PROPOSTA DE UM SISTEMA DE EXECUÇÃO PARA A LINGUAGEM DE PROGRAMAÇÃO ADA

ROGÉRIO DRUMMOND BURNIER PESSÔA DE MELLO FILHO

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO

CAMPINAS - SÃO PAULO
BRASIL

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

COORDENAÇÃO DOS CURSOS DE PÓS-GRADUAÇÃO

AUTORIZAÇÃO PARA QUE A UNICAMP POSSA FORNECER, A PREÇO DE CUSTO, CÓPIAS DA TESE A INTERESSADOS

Nome do Aluno: Rogério Drummond Burnier Pessôa de Mello Filho
Nº de Identificação: 786052
Endereço para Correspondência: IMECC- UNICAMP, 13100 Campinas SP
Curso: Mestrado em Ciência da Computação
Nome do Orientador: Tomasz Kowaltowski
Título da Dissertação ou Tese: Proposta de um Sistema de Execução para a Linguagem de Programação ADA

Data proposta para a Defesa: 22/08/1980

(O Aluno deverá assinar um dos 3 ítens abaixo)

1) Autorizo a Universidade Estadual de Campinas a partir desta data, a fornecer, a preço de custo, cópias de minha Dissertação ou Tese a interessados.

18/8/80 — Rogério Drummond Burnier Pessôa de Mello Filho

Data — assinatura do aluno

2) Autorizo a Universidade Estadual de Campinas, a fornecer, a partir de dois anos após esta data, a preço de custo, cópias de minha Dissertação ou Tese a interessados.

__/__/__ — _____

Data — assinatura do aluno

3) Solicito que a Universidade Estadual de Campinas me consulte, dois anos após esta data, quanto à minha autorização para o fornecimento de cópias de minha Dissertação ou Tese, a preço de custo, a interessados.

__/__/__ — _____

Data — assinatura do aluno

DE ACORDO

Tomasz Kowaltowski

Orientador

PROPOSTA DE UM SISTEMA DE EXECUÇÃO PARA A LINGUAGEM DE PROGRAMAÇÃO ADA

ROGÉRIO DRUMMOND BURNIER PESSÔA DE MELLO FILHO

Dissertação apresentada no Instituto de Matemática, Estatística e Ciência da Computação como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

CAMPINAS, AGOSTO DE 1980

À minha esposa

e

aos meus pais

RESUMO

A linguagem de programação ADA é uma linguagem muito recente que tem como objetivo principal ser uma linguagem de uso geral, podendo ser utilizada como ferramenta para as mais diversas aplicações; ser uma linguagem segura, garantindo uma razoável confiabilidade nos sistemas escritos nesta linguagem.

As características principais da ADA são: programação modular, tratamento de situações excepcionais pelo próprio programa e execução paralela.

O presente trabalho descreve um sistema de execução para esta linguagem. Este sistema tem por objetivo ser independente de máquina, possibilitando assim fácil implementação em máquinas reais. Para tal foi desenvolvida uma máquina hipotética, chamada MEADA - Máquina de Execução para a ADA.

A ênfase principal do projeto foi nos aspectos de segurança e eficiência.

Dada a extensão deste projeto, o trabalho apresenta somente as soluções das partes mais complexas, tais como: a implementação de tarefas e procedimentos, do tratamento das exceções e das entradas.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

A linguagem de programação ADA [IC79a] é uma linguagem muito recente. Ela foi projetada para cobrir uma ampla gama de aplicações,oferecendo recursos de programação desde os encontrados em linguagens clássicas como o ALGOL e PASCAL, até os encontrados somente em linguagens de uso especializado ou experimental.

Além de ser uma linguagem de uso geral, a ADA possibilita um razoável nível de segurança de programação, permitindo assim que os sistemas nela escritos sejam, no que depende da linguagem, bastante confiáveis.

Dentre suas características menos comuns pode-se citar:a possibilidade de programação modular e de compilação em separado de unidades de programação , recursos para execução em paralelo, tratamento de situações excepcionais pelo próprio programa, determinação da representação física de dados e acesso aos recursos da máquina. As duas últimas tornam a ADA conveniente para programação de sistemas.

Neste trabalho pressupõe-se o conhecimento dessa linguagem, como descrita em [IC79a&b].

A tradução de programas em linguagem de alto nível para a linguagem de uma máquina é quase sempre um trabalho bastante complexo. Os compiladores, ou seja, os programas que fazem estas traduções são em geral bastante extensos.

A tradução de um programa de uma linguagem para outra consiste em analisá-lo sob três aspectos: léxico, sintático e semântico.

A análise léxica consiste em determinar as unidades básicas do programa, ou seja, os átomos, e é em geral uma tarefa bastante simples.

A análise sintática consiste em determinar se o programa segue as regras gramaticais que definem a linguagem e revelar a sua estrutura.

A análise semântica consiste em atribuir significados às construções do programa, traduzindo cada uma destas construções para a linguagem de uma determinada máquina.

A tradução das construções de uma linguagem de alto nível para a linguagem de uma máquina pode ser bastante complicada. Dada a diferença de nível destas linguagens, cada construção da linguagem de alto nível pode corresponder a um grande número de instruções do computador.

A determinação da tradução para cada construção de uma linguagem pode ser grandemente simplificada através do uso de uma linguagem intermediária.

As construções da linguagem de alto nível seriam primeiro traduzidas para a linguagem intermediária e, posteriormente, cada instrução da linguagem intermediária para as instruções do computador.

Esta linguagem intermediária deve ser tal que facilite a tradução das construções da linguagem de alto nível. Por outro lado, ela não deve ser tão comple-

xa quanto a linguagem de alto nível, de forma que sua tradução para a linguagem do computador seja uma tarefa mais simples.

Se esta linguagem intermediária é projetada de forma a não depender de características particulares de uma máquina, então soma-se mais uma vantagem à sua utilização. Uma vez projetada a linguagem intermediária e um compilador que traduza a linguagem de alto nível para esta linguagem intermediária, este compilador pode ser adaptado para qualquer computador, com relativamente pouco esforço.

Esta linguagem intermediária pode ser vista como a linguagem de uma máquina hipotética. Esta máquina hipotética é comumente chamada de sistema de execução para a linguagem de alto nível ("run-time system").

O objetivo deste trabalho é a construção de um sistema de execução para a linguagem ADA, independente de máquina e com características de portabilidade.

Para tal foi desenvolvida uma máquina hipotética, chamada MEADA - Máquina de Execução para a ADA. A ênfase principal dada à MEADA foi nos aspectos de segurança e eficiência.

Dada a extensão deste projeto, o presente trabalho apresenta somente as partes mais complexas da MEADA. Estas partes são interdependentes e compreendem a implementação de: paralelismo, subprogramas , entradas e tratamento das exceções.

A implementação de expressões e declarações é superficialmente descrita, mas o bastante para que as outras partes sejam compreendidas.

Omitiu-se a discussão dos comandos *if*, *loop*, *case*, *goto*, *exit* e de atribuição, bem como de pacotes, cuja implementação é bem conhecida.

As unidades genéricas também não são cobertas e a sua implementação pode ser bastante simples através da macro expansão de seu código.

As representações físicas das especificações não foram incluídas no projeto, pois são dependentes de máquina.

Finalmente, entrada e saída estão definidas dentro da linguagem e exigem apenas rotinas de suporte, dependentes do sistema operacional de cada máquina.

A ADA pode ser implementada em máquinas de um ou mais processadores. O sistema de execução descrito neste trabalho foi projetado para máquinas com um único processador.

Note-se que neste trabalho não são tratadas nem a organização do compilador, nem a do ligador, e que são partes indispensáveis numa implementação real.

Supõe-se apenas que o compilador tem acesso à toda informação necessária, o que normalmente implica num compilador de dois passos. Supõe-se também que o compilador fornece todas as informações ao ligador, que deve unir todas as unidades de compilação num único módulo de execução.

A literatura sobre a ADA, além de [IC79a] e [IC79b], resume-se principalmente à apreciações e críticas da linguagem.

O único trabalho referente aos problemas de implementação do sistema de execução ao qual teve-se acesso foi [TA80]. Neste trabalho é feita uma análise crítica das sugestões para implementação da ADA apresentadas em [IC79b]. Os mesmos problemas foram descobertos ao desenvolver a MEADA, se bem que as soluções adotadas foram em geral diferentes.

A literatura sobre sistemas de execução não é muito vasta e não trata de muitos dos problemas encontrados na implementação da ADA.

Dentre as obras consultadas citam-se: [RA64], [PR75], [KO79], [OR73],[NO74], [PA73], [SA79] e [WI75].

Este trabalho está subdividido em três capítulos e dois apêndices. o capítulo II contém a descrição da MEADA e é o capítulo principal deste trabalho. O capítulo III é dedicado às conclusões sobre o sistema proposto. No apêndice I são descritas as instruções e primitivas da MEADA.Dada a interdependência das seções do capítulo II e a necessidade de inúmeras abreviações, foi incluido o apêndice II, que contém um dicionário de abreviações e que deve facilitar a leitura do capítulo II.

