114   BEGIN
24:3   114     LE_CCP (I, INFO);
```

```
24:3   122       CASE  INFO  OF
24:3   127         21 : PPBDT (2, 3);
24:3   133         22 : PPBDT (2, 4);
24:3   139         23 : BEGIN PPBDT (2, 5); TOPO_BDT 65Ç^.ERRO := FALSE END;
24:3   161         24 : BEGIN PPBDT (2, 5); TOPO_BDT 65Ç^.ERRO := TRUE END
24:3   181       END
24:2   198     END;
24:0   208 END;
24:0   226
 1:0     0 BEGIN
 1:1     0   FIM_PROCESSAMENTO := FALSE;
 1:1    50   CRIA_BUFFERS (TOPO_BUFFER 61Ç);
 1:1    71   CRIA_BD_TAREFA (TOPO_BDT 58Ç);
 1:1    94   REPEAT
 1:2    94     ESCALONADOR;
 1:2    96     LE_TECLADO
 1:1    96   UNTIL FIM_PROCESSAMENTO
 1:0    98 END.
```

```
        .PROC ESCRCXPINFO
        .PUBLIC INIC_CXP, FIM_CXP, INFO, NRO_LINHAS
;
;   GUARDA ENDERECO DE RETORNO
;
        PUL A
        STA A,END_RET
        PUL A
        STA A,END_RET+1
;
;   ESCREVE NA CAIXA POSTAL
;
;
;  PONTEIROS PARA LER   INFO    E
;  COLOCAR A INFORMACAO NA CAIXA POSTAL
;
        LDX FIM_CXP
        INX
        STX FIM_CXP

;
;  AJUSTA O PONTEIRO DE FIM DE CAIXA POSTAL
;
        LDX #INFO
        STX INIC_INFO

;
;  LEITURA
;
VOLTA   CPX #NRO_LINHAS
        BEQ FIM
        LDX INIC_CXP
        CPX FIM_CXP
        BEQ FIM

;
;  ARMAZENA NA CAIXA POSTAL
;
        LDA A,INIC_INFO
        STA A,INIC_CXP
;
;  INCREMENTA PONTEIRO DA LEITURA
;
        LDX INIC_CXP
        INX
        STX INIC_CXP
        LDX INIC_CXP
        INX
        STX INIC_INFO
        BRA VOLTA
;
;    FIM DA LEITURA
;
FIM     LDA A,END_RET+1
        PSH A
        LDA A,END_RET
        PSH A
        RTS
;
;   VARIAVEIS
;
END_RET     .WORD
INIC_INFO   .WORD
    .END
```

```
        .PROC LECXPINFO
        .PUBLIC INIC_CXP, FIM_CXP, INFO
;
;   GUARDA ENDERECO DE RETORNO
;
        PUL A
        STA A,END_RET
        PUL A
        STA A,END_RET+1
;
;   LE A CAIXA POSTAL
;
;
; PONTEIROS PARA LER A CAIXA POSTAL E
; COLOCAR A INFORMACAO EM  INFO
;
        LDX #INFO
        STX INIC_INFO
;
; AJUSTA O PONTEIRO DE FIM DE CAIXA POSTAL
;
        LDX FIM_CXP
        INX
        STX FIM_CXP
;
; LEITURA
;
        LDX INIC_CXP
VOLTA   CPX FIM_CXP
        BEQ FIM
;
; ARMAZENA EM  INFO
;
        LDA A,INIC_CXP
        STA A,INIC_INFO
;
; INCREMENTA PONTEIRO DA LEITURA
;
        LDX INIC_CXP
        INX
        STX INIC_CXP
        LDX INIC_INFO
        INX
        STX INIC_INFO
        BRA VOLTA
;
;   FIM DA LEITURA
;
FIM     LDA A,END_RET+1
        PSH A
        LDA A,END_RET
        PSH A
        RTS
;
;    VARIAVEIS
;
END_RET     .WORD
INIC_INFO   .WORD
        .END
```

```
        .PROC ESCRCCP
         .PUBLIC  NRO_CXP, INFO
;
;   GUARDA O ENDERECO DE RETORNO
;
        PUL A
        STA A,END_RET
        PUL A
        STA A,END_RET+1
;
;   LE  INFORMACAO  DO CCP
;
        LDA A,NRO_CXP
        STA A,INFO
;
;   FIM DA LEITURA
;
        LDA A,END_RET+1
        PSH A
        LDA A,END_RET
        PSH A
        RTS
;
;   VARIAVEIS
;
END_RET  .WORD
        .END
```

```
      .PROC LECCP
       .PUBLIC  NRO_CXP, INFO
;
;   GUARDA O ENDERECO DE RETORNO
;
        PUL A
        STA A,END_RET
        PUL A
        STA A,END_RET+1
;
;   ESCREVE NO CCP
;
        LDA A,INFO
        STA A,NRO_CXP
;
;   FIM DA LEITURA
;
        LDA A,END_RET+1
        PSH A
        LDA A,END_RET
        PSH A
        RTS
;
;   VARIAVEIS
;
END_RET .WORD
        .END
```

MÓDULO MONITOR DO PROCESSADOR DE COMUNICAÇÃO

```
PAGE 001  VPI      .SA=0

00001    VPI : PROC
00002    DCL 1 PIATDA DEF $F6, ! RECEBE PEDIDO DE INTERRUPCAO
00003          2 OP1 BIT (1),
00004          2 OP2 BIT (1),
00005          2 OP3 BIT (1),
00006          2 OP4 BIT (1)
00007    DCL NROP BIN (1) EXTERNAL
00008    !
00009      IF OP1 EQ 0 THEN
00010      DO NROP = 1; CALL TARPED  END ! OP1 PEDE INTERRUPCAO
00011      IF OP2 EQ 0 THEN
00012      DO NROP = 2; CALL TARPED  END ! OP2 PEDE INTERRUPCAO
00013      IF OP3 EQ 0 THEN
00014      DO NROP = 3; CALL TARPED  END ! OP3 PEDE INTERRUPCAO
00015      IF OP4 EQ 0 THEN
00016      DO NROP = 4; CALL TARPED  END ! OP4 PEDE INTERRUPCAO
00017      CALL TSTCCP
00018      RETURN
00019    END  ! VARRE PEDIDOS DE INTERRUPCAO
```

```
PAGE 001  TSTCCP  .SA#0

00001    TSTCCP : PROC
00002    DCL NROP BIN (1) EXTERNAL,
00003        TAREFA BIN (1),
00004        ENDCCP BIN (2),
00005        AUX BIN (1) BASED
00006    DCL CCPS (4) BIN (2)
00007          DEF $DFFC   ! PRIMEIRO CCP, OUTROS ($DFFD, $DFFE, $DFFF)
00008    !
00009      NROP = 1
00010      DO  WHILE NROP LE 4
00011        ENDCCP =CCPS (NROP)
00012        TAREFA = ENDCCP -> AUX
00013        IF TAREFA EQ 1 THEN CALL NI    ! NOVAS INSTRUCOES
00014                       ELSE
00015          IF TAREFA EQ 2 THEN CALL UI       ! ULTIMAS INSTRUCOES
00016                         ELSE
00017            IF TAREFA EQ 3 THEN CALL DT        ! DESCONTINUA TAREFA
00018                           ELSE
00019              IF TAREFA EQ 4 THEN CALL ED          ! ENTRADA DE DADOS
00020      END
00021      CALL VPI
00022      RETURN
00023    END  ! TESTA CCP PARA ATENDER O GERENTE
```

```
00001    TARPED : PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL PIAUDB BIN (1) DEF SFA, ! RECEBE DADO
00004        IOPVEL BIN (1),
00005        IOPNOV BIN (1)
00006    !
00007      CALL INTOP
00008      IOPVEL = 0
00009      IOPNOV = 0
00010      CALL PERCOM (IOPVEL, IOPNOV)
00011      IF PIAUDB EQ 21 THEN CALL PD              ! PEDE DADOS
00012                      ELSE
00013       IF PIAUDB EQ 22 THEN CALL R              ! RESULTADOS
00014                       ELSE
00015         IF PIAUDB EQ 23 THEN CALL TN           ! TERMINO NORMAL
00016                         ELSE
00017           IF PIAUDB EQ 24 THEN CALL TA         ! TERMINO ANORMAL
00018      RETURN
00019    END  ! TAREFA PEDIDA POR OPERADOR
```

```
00001    PD : PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL CCPS (4) BIN (1) DEF $DFFC
00004    !
00005       CCPS (NROP) = 21
00006       RETURN
00007    END  ! FEDE DAD0S
```