CAPÍTULO II

S I S T E M A D E E X E C U Ç Ã O

1 - INTRODUÇÃO

Neste capítulo é descrito o sistema de execução proposto para a ADA. Na seção 2 é apresentada a notação utilizada neste capítulo. Na seção 3 é introduzida a estrutura geral da MEADA e seu funcionamento. As seções 4 e 5 discorrem brevemente sobre a implementação das declarações e expressões. Nas seções 6,7, 9 e 10, são descritas, respectivamente, a implementação do tratamento das exceções, das tarefas, das entradas que são a forma de comunicação entre as tarefas, e finalmente dos subprogramas declarados na parte visível de uma tarefa. A implementação dos comandos *delay* e *abort* é apresentada nas seções 8 e 11, respectivamente.

Na descrição do sistema de execução sacrifica-se pequenos ganhos de eficiência em favor de uma explicação mais clara, concisa e uniforme.

O sistema de execução proposto, é baseado em registros de ativação (RA) associados às unidades de programa: tarefas,subprogramas,blocos etc. Os registros de ativação utilizados para a implementação da ADA são ilustrados na figura 1. Embora não se discuta a implementação de blocos e subprogramas locais (subprogramas que não são declarados na parte visível de uma tarefa), os registros de ativação necessários para estas unidades são também ilustrados na figura 1.

Supõe-se que cada campo de um registro de ativação ocupa uma palavra de memória, o que não é necessariamente verdadeiro numa implementação real.

Figura 1 - Registros de Ativação.

A MEADA é descrita em termos de suas instruções e primitivas. Estas últimas devem ser vistas como microinstruções. As instruções podem ou não ser interrompidas durante sua execução. As primitivas chamadas por uma instrução podem ou não ser interrompidas, dependendo da natureza da instrução que as chamou.

2 - NOTAÇÃO

As instruções da MEADA são referidas por mnemônicos de quatro caracteres (letras maiúsculas ou dígitos).

Cada instrução pode ter zero ou mais parâmetros, que são separados por vírgulas e seguem o mnemônico da instrução.

Uma barra inclinada (/) indica o início de um comentário, que se estende até o fim da linha.

As instruções podem ser rotuladas. Estes rótulos são compostos de letras ou dígitos e seguidos de dois pontos (:). O primeiro caracter de um rótulo é sempre a letra R.

Exemplo: RB: CRCT *1* /carrega constante

Uma seqüência de instruções que traduz uma construção X da ADA é referida por "código gerado para X" ou simplesmente "código para X".

Uma seqüência de instruções pode ser abreviada por mnemômicos compostos de letras minúsculas.

Exemplo: caa /código de alternativa *accept*

A letra c (minúscula) abrevia qualquer instrução da MEADA.

Uma ou mais instruções entre colchetes ([...]) indica a ocorrência ou não destas instruções.

Um fecha-colchete seguido do sinal * indica que a instrução ou instruções entre os colchetes podem ocorrer zero ou mais vezes.

Se o fecha-colchete é seguido do sinal + então estas instruções podem ocorrer uma ou mais vezes.

Exemplo: Código para um Comando *if* com Cláusula *else*.

```
        [c]+        /código para expressão booleana
        DSVF   R1   /desvia se falso para R1
        [c]*        /código para os comandos do then
        DSVS   R2   /desvia sempre
R1:     [c]*        /código para os comandos do else
R2:     c           /primeira instrução do comando seguinte ao if
```

Nas figuras os registros de ativação são realçados com traços mais grossos. Na maioria destes registros somente alguns de seus campos são mostrados. Os campos com * contêm valores relevantes e os com X contêm valores irrelevantes.

A notação para seleção dos campos dos registros é tal que:

X(Y)

indica o campo X do registro apontado por Y. Se Y é omitido, então X indica

X(ARATC).

Uma indireção em X é denotada por (X).

As instruções e primitivas são descritas em PASCAL traduzido para o português.

H0,H1... são registradores auxiliares para a execução de instruções e primitivas.

3 - ESTRUTURA GERAL DA MEADA

Nesta seção serão estudadas as características gerais da MEADA, a organização de sua memória e seu funcionamento.

A MEADA é uma máquina à pilha. Esta decisão foi tomada levando em conta a natureza recursiva e a estrutura de bloco da ADA. Como cada tarefa preserva estas características, a cada uma estará associada uma pilha. Pode-se dizer então que a MEADA é mais que uma máquina à pilha, é uma máquina à multipilha.

3.1 - Organização de Memória

A memória da MEADA é composta de quatro regiões principais:

- Registradores Especiais (MRE)
- Memória de Programa (MP)
- Memória de Pilhas (MPIL)
- Memória de Variáveis Dinâmicas (MVD)

A organização de memória da MEADA é ilustrada pela figura 2 .

3.1.1 - Registradores Especiais

CP

Contador de programa .

Conterá o endereço na MP da próxima instrução a ser executada Antes da execução da instrução de endereço i, CP é atualizado para $i+1$, assim, se esta instrução não alterar CP, a próxima instrução a ser executada será a de endereço $i+1$.

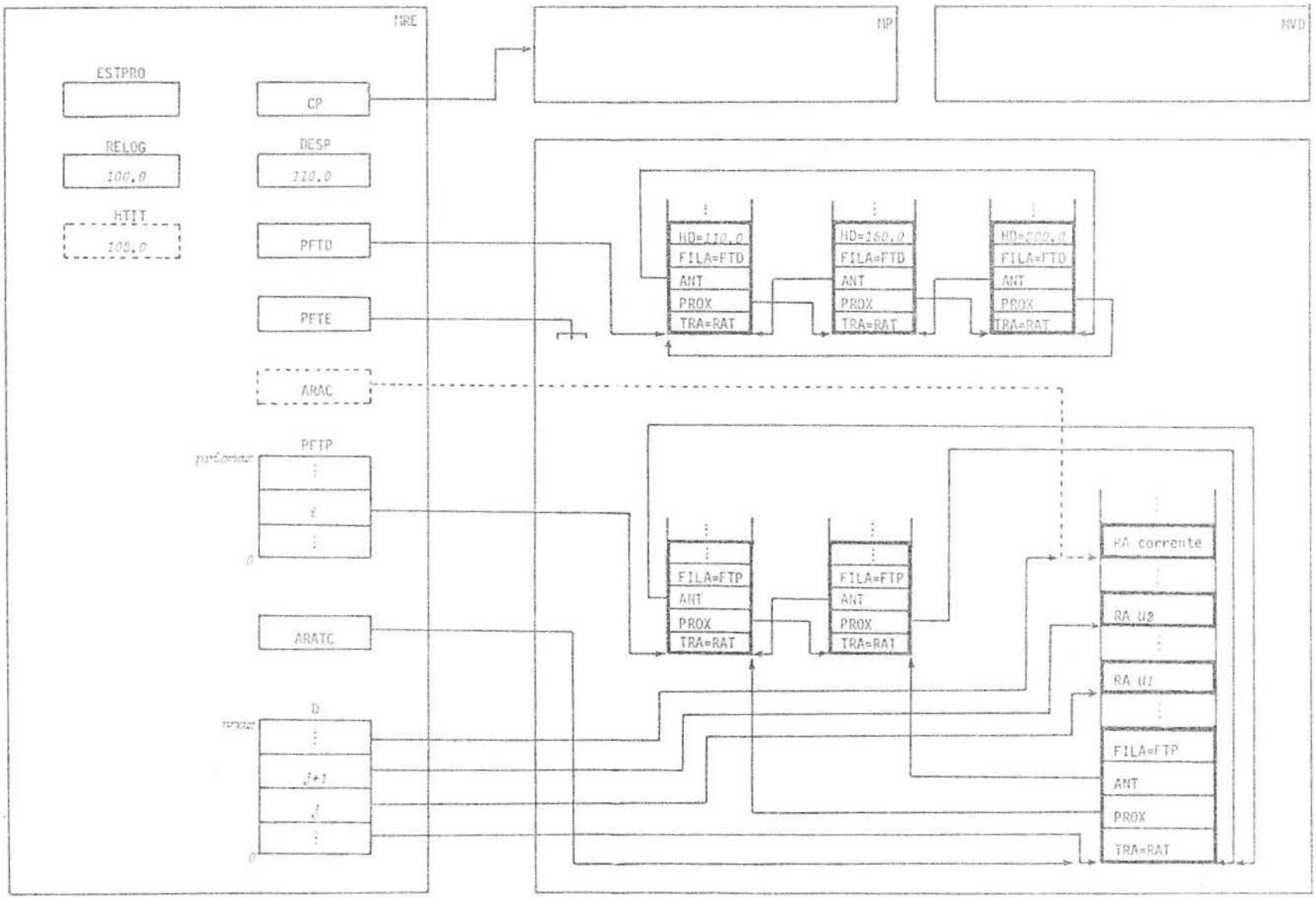


Figura 2 - Organização Geral da MEADA

D

Dnmax
⋮
D0

Vetor de registradores que contêm apontadores para cada nível (de *0* a *nmax*), para as bases dos RAs acessíveis num dado momento ("display").

PFTP

PFTP*0*
PFTP*1*
⋮
PFTP*priomax*

Vetor de apontadores para a primeira tarefa na fila de tarefas prontas para a execução segundo sua prioridade.

PFTP[*i*] aponta para o registro de ativação da primeira tarefa de prioridade *i*, $i \in [0, priomax]$. PFTP[*i*] é igual a NIL se não houver tarefa na fila.

A organização das tarefas numa fila de tarefas prontas segundo a sua prioridade (FTP) é ilustrada pela figura 3 . Os campos PROX e ANT pertencem a registros de ativação de tarefas (RAT).

PFTD

Primeiro da fila das tarefas dormentes.

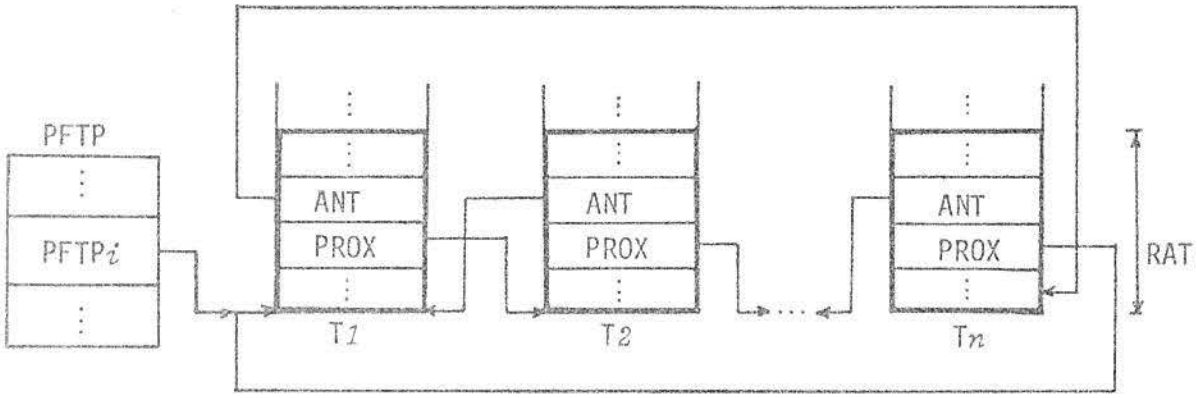
Fila das tarefas que estão "executando" um comando *delay*. A organização das tarefas nesta fila é igual a das FTPs.

A primeira tarefa na fila é a que tem o menor valor de HD(hora de despertar), que é o valor de DESP. As tarefas estão em ordem crescente dos valores de seus HDs (no sentido PROX e até PROX=PFTD obviamente).

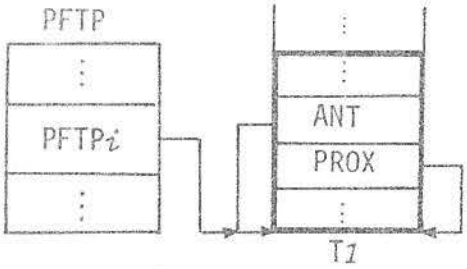
PFTE

Primeiro da fila das tarefas em elaboração.

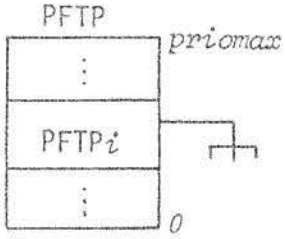
Fila das tarefas para as quais foi executado um comando *initiate* mas que ainda não terminaram a elaboração de suas declarações. A organização das tarefas nesta fila é igual a das FTPs.



a - Fila com n tarefas



b - fila com uma tarefa



c - fila vazia

Figura 3 - Organização das Filas de tarefas

ARATC — Apontador para o registro de ativação da tarefa corrente. Aponta para a base da pilha da tarefa que está sendo executada no momento.

ARAC — Apontador para o registro de ativação corrente (opcional). Com ARAC o acesso às variáveis locais seria mais rápido, pois a indexação em D deixa de ser necessária.

ESTPRO — Estado do processador da MEADA . Indica se o processador pode ou não ser interrompido . Estes dois estados são o mínimo necessário, no entanto numa implementação real é muito conveniente que se tenha vários níveis de interrupção.

RELOG — Registrador de tempo real. Conterá sempre a hora atual.

HTIT — Hora de término de intervalo de tempo.
Hora que o processador deve ser interrompido para que o SCHEDULER atenda a outra tarefa. Este registrador é utilizado nas implementações onde o compartilhamento do processador é feito pela concessão de intervalos de tempo de execução para cada tarefa. A utilização de HTIT é descrita em 3.2.1

DESP — Hora que o processador deve ser interrompido para atender uma tarefa da fila das tarefas dormentes (FTD) que está "acordando" (quando expira o *delay*).

A figura 2 mostra a organização dos registradores especiais e sua relação com as outras regiões de memória.

3.1.2 - Memória de Programa

Esta região de memória conterá instruções da MEADA. Nela estará o código do programa a ser executado.

O tamanho das palavras da MP é tal que qualquer instrução da MEADA ocupa uma palavra. As palavras são numeradas de um em um, a partir de *0*. Supõe-se que o tamanho desta região é sempre o suficiente.

O endereço efetivo da próxima instrução a ser executada é dado por CP.

3.1.3 - Memória de Pilhas

Nesta região serão alocadas as pilhas das tarefas.

As palavras de memória de cada pilha são numeradas de um em um a partir de *0*. A palavra de número *0* corresponde à base da pilha, assim toda referência à palavras (ou posições) da pilha são relativas à sua base. O tamanho de cada palavra da pilha é tal que comporta um endereço (na MP, MVD ou na MPIL)ou qualquer objeto dos tipos predefinidos com exceção do tipo STRING.

A pilha de uma tarefa T tem várias funções, nela são alocadas os RAs e as variáveis locais das unidades ativadas sob o controle de T. Além disso, elas

são utilizadas para o cálculo de expressões e para a passagem de parâmetros nas chamadas de subprogramas.

As primeiras posições de pilha são utilizadas para o RA da própria tarefa. Assim, quando se disser que um apontador aponta para T, ou para o RA de T é o mesmo que dizer que aponta para a base da pilha de T. Similarmente dizer que T foi retirado de uma fila significa que a pilha de T não mais está ligada (através dos campos PROX e ANT) às outras tarefas desta fila.

A alocação e desalocação de memória para estas pilhas são alcançadas através das primitivas ALOCPIL e DESALOCPIL(E). ALOCPIL aloca uma pilha na MPIL e retorna o endereço da primeira posição alocada (endereço da base da pilha). DESALOCPIL(E) libera todas as posições alocadas para a pilha cuja base tem endereço E.

Neste trabalho considerou-se que as pilhas não tem limites de tamanho. Evidentemente numa implementação real isto não é verdade. Estes limites dependem da organização de memória da máquina real e da política de alocação adotada. Se estes limites são ultrapassados, o processador deve levantar a exceção STORAGE-OVERFLOW. Para que isto seja possível a máquina deve contar com um mecanismo de proteção à memória.

3.1.4 - Memória de Variáveis Dinâmicas

Nesta região serão alocadas as variáveis criadas dinamicamente, ou seja, as alocadas por *new*. As palavras da MVD tem o mesmo tamanho que as da MPIL.

A primitiva ALOCMVD(K) aloca K posições na MVD e retorna o endereço da primeira posição alocada. Se não houver espaço disponível, pode ser executado um algoritmo que tenta recuperar o espaço utilizado por variáveis que não são mais acessíveis (processo conhecido como "Garbage Collection"), se mesmo assim não houver espaço, então a exceção STORAGE-OVERFLOW é levantada.

A primitiva DESALOCMVD(E,K) desaloca K posições da MVD a partir do endereço E.

3.2 - Funcionamento

Uma vez inicializados os registradores, filas, etc, ARATC aponta para o RA da tarefa em execução (tarefa corrente) e CP para a próxima instrução a ser executada. CP é automaticamente incrementado de um antes da execução de cada instrução, assim, as instruções são executadas sequencialmente, exceto nas instruções que alteram o valor de CP. Uma tarefa é executada até que ocorra uma interrupção ou que o SCHEDULER (vide 3.2.1) seja invocado por alguma instrução.

As instruções da MEADA podem ou não ser interrompidas; o registrador ESTPRO é automaticamente atualizado antes da execução de cada instrução conforme ela seja interrompível ou não. Quando ocorre uma interrupção e ESTPRO é igual à interrompível ela é atendida; em caso contrário, é salva em uma fila de interrupções para ser atendida posteriormente. Existem dois tipos básicos de interrupção, cujo tratamento é estudado nas seções referidas:

1 - Por tempo: a) Interrupção por término de intervalo de tempo de execução. (vide 3.2.1)

b) Interrupção por término de dormência de tarefa.
(vide seção 8)

2 - Comum a) Interrupção transformada em chamada de entrada.
(vide seção 9.12)

As interrupções que não se encaixam nos tipos acima citados tem seu tratamento determinado pela implementação.