```
PAGE 001  R        „SA¤O

00001    R : PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL CCPS (4) BIN (2) DEF $DFFC
00004    !
00005      CALL INTOP            ! ACEITO INTERRUPCAO DO OPERADOR (NROP)
00006      CALL TRFOPC (0)       ! TRANSFERE RESULTADOS DO OPERADOR
00007                            ! AO PROCESSADOR DE COMUNICACAO
00008      CCPS (NROP) = 2¤
00009      RETURN
00010    END ! RESULTADOS
```

```
00001    TN:PROC (NROP)
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL CCPS (4) BIN (1) DEF $DFFC
00004    !
00005      CCPS (NROP) = 23 ! TERMINO NORMAL
00006      RETURN
00007    END . ! TERMINO NORMAL
```

```
PAGE 001   TA        .SA:0

00001     TA : PROC (NROP)
00002     DCL NROP BIN (1) EXTERNAL
00003     DCL CCPS (4) BIN (1) DEF $DFFC
00004     !
00005       CCPS (NROP) = 24      ! TERMINO ANORMAL
00006       RETURN
00007     END TERMINO ANORMAL
```

```
00001   NI : PROC
00002   DCL NROP BIN (1) EXTERNAL
00003   DCL CXPS (4) BIN (1)
00004         DEF $DEEC,    ! PRIMEIRA CXP, OUTRAS ($DF2C, $DF4C, $DFAC)
00005       CCPS (4) BIN (1) DEF $DFFC,  ! PRIMEIRO CPP
00006       ENDCXP BIN (2), IOPNOV BIT (1), IOPVEL BIT (1),
00007       AUX BIN (1) BASED,
00008       TABMP (4,2) BIN (1) EXTERNAL,
00009       PROTO BIN (1), PROT1 BIN (1), I BIN (1),
00010       PIAUDA BIN (1) DEF $F8
00011   !
00012     PIAUDA = 1
00013     CALL INTOP    ! AVISO QUE VOU  ENVIAR DADO
00014     ENDCXP = CXPS (NROP)
00015     PROTO = ENDCXP -> AUX
00016     ENDCXP = ENDCXP + 1
00017     PROT1 = ENDCXP -> AUX
00018     IF (PROTO EQ 2) AND (PROT1 EQ 2) THEN ! CARREGA NRO. DA TAR.
00019     DO                                    ! E O NUMERO DO PROGRAMA
00020       ENDCXP = ENDCXP + 1
00021       TABMP (NROP ,1) = ENDCXP -> AUX        ! NUMERO DA TAREFA
00022       ENDCXP = ENDCXP + 1
00023       TABMP (NROP, 2) = ENDCXP -> AUX      ! NUMERO DO PROGRAMA
00024       IOPNOV = 0
00025       IOPVEL = 0
00026       I = 1
00027       DO WHILE I LT 5
00028         IF I NE NROP THEN
00029           IF TABMP (I, 1) EQ PROT THEN
00030           DO
00031             CALL INTOP (I)
00032             CALL PERCOM (I, IOPNOV, IOPVEL)
00033             PIAUDA = ENDCXP -> AUX  ! NUMERO DO PROGRAMA
00034             CALL PERCOM (I, IOPNOV, IOPVEL)
00035               PIAUDA = NROP          ! NUMERO DO OPERADOR
00036             I = I + 1
00037           END
00038       END
00039       CALL TRFPCO (12)                    ! TRANSFERE STATUS
00040     END
00041                     ELSE
00042     CALL TRFPCO (0)
00043     CCPS (NROP) = 0                      !CAIXA POSTAL LIDA
00044     RETURN
00045   END  ! NOVAS INSTRUCOES
```

```
PAGE 001  UI       .SA=0

00001    UI : PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL CCPS (4) BIN (1)  DEF $DFFC,    ! CONTROLE DE CAIXA POSTAL
00004         PIAUDA BIN (1)  DEF $F8          ! ENVIA DADO
00005    !
00006      PIAUDA = 2
00007      CALL INTOP
00008      CALL TRFPCO (0)
00009      CCPS (NROP) = 0
00010      RETURN
00011    END    ! ENVIA ULTIMAS INSTRUCOES
```