3.2.1 - SCHEDULER

O SCHEDULER é uma primitiva que gerencia o processador. Ele determina qual tarefa o processador deve executar num dado momento e aloca o processador para esta tarefa.

As tarefas que podem ser executadas são aquelas que estão na FTE ou nas FTPs. As tarefas de FTE tem prioridade sobre as outras tarefas. Esta decisão é arbitrária e foi tomada levando-se em conta o processo de semi-atividade por que passam as tarefas até tornarem-se ativas. A FTE poderia não existir e, neste caso, as tarefas seriam colocadas na FTP correspondente à sua prioridade. O processo de ativação de uma tarefa é descrito em detalhes na seção 7.9.

As tarefas nas FTPs estão divididas em grupos conforme suas prioridades. Sempre que não houver tarefa na FTE, o SCHEDULER deve escolher a FTP de prioridade mais alta que não estiver vazia e executar a primeira tarefa desta fila.

O compartilhamento do processador entre as tarefas, pode ser implementado a-

través da determinação de intervalos de tempo de execução para cada prioridade. Assim, uma tarefa seria executada durante um intervalo de tempo ("time--slice") determinado pelo SCHEDULER. Isto se ela não for interrompida ou invocar o SCHEDULER antes do término deste intervalo de tempo.

A política de compartilhamento do processador entre as tarefas é deixada a cargo da implementação, podendo ser através de intervalos de tempo de execução, como no exemplo acima, ou não, como proposto em [IC79b]seção 11.5.1.3 .

Se for por intervalos de tempo, então,antes da execução ser transferida para a tarefa, HIIT deve ser atualizada para RELOG mais o valor do intervalo. Desta forma o processador será interrompido quando RELOG for igual a HTIT e as primitivas abaixo devem ser executadas:

```
SALVACTF;  (* salva contexto da tarefa *)
SCHEDULER; (* chamada ao SCHEDULER *)
```

A primitiva SALVACTF salva os valores de CP, do vetor D e atualiza o tempo acumulado de execução da tarefa,nos campos CPCT, DISP e TAET do registro de ativação da tarefa (RAT),respectivamente. Estas operações devem ser executadas sempre que, por qualquer motivo, a execução de uma tarefa seja interrompida e o SCHEDULER invocado em seguida.

Após escolhida uma fila (FTE ou FTP) o SCHEDULER coloca a primeira tarefa des

ta fila no seu fim (o que é conseguido dando-se uma rotação na fila, que é circular), faz ARATC apontar para o RAT da tarefa, atualiza os registradores D com com os valores de DISP, atualiza o campo TAET (tempo acumulado de execução de tarefa) e transfere a execução para CPCT (que contém o endereço da próxima instrução a ser executada da tarefa).

O tempo acumulado de execução da tarefa é calculado da seguinte maneira: quando uma tarefa é inicializada, a TAET é atribuido o valor zero. Antes de toda e qualquer execução de uma tarefa, o seu campo TAET é atualizado para TAET' (seu valor anterior) menos RELOG' (hora do início desta execução). Quando a execução da tarefa é interrompida TAET passa a valer TAET+RELOG, que resulta em TAET'+RELOG-RELOG'. Assim, quando a tarefa está sendo executada TAET contém um valor negativo e o tempo acumulado é calculado por TAET+RELOG. Quando não está sendo executada TAET contém o valor do tempo acumulado.

4 - DECLARAÇÕES

Nesta seção são estudadas certas declarações que, devido às suas características, justificam algumas das decisões tomadas na implementação proposta neste trabalho.

A ADA permite a definição e declaração de tipos cujas características só são totalmente conhecidas em tempo de execução, ao contrário de linguagens como o PASCAL, onde todas as características de um tipo são determinadas na compilação.

Utiliza-se o termo elaboração para denotar o processo pelo qual uma declaração atinge o seu efeito. Assim, diz-se que uma declaração de um tipo é elaborada dinamicamente se algumas das características deste tipo só são conhecidas em tempo de execução, ou estaticamente, se todas as características são conhecidas em tempo de compilação. A estes tipos chama-se simplesmente de tipos dinâmicos ou estáticos, respectivamente.

A existência destes tipos dinâmicos advém da possibilidade dos limites de intervalos ("ranges") serem especificados por expressões cujo valor só pode ser determinado durante a execução.

Exemplo: V: *array* (1..N) *of* INTEGER;

Se N é uma variável, então o intervalo 1..N é dinâmico, e como conseqüência o tipo de V é dinâmico. A quantidade de memória necessária para representar um objeto (variável ou constante) de um tipo dinâmico só será conhecida durante

a execução.

Conseqüentemente, o espaço para as declarações de um subprograma , módulo ou bloco é subdividido em duas partes. Uma chamada parte estática (PED) e outra chamada parte dinâmica (PDD).

Na PED são alocados os objetos estáticos, os descritores para objetos dinâmicos e os descritores para tipos dinâmicos. Os descritores de tipos dinâmicos são utilizados para guardar as características do tipo, que só são conhecidas durante a execução. Os descritores de objetos dinâmicos conterão, após a elaboração de sua declaração, as características do seu tipo (ou uma referência ao descritor deste tipo, que é dinâmico) e o endereço para a primeira posição alocada para este objeto na PDD.

Na PDD são alocados os objetos dinâmicos e descritores de membros de famílias de tarefas e entradas (vide seção 7.5).

Os parâmetros de subprogramas também podem ser de tipos estáticos ou dinâmicos e são alocados em duas partes,chamadas parte estática dos parâmetros(PEP) e parte dinâmica dos parâmetros(PDP). Este espaço é alocado junto ao espaço para as declarações do subprograma como mostra a figura 4b.

A rigor a PDP só não é vazia se a passagem dos parâmetros for implementada por cópia e quando pelo menos um destes parâmetros for dinâmico.

O compilador associa um endereço textual a cada objeto ou descritor alocado nas partes estáticas.

Os endereços textuais são associados aos descritores de tarefas ou de famílias de tarefas de tal forma que estes descritores ficam agrupados nas primeiras posições das PED de toda unidade que contêm tarefas locais. Desta forma, o início da região onde estes descritores são alocados é sempre conhecido e se a unidade é desativada em conseqüência de um comando *abort* (vide seção 11), as tarefas locais ativas podem ser desativadas.

Para toda declaração dinâmica é gerado código que a elabora. Este código consiste de instruções que montam o descritor para esta declaração e, quando necessário, reservam espaço na PDD. Para toda declaração de objeto inicializado por expressões dinâmicas é gerado código que calcula a expressão e atribui seu valor ao objeto.

O código para as declarações do corpo de uma tarefa ou bloco é dado por:

```
RSRV    |PED|   /reserva |PED| posições na pilha
        [c]*    /código para as declarações dinâmicas e para as que têm
                /inicializações
```

onde |PED| é o tamanho da PED das declarações.

A alocação de espaço para subprogramas difere um pouco do esquema apresentado para tarefas e blocos. Quando um subprograma é chamado, seus parâmetros devem ser elaborados (alocados e avaliados) antes da execução ser transferida para o subprograma. Estes parâmetros devem ser alocados a uma distância fixa relativa ao RA do subprograma. Assim, o espaço utilizado pelo subprograma é alocado na seguinte ordem: espaço para o RA do subprograma, espaço para as partes PED,PEP

PDP e PDD.

Os endereços textuais dos parâmetros são compostos de: nível do subprograma e distância relativa do parâmetro à base do RA do subprograma.

A elaboração dos parâmetros deve ser efetuada antes de se mudar o contexto. Durante a elaboração dos parâmetros, o campo PADCT do registro de ativação de uma tarefa é utilizado para guardar o endereço do RA do subprograma a ser chamado.

A ocorrência de uma exceção durante a elaboração dos parâmetros é tratada pela unidade que está chamando o subprograma, pois o contexto não foi alterado (vide seção 6).

Após a elaboração dos parâmetros, o contexto é atualizado, o RA do subprograma inicializado e a execução desviada para o código do subprograma.

O código gerado para a chamada de um subprograma depende das características deste subprograma. Este código é dado esquematicamente por:

c	K /instrução que marca a pilha para subprograma
$[c]^*$	/código para elaboração dos parâmetros
c	/instrução de desvio para subprograma
$[c]^*$	/ código para as operações de retorno dos parâmetros

K é o tamanho do RA do subprograma mais as partes estáticas PED e PEP.

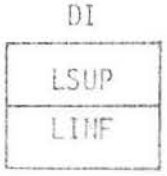
A instrução que marca a pilha, reserva K posições de memória, salva o valor cor

rente do campo PADCT da tarefa onde o subprograma está sendo ativado e atualiza o valor de PADCT, fazendo-o apontar para a primeira posição onde o RA do subprograma foi alocado.