```
PAGE 001   ED        „SA=0

00001    ED : PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL CCPS (4) BIN (1) DEF ($DFFC), ! CONTROLES DE CXP
00004         PIAUDA BIN (1) DEF ($00F8)      ! ENVIA DADO
00005    !
00006      PIAUDA = 4
00007      CALL INTOP
00008      CALL TRFPCO (0)
00009      CCPS (NRO) = 0
00010      RETURN
00011    END ENTRADA DE DADOS
```

```
00001   PERCOM : PROC (IOPVEL,IOPNOV)
00002   DCL IOPVEL BIN (1),
00003       IOPNOV BIN (1)
00004   DCL NROP BIN (1) EXTERNAL,
00005       AUXIOP BIN (1),
00006       AUXVEL BIN (1),
00007       AUXNOV BIN (1)
00008   DCL 1 PIATDB DEF %F6,
00009         2 IOP1 BIN (1),
00010         2 IOP2 BIN (1),
00011         2 IOP3 BIT (1),
00012         2 IOP4 BIT (1)
00013   !
00014     DO WHILE IOPNOV EQ IOPVEL
00015       IF NROP EQ 1 THEN
00016       DO
00017         AUXIOP = IOP1
00018         IF AUXIOP NE IOPVEL THEN IOPNOV = IOP1
00019       END
00020                    ELSE
00021       IF NROP EQ 2 THEN
00022       DO
00023         AUXIOP = IOP2
00024         IF AUXIOP NE IOPVEL THEN IOPNOV = IOP2
00025       END
00026                    ELSE
00027       IF NROP EQ 3 THEN
00028       DO
00029         AUXIOP = IOP3
00030         IF AUXIOP NE IOPVEL THEN IOPNOV = IOP3
00031       END
00032                    ELSE
00033       IF NROP EQ 4 THEN
00034       DO
00035         AUXIOP = IOP4
00036         IF AUXIOP NE IOPVEL THEN IOPNOV = IOP4
00037       END
00038     END
00039     IOPVEL = IOPNOV
00040     RETURN
00041   END  ! PERMITE COMUNICACAO
```

```
00001    INTOP # PROC
00002    DCL NROP BIN (1) EXTERNAL
00003    DCL PIATDA BIN (1)  DEF $F4 ! ENVIA INTERRUPCAO
00004    !
00005       IF NROP EQ 1 THEN PIATDA = $7F
00006                    ELSE
00007         IF NROP EQ 2 THEN PIATDA = $BF
00008                      ELSE
00009           IF NROP EQ 3 THEN PIATDA = $DF
00010                        ELSE
00011             IF NROP EQ 4 THEN PIATDA = $EF
00012       RETURN
00013    END ! ENVIA INTERRUPCAO PARA O OPERADOR (NROP)
```

```
00001   TRFOPC : PROC (J)
00002   DCL J BIN (2),
00003       NROP BIN (1) EXTERNAL
00004   DCL PIAUDB  BIN (1) DEF $FA,                  ! RECEBE DADO
00005       CXPS (4) BIN (1) DEF $DEEC,  ! PRIMEIRA CAIXA POSTAL
00006       AUX BIN (1) BASED,
00007       INICXP BIN (2), FIMCXP BIN (2),
00008       IOPNOV BIT (1), IOPVEL BIT (1)
00009   !
00010     IOPNOV = 0
00011     IOPVEL = 0
00012     INICXP = CXPS (NROP) + J
00013     FIMCXP = CXPS (NROP + 1)
00014     DO WHILE INICXP LT FIMCXP
00015       CALL PERCOM (IOPVEL, IOPNOV)
00016       INICXP -> AUX = PIAUDB
00017       INICXP = INICXP + 1
00018     END
00019     RETURN
00020   END ! TRANSFERE DADO DO OPERADOR PARA O PROCESSADOR COMUNICACAO
```

```
00001    TRFPC0 : PROC (J)
00002    DCL J BIN (2)
00003    DCL NROP BIN (1) EXTERNAL
00004    DCL PIAUDA  BIN (1) DEF $F0,                    ! ENVIA DADO
00005        CXPS (4) BIN (2)  DEF $DEEC,    ! PRIMEIRA CAIXA POSTAL
00006        AUX BIN (1) BASED,
00007        INICXP BIN (2), FIMCXP BIN (2),
00008        IOPNOV BIN (1), IOPVEL BIN (1)
00009    !
00010      IOPVEL = 0
00011      IOPNOV = 0
00012      IF J EQ 12 THEN
00013      DO
00014        INICXP = CXPS (NROP) + 2
00015        FIMCXP = INICXP + 12
00016      END
00017                    ELSE
00018      DO
00019        INICXP = CXPS (NROP)
00020        FIMCXP = CXPS (NROP + 1)
00021      END
00022      DO  WHILE  INICXP LT FIMCXP
00023        CALL PERCOM (IOPVEL, IOPNOV)
00024        PIAUDA =  INICXP -> AUX
00025        INICXP = INICXP + 1
00026      END
00027      RETURN
00028    END  ! TRANSFERE DO PROCESSADOR COMUNICACAO AO OPERADOR
```

```
00001   PROGPIAS : PROC ( PIA, PORTO, EOUS )
00002   ! PROGRAMA O PORTO DA PIA COMO E/S
00003
00004   DCL PIA CHAR (1),
00005       PORTO CHAR (1),
00006       EOUS CHAR (1)
00007
00008   DCL 1 PIAXPY BASED,
00009         2 DDPR BIN (1), ! DATA DIRECTION/PERIPHERAL REGISTER
00010           2 CTRL BIN (1) ! CONTROL REGISTER
00011
00012   DCL ENDERD BIN (2),
00013       ENDERC BIN (2),
00014       MASC BIN (1)
00015
00016   IF PIA EQ 'T' THEN ENDERC = $F4 ELSE
00017     IF PIA EQ 'U' THEN ENDERC = $F8
00018   IF PORTO EQ 'B' THEN ENDERC = ENDERC + 2
00019   ENDERD = ENDERD + 1
00020   IF EOUS EQ 'S'
00021     THEN MASC = $FF ! SAIDA
00022     ELSE MASC = $00 ! ENTRADA
00023   ENDERD -> CTRL = $00 ! ACESSO AO DATA DIRECTION
00024   ENDERC -> DDPR = MASC ! PROGRAMA COMO E/S
00025   ENDERC -> CTRL = $04 ! OBTEM ACESSO AO PERIPHERAL REGISTER
00026   IF EOUS EQ 'S' THEN
00027     THEN ENDERC -> DDPR = $FF
00028   RETURN
00029   END PROGRAMA AS PIAS
```

MÓDULO NÚCLEO DO OPERADOR

```
00001   ENVIA : PROC (DESTINO, FONTE)
00002   DCL DESTINO BIN (2),
00003       FONTE BIN (2),
00004       PIAJDB  DEF ($2002)
00005   DCL I BIN (1),
00006       NROP BIN (1)
00007   !
00008     CALL SALVACONTEXTO
00009     TABMP (NROPLOCAL, 3) = 3    ! RESULTADOS
00010     NOE = NOE + 1               ! NUMERO ORDEM PARA ENVIO
00011     IF  DESTINO E@ 0  THEN      ! MENSAGEM PARA O GERENTE?
00012     DO
00013       CALL PDINTPC
00014   $   CWAI                      ! ESPERA INTERRUPCAO DO PC
00015       PIAJDB = 22
00016       CALL PDINTPC
00017   $ CWAI
00018       CALL TRFOPC
00019     END
00020                           ELSE
00021     DO
00022       PROCTAR (DESTINO, I, NROP)       ! PROCURA TAREFA
00023       IF  I E@ 4  THEN            ! TAREFA DESTINO ESTA NA MP?
00024       THEN DO
00025              CALL FORMATA (NROP)
00026              CALL TRFOPOP (NROP)   ! ENVIA RESULTADOS P/ NROP
00027            END
00028       ELSE DO
00029   $            CWAI                ! ESPERA INTERRUPCAO DO PC
00030            END
00031     END
00032     TABMP (NROPLOCAL, 3) = 1
00033     CALL RESTAURACONTEXTO
00034     RETURN
00035   END  ATENDE A CHAMADA DA PRIMITIVA ENVIA DO PROGRAMA DO USUARIO
```