A instrução de desvio salva o endereço de retorno, atualiza o valor de ADCT, fazendo-o apontar para o RA do subprograma, restaura o valor de PADCT e desvia para o código de subprograma. As figuras 4a e 4b ilustram a situação da pilha de uma tarefa T durante a elaboração dos parâmetros de um subprograma e após sua elaboração. Após o retorno, os parâmetros de modos *out* e *in out*, cuja passagem é implementada por cópia, devem ser retornados.

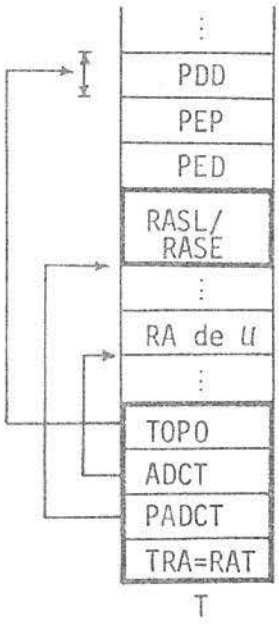
Um exemplo mais simples de tipo dinâmico é o intervalo dinâmico. Todos os outros tipos dinâmicos dependem sempre destes intervalos.

Um descritor de intervalo (DI) consiste de dois campos: o limite inferior e o limite superior do intervalo



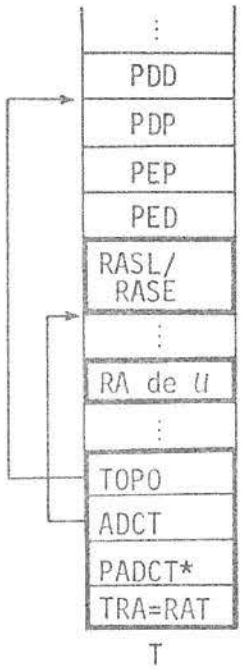
A rigor, numa implementação real, este descritor deve conter informações sobre a representação dos objetos do tipo determinado por este intervalo. Estas informações consistem do tamanho ocupado pelos objetos deste tipo e o alinhamento ("alignment") utilizado.

Este descritor é utilizado para todos os tipos escalares da ADA e particular-



O valor anterior de PADCT é salvo num dos campos do RASL/RASE

Figura 4a. - Situação da pilha de uma tarefa T, durante a elaboração dos parâmetros de um subprograma chamado por uma unidade *U*.



A parte dinâmica das declarações será alocada nesta posição após a elaboração das declarações do subprograma.

O valor anterior de PADCT já foi restaurado.

Figura 4b. - Situação da pilha de T após a elaboração dos parâmetros do subprograma e a chamada ser completada.

mente para o tipo do índice de famílias de tarefas e entradas.

Quando uma família de tarefas ou entradas é declarada, o DI do índice da família deve ser calculado (se contiver expressões dinâmicas)e carregado na pilha.

Código que carrega descritor de intervalos (cdi)

$[c]^+$ /código que calcula o valor do limite inferior do intervalo

$[c]^+$ /código que calcula o valor do limite superior do intervalo

5 - EXPRESSÕES

A implementação das expressões na MEADA segue os padrões usualmente utilizados em máquinas à pilha, ou seja, a execução de uma expressão deixa sempre seu resultado no topo da pilha.

Toda expressão deve pertencer a um determinado tipo, que é determinado pelo contexto em que se encontra a expressão. O compilador deve, portanto, gerar instruções que testam se o resultado da expressão pertence a este tipo. Para as constantes estáticas não é necessário este teste.

As expressões da forma E1 ⊕ E2, onde E1 e E2 são expressões e ⊕ uma operação, são traduzidas por:

$[c]^+$ /código para E1

$[c]^+$ /código para E2

$[c]^+$ /código para operação ⊕

$[c]^*$ /código que testa se o conteudo do topo da pilha pertence /a um determinado tipo

Se E1 e E2 são literais ou constantes estáticas e ⊕ é uma operação pré-definida, então o compilador calcula esta expressão e a considera um literal.

Uma expressão na forma de um literal é simplesmente carregada na pilha por:

```
CRCT    K /carrega a constante de valor K
```

Se o literal é de um tipo T e o contexto exige que ele seja de um subtipo dinâmico de T, então devem ser geradas instruções que testam a pertinência deste

literal ao subtipo.

Um objeto (variável ou constante) deve primeiro ter seu endereço calculado(se for um campo de um registro ou um elemento de uma matriz) e aí seu valor deve ser carregado na pilha. Como para literais, deve-se, em alguns casos, testar a pertinência deste objeto a um tipo.

Em resumo, o cálculo de uma expressão deixa o seu resultado no topo da pilha se ele pertence ao tipo determinado pelo contexto. Em caso contrário, a exceção RANGE-ERROR é levantada.

As exceções ACCESS-ERROR, DISCRIMINANT-ERROR e INDEX-ERROR podem ser levantadas durante o cálculo do endereço do objeto.

As exceções DIVIDE-ERROR, OVERFLOW e UNDERFLOW podem ser levantadas durante a execução de uma operação $\oplus$.

Finalmente, a exceção NO-VALUE-ERROR é levantada quando há acesso a um valor indefinido.

6 - EXCEÇÕES

Nesta seção é estudada a implementação das exceções e o seu tratamento.

A declaração de uma exceção tem como resultado a inclusão de seu nome na tabela de símbolos do compilador. A cada exceção o compilador associa um número inteiro que a identifica. As exceções predefinidas da ADA também tem um número inteiro associado a cada uma delas.

Para que a implementação do tratamento de exceções seja feito de forma controlada e segura, são criados num programa, vários pseudo manipuladores de exceções ("exception handlers"). Estes pseudo manipuladores de exceção não tratam as exceções, mas as propagam para as unidades externas.

A função principal da existência destes pseudo manipuladores é de fazer as operações de retorno ou término das unidades presentemente ativas, desalocando seus RAs de forma ordenada.

Os manipuladores de exceções (ME) estão sempre associados a um RA. Todos os RAs contém o campo EME, endereço do manipulador de exceções. Quando um RA é ativado, EME é inicializado com o endereço do ME desta unidade, quer ele seja um pseudo ME ou não.

Se um bloco, subprograma ou tarefa em ADA não é provido de um ME, então suas instruções de término ou retorno são consideradas pseudo MEs.

As entradas e comandos *select*, que não podem ter ME em ADA, tem suas instru-

ções de término como pseudo MEs.

Como as exceções não podem ser propagadas pelas tarefas, uma exceção é propagada até que ela seja tratada ou que alcance uma tarefa.

Um comando *raise* E executado sob o controle de uma tarefa T é traduzido por :

LVTE N_E /levanta a exceção E de número N_E

A execução desta instrução tem como efeito a atualização dos campos EP (exceção pendente) e IE (identificação da exceção) do RA de T, com os valores TRUE e N_E respectivamente. A execução é então desviada para o ME da ativação corrente, cujo RA é apontado por ADCT. O endereço do código deste ME é portanto dado por EME(ADCT).

Enquanto uma exceção não é tratada EP vale TRUE, passando a FALSE se for tratada .

Um ME real se aparenta com um comando *case*. Sua implementação é feita através de um desvio indireto numa tabela de desvios (do inglês "jump table"). Como normalmente o número de exceções não deve ser muito grande, o tamanho da tabela também não o será.
A seguir é apresentado o código gerado para um ME real.

```
DIME    /desvio indireto em manipulador de exceções
 :      /tabela de desvios gerada pelo compilador
[came]* /código de alternativa selecionável em manipulador de exceção
```

Onde cada $came_i$ é composto pelas instruções:

```
R_i:   TRTE        /trata exceção
       [c]+        /código para os comandos da i-ésima alternativa no ME
       DSVS   R    /desvia para a instrução de retorno ou término
```

O tamanho da tabela de desvios deve ser igual ao número total de exceções, contadas as exceções predefinidas e as declaradas no programa. A numeração das exceções deve ser contigüa e a partir de zero.

Quando um ME em ADA não tiver alternativa *others*, o compilador deve considerar que o ME tem esta alternativa e montar a tabela com o endereço R nas posições referentes à esta alternativa.

A instrução DIME efetua o desvio indireto (através da tabela de desvios) para o código de uma das alternativas do ME, ou para a instrução de término ou retorno.

A instrução TRTE atribui ao campo EP o valor FALSE, indicando que a exceção foi tratada.

Os comandos *raise* sem especificação de exceção (que tem o sentido relevantar uma exceção) são traduzidos por:

```
RLVE /relevanta exceção
```

Sua execução consiste em fazer o campo EP tomar o valor TRUE, pois o campo IE não foi alterado.

Um comando *raise* T'FAILURE é traduzido por:

```
[c]+ /carrega endereço do descritor da tarefa T
LVEF /levanta exceção FAILURE
```

Se a tarefa que executa o comando *raise* T'FAILURE é a própria tarefa T, então a execução de LVEF tem o mesmo efeito que LVTE para a exceção FAILURE.