```
00001    RECEBE : PROC (FONTE)
00002    DCL FONTE BIN (2)
00003    DCL I BIN (1),
00004        TABMENS (4) BIN (1),
00005        NROP BIN (1),
00006        ENDE BIN (2)
00007    !
00008      CALL SALVACONTEXTO
00009      TABMP (NROPLOCAL, 3 ) = 2
00010      NOR = NOR + 1
00011      IF  FONTE  EQ  0  THEN
00012      DO
00013        CALL PDINTPC
00014        PIAJDB = 21
00015    $   CWAI
00016        CALL TRFPCOP
00017      END
00018                          ELSE
00019        DO  I = 1  WHILE  I LT 5   ! VE SE JA CHEGOU MENSAGEM
00020          IF  TABMENS (I) EQ 1 ! JA EXISTE MENSAGEM PARA LER?
00021          THEN DO
00022                 CALL COPMENS (I, ENDE) ! ENCONTROU MENSAGEN
00023                 I = 6
00024               END
00025          ELSE I = I + 1
00026        END
00027        IF  I EQ 5  THEN              ! NAO EXISTE MENSAGEM P/ COPIAR
00028        DO
00029          PROCTAR (FONTE, I, NROP)
00030          IF  I EQ 6  THEN            ! EXISTE ESTA TAREFA NA MP?
00031          DO
00032    $       CWAI                      ! FICA ESPERANDO MENSAGEN DO NO
00033            CALL TRFNOP
00034          END
00035                        ELSE
00036          DO
00037           TABMP (NROPLOCAL,3) = 4  ! FALTA TAREFA, ERRO EXEC.
00038            CALL PDINTPC
00039    $       CWAI                      ! ESPERA INTERRUPCAO DO PC
00040            PIAJDB = 24       ! TERMINO ANORMAL
00041    $       JMP  INICNOP      ! INICIO NUCLEO DO OPERADOR
00042          END
00043        END
00044      TABMP (NROPLOCAL, 3) = 1
00045      CALL RESTAURACONTEXTO
00046    RETURN
00047    END  ATENDE A CHAMADA DA PRIMITIVA RECEBE DA TAREFA
```

```
00001   RECEBECONDICIONAL : PROC (FONTE, ENDE)
00002   DCL FONTE BIN (2),
00003       ENDE BIN (2),
00004       I BIN (2)
00005   !
00006     CALL SALVACONTEXTO
00007     TABMP (NROPLOCAL, 3) = 2
00008     I = 1
00009     DO WHILE I LT 5
00010       IF TABMENS (I) EQ FONTE
00011         THEN DO
00012                NOR = NOR + 1
00013                CALL COPMENS (I, END)
00014                I = 6
00015              END
00016         ELSE
00017              I = I + 1
00018     END
00019     TABMP (NROPLOCAL, 3) = 1
00020     CALL RESTAURACONTEXTO
00021     RETURN
00022   END ATENDE A CHAMADA DA PRIMITIVA RECEBECONDICIONAL DA TAREFA
```

```
00001    SALVACONTEXTO : PROC
00002    $ .  PSHS A,B,CC
00003    $    PSHS DP,D
00004    $    PSHS X,Y
00005    $    PSHS U,PC
00006    RETURN
00007    END     SALVA CONTEXTO DA TAREFA
```

```
00001    RESTAURACONTEXTO : PROC
00002    $    PULS A,B,CC
00003    $    PULS DP,P
00004    $    PULS X,Y
00005    $    PULS U,PC
00006    RETURN
00007    END
```

```
00001    PDINTPC # PROC          ! PEDE INTERRUPCAO AO PROC. DE COMUN.
00002    DCL PIAJCA2 DEF ($2001),
00003        SENDREQ INIT ($00)
00004        PIAJCA2 = SENREQ
00005    RETURN
00006    END
```

```
PAGE 001   TRFOPOP .SA:0

00001    TRFOPOP : PROC
00002    DCL PIAIDA DEF ($2800)
00003    $ LDA X
00004    $ STA PIAIDA
00005    $ CWAI
00006    $ LDA Y
00007    $ STA PIAIDA
00008    $ CWAI
00009    $ LDA NOE
00010    $ STA PIAIDA
00011    $ CWAI
00012    $ LDA TD
00013    $ STA PIAIDA
00014    $ CWAI
00015    $ LDA TF
00016    $ STA PIAIDA
00017    $ CWAI
00018    $ LDA RF
00019    $ STA PIAIDA
00020    $ CWAI
00021       I = 1
00022       DO WHILE I LT 65
00023    $ PULS A
00024    $ STA PIAIDA
00025    $ CWAI
00026       I = I + 1
00027       END
00028    $ LDA PAR
00029    $ STA PIAIDA
00030    RETURN
00031    END
```

```
PAGE 001  FORMATA .SA=0

00001    FORMATA : PROC (NROP)
00002    DCL NROP BIN (2)
00003        X BIN (1) EXTERNAL,
00004        Y BIN (1) EXTERNAL,
00005        PAR BIN (1) EXTERNAL,
00006        TD BIN (1) EXTERNAL,
00007        TF BIN (1) EXTERNAL,
00008        PF BIN (1) EXTERNAL
00009    ! LOCALIZA O OPERADOR NA MALHA DE PROCESSAMENTO
00010      IF NROP = 1
00011      THEN  DO
00012                X = 0
00013                Y = 0
00014              END
00015        ELSE IF NROP = 2
00016             THEN DO
00017                     X = 0
00018                     Y = 1
00019                  END
00020             ELSE IF NROP = 3
00021                  THEN DO
00022                          X = 1
00023                          Y = 0
00024                       END
00025                  ELSE DO
00026                          X = 1
00027                          Y = 1
00028                       END
00029    ! TAREFA DESTINO
00030      TD = DESTINO
00031    ! TAREFA FONTE
00032      TF = TABMP (NROPLOCAL, 2)
00033    ! PROGRAMA FONTE
00034      PF = TABMP ( NROPLOCAL, 1)
00035    !INFORMACAO DE VALIDADE DE MENSAGEM
00036      PAR = NO * TD
00037      PAR = PAR + TF
00038      PAR = PAR + PF
00039    RETURN
00040    END   CRIA OS DADOS DE CONTROLE DA MENSAGEM
```

```
PAGE 001  COPMENS „SA#0

00001    COPMENS # PROC (I, ENDE)
00002    DCL I BIN (1),
00003        ENDE BIN (2)
00004    DCL NO BIN (1) EXTERNAL,
00005        NOR BIN (1) EXTERNAL
00006        IF NO = NOR
00007          THEN DO
00008                 I = 1
00009                 DO WHILE I LT 65
00010    $       LDA I
00011    $       PSHS A
00012                   I = I +1
00013                 END
00014                 ENDE = 1
00015             END
00016           ELSE ENDE = 0
00017    RETURN
00018    END
```

```
00001    TRFPCOP : PROC
00002    DCL   I BIN (1),
00003          PIAJDA  DEF ($2000)
00004    !
00005    I = 1
00006    DO  WHILE  I  LT  65
00007    $ LDAA PIAJDA
00008    $ PSHS A
00009      I = I + 1
00010    END
00011    RETURN
00012    END TRANSFERE  DADOS DO PROC. DE COMUN. AO OPERADOR
```

```
00001    PROCPROC : PROC (DESTINO, I, NROP)
00002    DCL DESTINO BIN (2),
00003        I BIN (2),
00004        NROP BIN (2)
00005    !
00006      DO  I = 1  WHILE  I  LT  5
00007      IF TABMP (2, I) EQ DESTINO
00008           THEN DO
00009                  NROP = I
00010                  I = 6
00011                END
00012          ELSE I = I + 1
00013      END
00014    RETURN
00015    END  PROCURA TAREFA NA MALHA DE PROCESSAMENTO
```

```
00001    NI * PROC (ENDALOC)
00002    DCL ENDALOC BIN (2),
00003        I BIN (2)
00004    DCL PIAJDA DEF ($2000)
00005       I = 1
00006       DO WHILE I LT 65
00007    $     LDA PIAJDA
00008    $     STA ENDALOC
00009          ENDALOC = ENDALOC +1
00010          I = I + 1
00011       END
00012    RETURN
00013    END
```

```
00001    UI : PROC (ENDALOC)
00002    DCL ENDALOC BIN (2)
00003    DCL I BIN (2),
00004        PIAJDA DEF ($2000)
00005    !
00006      I = 1
00007      DO WHILE I LT 65
00008    $    LDA PIAJDA
00009    $    STA ENDALOC
00010        ENDALOC = ENDALOC + 1
00011        I = I + 1
00012      END
00013      ENDALOC = $2050
00014    $ JMP $2050          ! SALTA PARA O INICIO DA TAREFA
00015    RETURN
00016    END
```

```
00001    TERMINO # PROC
00002    DCL  PIAJDB DEF ($2002)
00003    !
00004      TABMP (NROPLOCAL, 3) = 5
00005      CALL PDINTPC
00006    $ CWAI                              ! ESPERA INTERRUPCAO DO PC
00007      PIAJDB = 23
00008    $  CWAI
00009    END
```