Em caso contrário, seu efeito depende da situação em que se encontra a tarefa T.

A exceção FAILURE é levantada em T (atualizando seus campos EP e IE). O campo CPCT (do RA de T) é atualizado com o endereço do ME da ativação corrente em T. Desta forma, quando T for escolhido pelo SCHEDULER para execução, este ME será executado.

Se T estiver suspensa (em nenhuma fila)ou numa fila diferente de uma das FTPs ela será colocada na FTP correspondente à sua prioridade para executar os comandos do ME.

7 - TAREFAS

Nesta seção são estudados os códigos gerados para tarefa, família de tarefas e para o comando *initiate*.

A declaração completa de uma tarefa é composta de duas partes, sua especificação (parte visível) e seu corpo. Estas partes não necessariamente vem juntas, podendo inclusive pertencer a unidades de compilação diferentes. Da mesma forma o código gerado é composto de duas partes.

A alocação da declaração de uma tarefa tem como resultado a montagem de um descritor de tarefa e a elaboração das declarações da parte visível. A elaboração das declarações do corpo só é efetuada quando a tarefa é inicializada.

A cada tarefa está associada uma pilha, que pode ser alocada estaticamente , (quando da montagem de seu descritor) ou dinamicamente (quando ela é inicializada).

Quando a tarefa é inicializada,o seu RAT (registro de ativação de tarefas) é alocado nas primeiras posições da pilha e alguns de seus campos são inicializados. O RAT guarda diversas informações de controle da tarefa.

Os estados pelos quais a tarefa pode passar são em número de quatro a nível da MEADA, cada um deles correspondendo a um dos dois estados possíveis a nível da ADA (ativa e inativa).

Além destes estados uma tarefa pode estar em uma das várias filas existentes.

O estado e a fila determinam a situação corrente de uma tarefa.

Uma tarefa adquire o atributo estado quando ela é declarada e o atributo fila quando ela é inicializada.

7.1 - Estados de uma Tarefa

A nível da ADA uma tarefa pode ou não estar ativa. Estes dois estados no entanto não são suficientes para se implementar tarefas de modo seguro . Assim, foram criados quatro estados para tarefas a nível da MEADA, que são ATIVO, SEMI-ATIVO, SEMI-INATIVO e INATIVO.

Uma tarefa inicia sua execução após ser ativada (tornada ativa) pelo comando *initiate*. Assim, a elaboração de suas declarações é feita com a tarefa no estado ativo.

Uma tarefa no estado ativo implica que suas entradas e subprogramas declarados na parte visível podem ser chamados externamente. Estas chamadas podem ser muito perigosas. Por exemplo, um subprograma pode acessar variáveis que ainda não foram elaboradas e portanto a memória por elas utilizada ainda não foi alocada (no caso de variáveis alocadas dinamicamente) ou seu valor inicial ainda não foi atribuído (no caso de variáveis inicializadas na declaração).

O subprograma provocaria ou não uma exceção dependendo do estado da tarefa e não em decorrência da sua própria execução, pois se chamado após a elaboração das declarações da tarefa esta exceção não seria levantada.

À primeira vista uma solução para este problema num sistema de um único processador como se propõe, seria fazer com que a tarefa, ao ser inicializada elaborasse todas as suas declarações sem que pudesse ser interrompida, evitando que a tarefa ficasse à vista das outras tarefas neste "meio estado". No entanto as variáveis (e os limites de intervalo de tipos) podem ser inicializadas (descritos) por expressões dinâmicas quaisquer, que são avaliadas na elaboração. Estas expressões podem conter chamadas de subprogramas (que retornem valor) não locais. Estes subprogramas por sua vez podem chamar entradas ou executar comandos *delay* que causam uma suspensão na execução da tarefa. Ou seja, o fato de subprogramas poderem ser chamados durante a elaboração faz com que a elaboração não possa ser executada de uma única vez, anulando a solução proposta.

Os estados ativo e inativo são insuficientes para descrever os estados reais por que passa uma tarefa, aparecendo então a necessidade de um terceiro estado, chamado elaboração (ou semi-ativo). Desta forma a linguagem se tornaria mais segura , ou melhor, daria chance de se escrever programas mais seguros .

Este estado foi incluido a nível da MEADA sob o nome SEMI-ATIVO. Ele corresponde juntamente com o estado INATIVO ao estado não-ativo em ADA. Sua utilização bem como a solução do problema exposto acima são descritas na seção 7.9.

As tarefas no estado INATIVO não são executadas, ainda não foram inicializadas ou já terminaram sua execução. Suas declarações ainda não foram elaboradas e portanto não são acessíveis. As declarações da parte visível são conhe-

cidas mas os subprogramas e entradas lá declarados também não são acessíveis.

Uma tarefa T só pode terminar (tornar-se INATIVA) quando suas tarefas locais tiverem terminado. É possível que T alcance o seu *end* final sem que suas tarefas locais tenham terminado. Neste caso T deve esperar que todas suas tarefas locais terminem para poder terminar. Durante este período os subprogramas declarados em T (visíveis externamente ou não) podem ser chamados sem problemas, já chamadas a entradas podem causar problemas pois T não mais as aceitará. Se a tarefa S local a T está ativa e chama uma entrada E de T, nem S nem T terminarão sua execução (a menos que outra tarefa interfira). Esta é uma situação conhecida por "Dead-Lock" e que deve ser evitada sempre que possível.

Para solucionar este problema foi criado o estado SEMI-INATIVO a nível da MEADA, que juntamente com o estado ATIVO correspondem ao estado ativo em ADA. Neste estado as chamadas de subprogramas são permitidas mas as chamadas de entradas provocam a exceção TASKING-ERROR no escopo de onde foi efetuada a chamada.

No estado ATIVO as tarefas podem ser executadas, suas declarações já foram elaboradas mas seu término ainda não foi alcançado.

7.2 - Filas de uma Tarefa

Após inicializada, uma tarefa ou está em uma das diversas filas existentes ou está suspensa (em nenhuma fila).

Esta informação é armazenada no RAT no campo denominado FILA. A FILA e o estado de uma tarefa determinam por completo a situação corrente da tarefa.

Uma tarefa é ligada a uma fila através dos campos ANT e PROX do seu RAT. Todas as filas são duplamente ligadas e circulares.

Os possíveis valores de FILA são:

FTE - Fila das tarefas em elaboração.

FTP - Fila das tarefas prontas.

FTD - Fila das tarefas dormentes.

FTER - Fila das tarefas esperando um "rendezvous" com uma dada entrada.

SEDA - Suspensa já elaborada.

SIT - Suspensa inicializando tarefas.

SECE - Suspensa esperando chamada de entrada.

RENDS - Suspensa em "rendezvous".

SAPT - Suspensa com ativação pronta para terminar.

A tabela *1* mostra os possíveis valores de FILA conforme o estado da tarefa.

TABELA *1* - Situações possíveis de uma tarefa

FILA / ESTADO	FTE	FTP	FTD	FTER	SEDA	SIT	SECE	RENDS	SAPT
ATIVA		X	X	X		X	X	X	X
SEMI-ATIVA	X		X	X	X	X		X	X
SEMI-INATIVA		X							X
INATIVA									

7.3 - Descritores de Tarefas

Os descritores de tarefas guardam certas informações sobre a tarefa. Eles são montados quando a especificação da tarefa é elaborada e existem enquanto a unidade *U* onde a tarefa foi declarada estiver ativa.

Existem dois tipos de descritores de tarefas.Um para família de tarefas (DFT) e outro para tarefas simples (DT).

DFT

LSUPF
LINFF
ADTF
TDT=DFT

DT

ET
ARAT
ATFM
TA=EST/DIN
IT ou TDT=DT

TDT - Tipo do descritor de tarefa (DT/DFT).

ADTF - Apontador para a região alocada para os descritores de tarefas da família.

LINFF - Limite inferior da família de tarefas.

LSUPF - Limite superior da família de tarefas.

TA - Tipo de alocação (EST/DIN).

ATFM - Apontador para a tarefa mãe.

ARAT - Apontador para o registro de ativação (para a base da pilha).

ET - Estado da tarefa (ATIVA, SEMI-ATIVA, SEMI-INATIVA e INATIVA).

IT - Índice da tarefa na família. Nos descritores dos membros da família não é necessário o campo TDT. Assim, utiliza-se o campo IT.

O endereço textual de uma tarefa é o endereço textual de seu descritor: nível de u, deslocamento do descritor em relação à base do RA de u (N_u,d_{DT}). O seu endereço efetivo é carregado na pilha por:

CREN N_u,d_{DT} /carrega endereço.

O endereço efetivo da i-ésima tarefa de uma família é dado por:

$$ADTL(End_{DFT})+(i-LINFF(End_{DFT}))*|DT|$$

onde End_{DFT} é o endereço efetivo do descritor da família e $|DT|$ é o tamanho de um DT.

Este endereço efetivo é carregado na pilha a partir do endereço textual (N_u, d_{DFT}) do descritor da família e do valor do índice da tarefa. Assim, o endereço efetivo de T(e) é carregado pelas instruções:

$[c]^+$ /código para a expressão e.

CMFT N_u,d_{DFT} /carrega endereço de descritor de membro de família /de tarefas.

7.4 - Registro de Ativação de Tarefas (RAT)

O registro de ativação de uma tarefa é ilustrado na figura 1 juntamente com os outros registros de ativação existentes.

Descrição dos campos:

TRA - Tipo de registro de ativação. Identifica o registro de ativação, portanto é igual a RAT. Este campo é utilizado pela primitiva DESATIVA que termina incondicionalmente a execução de uma tarefa(v.seção 11). mando *abort* foi executado.

EME - Endereço do código do manipulador de exceções da tarefa. Quando uma exceção é levantada na tarefa, a execução deve ser desviada para este endereço.

PADCT-Próximo valor a ser atribuido ao apontador dinâmico do contexto da tarefa. Este campo é utilizado nas chamadas de sub-programas durante a elaboração dos parâmetros efetivos. Neste intervalo o RA do subprograma já foi alocado mas o contexto ainda não foi mudado. Este campo só é utilizado neste intervalo de tempo. (vide seção 4)

NDTL- Número de descritores de tarefas locais. Os descritores de membros de família não são contados. Este campo é utilizado pela primitiva DESATIVA quando a tarefa termina incondicionalmente (v.seção 11).

NTLA- Número de tarefas locais ativas. Em situações normais (diferente de aborto), uma tarefa (ou qualquer unidade ativa) não pode terminar sem que suas tarefas locais tenham terminado. Assim, o campo NTLA é utilizado para saber se a tarefa pode terminar ou deve ficar suspensa até que NTLA seja igual a zero. (vide 7.10)

IND - Índice da tarefa dentre as que foram inicializadas pelo mesmo comando *initiate*. Este campo é utilizado quando uma tarefa está SEMI-ATIVA. Neste estado, a tarefa não pode ter tarefas locais ativas, pois ela própria não está ativa. Assim, pode-se usar a posição do campo NTLA para este propósito. (vide 7.9)

NC - Nível léxico da ativação corrente. É utilizado para as atualizações no contexto da tarefa.

CPCT- Contador de programa do contexto da tarefa. Quando a tarefa não está sendo executada, este campo contém o endereço na MP da próxima instrução a ser executada.

PROX,ANT -Interligam a pilha de uma tarefa à outras tarefas numa mesma fila. Os quatro campos seguintes podem ocupar estas posições, pois só são utilizados quando a tarefa está suspensa.

NTI,NTE -Número de tarefas inicializadas num mesmo *initiate* e número das que ainda estão em elaboração. Estes campos são utilizados quando a tarefa está suspensa inicializando tarefas (vide seção 7.9.).

ATDE,ARAE -Apontador para o RA da entrada que foi chamada por esta tarefa e apontador para a tarefa de declaração da entrada. Estes campos são utilizados quando a tarefa está suspensa esperando "rendezvous" (vide seção 9.9)

ADCT- Apontador dinâmico do contexto da tarefa. Contém o endereço efetivo da base do último RA ativado na pilha da tarefa. É igual a D[NC] quando a tarefa está sendo executada.

TOPO- Apontador para a última posição utilizada da pilha.

EP - Exceção pendente. Vale TRUE se existe exceção a ser tratada e FALSE em caso contrário.

IE - Identificação da exceção. Identifica a exceção pendente.

PRIO- Prioridade de execução da tarefa. Seu valor pode variar de zero a *priomax*. O valor de *priomax* é determinado em cada implementação.

FILA- Identifica a fila em que está a tarefa ou, se não estiver em nenhuma fila, as condições por que está suspensa. (vide 7.2)

HD - Hora de despertar. Este campo é utilizado quando a tarefa executa o comando *delay*.

TAET- Tempo acumulado de execução da tarefa.(vide 3.2.1).

PLSE- Primeiro registro de ativação na lista das ativações externas de subprogramas externos da tarefa. Por procedimentos externos (SE) entende-se os subprogramas declarados na parte visível de uma tarefa. Esta lista é necessária pois a tarefa, se terminar, deve propagar a exceção TASKING-ERROR para as unidades que chamaram e estão executando seus subprogramas externos. (vide seção 7.10)

TQEI- Apontador para a pilha da tarefa que executou o comando *initiate* para esta tarefa. Este campo é utilizado quando a tarefa está SEMI-ATIVA e portanto seus subprogramas externos não podem ser chamados. Assim, pode-se usar a posição do campo PLSE para este propósito. (vide 7.9)

PFDEA-Primeiro na fila dos descritores de entradas abertas. (vide 9)

ADET- Apontador para a região dos descritores de entradas da tarefa. Esta informação é utilizada no término da tarefa ou quando ela é abortada.

NDET- Número de descritores de entrada. Os descritores de membros de famílias de entradas não são contados. Esta informação é utilizada no término de uma tarefa ou quando ela é abortada.

AUDT- Apontador para a unidade de declaração da tarefa. Esta informação é usada para atualizar o campo NTLA da unidade onde esta tarefa foi declarada quando ela terminar ou for abortada.

APDT- Apontador para o seu descritor de tarefas. Esta informação é utilizada para que a tarefa possa acessar as informações contidas no seu descritor. São várias as situações em que APDT é utilizado.

DISP- Vetor de *nmax+1* posições que guarda o contexto da tarefa quando ela não está sendo executada.

7.5 - Esquema de Alocação de Espaço para as Declarações de uma Tarefa.

A alocação de espaço para as declarações do corpo de uma tarefa segue o esquema geral proposto na seção 4. Este espaço também é composto de duas partes principais PED e PDD. O espaço para PED é reservado quando a tarefa inicia a elaboração de suas declarações e é organizado como mostra a figura 5.

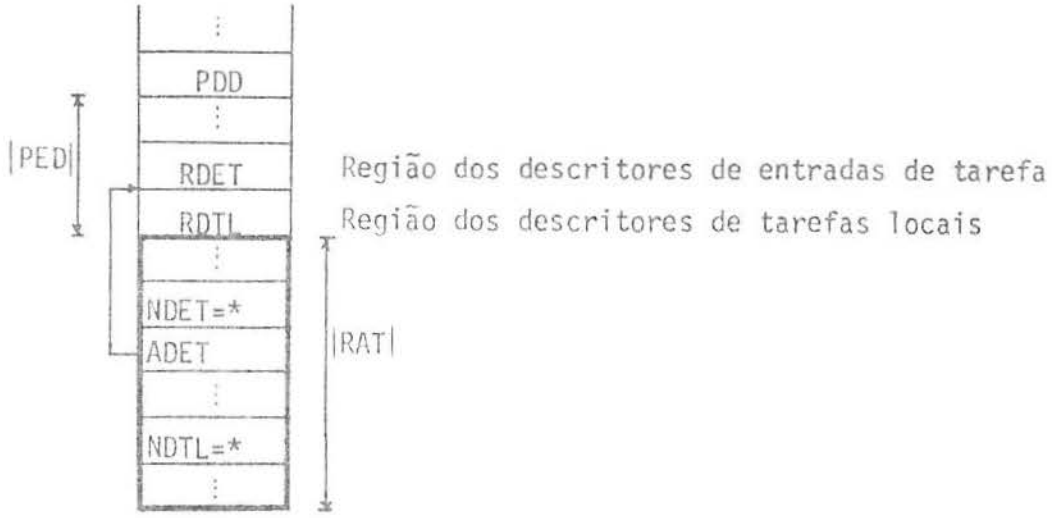


Figura 5 - Esquema de Alocação de Espaço para Tarefas

O compilador atribui endereços textuais aos descritores de tarefas e entradas de tal forma que eles ficam agrupados nas posições seguintes ao RAT e nesta ordem. Ao término da elaboração das declarações da tarefa os campos NDTL,ADET e NDET são inicializados.

Os descritores de membros de famílias de tarefas ou entradas são montados (durante a elaboração) na PDD, mesmo que os limites de intervalos da família sejam estáticos (conhecidos do compilador). Os valores atribuídos a NDET e NDTL não incluem os descritores dos membros das famílias e portanto contêm o número de descritores em RDET e RDTL respectivamente.

A alocação dos descritores é feita desta forma para facilitar a propagação da exceção TASKING-ERROR às tarefas que estão esperando "rendezvous" com entradas de uma tarefa que é abortada ou termina. Ou ainda, para abortar as tarefas locais ativas.

O restante da PED é ocupado pelas outras declarações da tarefa.

7.6 - Código para a Parte de Especificação de Tarefa.

O código gerado para a especificação de uma tarefa (ou família de tarefas) consiste do código que monta um DT (ou DFT) e do código que elabora suas declarações.

O quadro *1* mostra o código que deve ser gerado para a montagem de um DT (ou DFT) para tarefas ou família de tarefas com alocação estática ou dinâmica.

QUADRO *1* - Código para montagem de um DT ou DFT

TIPO DO DESCRITOR	ALOCAÇÃO ESTÁTICA	ALOCAÇÃO DINÂMICA
DFT	cdi MFTE d_{DFT}	cdi MFTD d_{DET}
DT	MDTE d_{DT}	MDTD d_{DT}

d_{DT} - Deslocamento do DT relativo à base do RA da unidade de declaração da tarefa.

d_{DFT} - Deslocamento do DFT relativo à base do RA da unidade de declaração da tarefa.

É necessário carregar o DI da família antes da execução da instrução que monta o seu descritor, o que é feito pelo código cdi descrito na seção 4.

Apesar das declarações da parte de especificação pertencerem à tarefa, sua elaboração é efetuada no escopo da sua unidade *U* de declaração. Assim, se for necessário alocar espaço para estas declarações, este espaço será alocado nas regiões PED e PDD de *U*, salvo nos casos mencionados abaixo.

Nas declarações de subprogramas nenhum espaço precisa ser alocado, pois os subprogramas são redeclarados no corpo da tarefa.

Nas declarações de entradas, o espaço a ser utilizado para o seu descritor será alocado na RDET da tarefa e o compilador deve, portanto, guardar esta informação para quando o corpo da tarefa for traduzido.

As declarações que podem necessitar da alocação de espaço são aquelas que envolvem intervalos dinâmicos ou inicializações com expressões dinâmicas.

Como a declaração de variáveis não é permitida, a alocação de espaço em *U* fica restrita aos descritores de tipos (que contenham intervalos dinâmicos e/ou inicializações dinâmicas) e as possíveis inicializações em parâmetros de entradas ("default parameter").

Estes tipos são tratados pelo compilador como sendo de *U*. As inicializações são tratadas como declarações de constantes em *U*. Recebem um endereço textual e é gerado código que calcula a expressão dinâmica e a atribui a esta constante.

O código gerado para a especificação de uma tarefa com alocação dinâmica é , por exemplo:

```
        MDTD    d_DT
        [c]*    /código que elabora as declarações da especificação.
```

7.7 - Código para o corpo de tarefas

O corpo de uma tarefa só é executado quando ela é inicializada. Assim, a primeira instrução gerada é um desvio para a instrução seguinte ao código do corpo.

```
        DSVS    RA
RET:    [c]*                        /Código da parte declarativa da tarefa.
        TNEL    d_DET,N_DET,N_DTL   /Término de elaboração de tarefa.
        [c]*                        /Código da parte executável da tarefa.
R:      TNT1                        /Término de tarefa.
        TNT2                        /Término de tarefa.
RM:     [c]+                        /Código do ME da tarefa
RA:     c                           /Instrução seguinte.
        :
```

O ME de tarefas segue o padrão apresentado na seção 6. Quando a tarefa não tiver ME o código gerado é modificado a partir da instrução TNT1 para:

```
RM:     TNT1
        TNT2
RA:     c
        :
```

7.8 - Comando *initiate*

O comando *initiate* pode inicializar uma ou mais tarefas. As informações necessárias para a inicialização de cada tarefa são carregadas na pilha da tarefa onde está sendo executada a unidade que contém o comando *initiate*.

initiate T1,...Tn(e);

é traduzido por:

```
                        /Informações de T1.
   CRCT    RM1
   CRCT    N_U1
   CREN    N_U1,d_DT1
   CRCT    RET1
    ⋮
                        / Informações de Tn(e).
   CRCT    RMn
   CRCT    N_Un
   [c]+                 /Código para a expressão e.
   CMFT    N_Un,d_DFTn  /Carrega endereço do descritor de membro de família
                        /de tarefas.
   CRCT    RETn
   INIT    n            /Inicializa tarefas.
```

RMi - endereço de entrada no manipulador de exceções de Ti.

N_{Ui},d_{DTi} - endereço textual do descritor da tarefa Ti.

N_{Ui},d_{DFTi} - endereço textual do descritor de família de tarefas da tarefa Ti.

RETi - endereço de entrada na tarefa Ti.

7.9 - Processo de Ativação de Tarefas

Nesta seção é descrito o processo de ativação de uma tarefa e como é contornado o problema apresentado na seção 7.1 referente ao estado de uma tarefa durante a elaboração de suas declarações.

São analisadas a seguir certas situações que servirão de subsídio para que se mostre as vantagens do processo proposto e como ele essencialmente não altera a semântica da ADA.

Supondo que: as tarefas T*0* e S estão ativas; as tarefas T*1*,..T*i*,..T*n* estão ativas; o corpo de T*0* contem o comando

initiate T*1*,..T*i*,..T*n*;

que T*i* tem um subprograma P visível externamente e que o corpo de S contém um dos dois comandos abaixo

T*i*.P; -- *

if T*i*'ACTIVE *then* T*i*.P *end if*; -- **

São analisadas as situações que podem ocorrer conforme o estado mais ou menos avançado da execução de cada tarefa.

1 - A tarefa T*i* não está ativa e S executa o comando marcado por *.
RESULTADO: A tarefa S recebe a exceção TASKING-ERROR.

2 - A tarefa T*i* não está ativa e S executa o comando marcado por **.
RESULTADO: A chamada a T*i*.P não é executada e S segue sem exceção.

3 - A tarefa T*i* está ativa e S executa o comando marcado por *.
RESULTADO: a-Se T*i* já estiver elaborada, a chamada a T*i*.P é executada e S segue sem exceção

b-Se Ti ainda não estiver elaborada, a chamada a Ti.P é executada podendo ou não retornar com exceção, ou então produzir resultados sem sentido.

4 - A tarefa Ti está ativa e S executa o comando marcado por **.

RESULTADO: a-O mesmo que 3a.

b-O mesmo que 3b.

As situações 1 e 2 são bem determinadas, a confiabilidade do programa depende unicamente de sua programação.

As situações 3 e 4 não são determinísticas, pois não se pode testar se a tarefa está em elaboração ou já elaborada. Se a elaboração pudesse ser um processo instantâneo, o não determinismo nas situações 3 e 4 seria evitado. Em 1 uma exceção seria levantada de qualquer forma, mostrando que esta programação é por si perigosa.

Estando Ti ativa, se por sorte sempre que Ti.P fosse chamado Ti já tivesse terminado sua elaboração, então o não determinismo em 3 e 4 desapareceria,restando somente o não determinismo entre 1 e 3, o que outra vez mostra a insegurança que é introduzida num programa que tenha um comando como *. Este não determinismo no entanto é inerente à linguagem e pode ser controlado como por exemplo no comando **.

A semântica de um comando *initiate* T1,..Tn assegura que ao término de sua execução as tarefas T1,..Tn estão ativas (sendo que algumas delas já podem ter terminado quando o comando seguinte ao *initiate* é executado). Se o comando * ou ** seguir o comando *initiate* em T0, as quatro situações anteriormente descritas podem ocorrer. Da mesma forma, o não determinismo de 3 e 4 desaparece

se, por sorte, *Ti* já tiver terminado sua elaboração quando o comando seguinte ao *initiate* é executado.

Caso seja feito da sorte uma regra, não se altera essencialmente a semântica da linguagem, que se torna mais confiável.

As tarefas inicializadas por um mesmo comando *initiate* recebem a exceção TASKING-ERROR se chamarem durante a elaboração de suas declarações subprogramas visíveis externamente de uma das outras. Assim, a semântica de um comando *initiate* para mais de uma tarefa passa a ser: as tarefas inicializadas por um mesmo comando *initiate* tornam-se ativas simultaneamente após a elaboração de suas declarações, sendo possível que alguma delas (ou todas) terminem durante a elaboração de suas declarações.

A implementação proposta neste trabalho considera uma tarefa ativa somente quando ela está com todas as suas declarações elaboradas. A tarefa que executa o *initiate* fica suspensa até que as tarefas por ela inicializadas estejam elaboradas ou terminem, caso uma exceção seja levantada durante a elaboração.

Para que a elaboração das declarações de uma tarefa se aproxime (a vista das outras tarefas) de um processo instantâneo, as tarefas em elaboração tem prioridade absoluta sobre todas as outras tarefas (vide seção 3.2.1).

A seguir, são ilustradas as situações por que passam as tarefas *T0*,*T1*....*Tn* quando o comando

initiate *T1*,...*Tn*;

é executado pela tarefa *T0*. Este comando é traduzido como descrito na seção

7.8. As informações das tarefas *T1*,...*Tn* são carregadas na pilha de *T0* e a instrução INIT *n* é executada.

A tarefa *T0* pode estar ATIVA numa FTP ou SEMI-ATIVA na FTE. Se estiver na FTE é porque durante sua elaboração ela chamou um subprogramas externo e este executou o comando *initiate*.

A figura 6 ilustra a situação da pilha de *T0* imediatamente antes da execução da instrução INIT *n*. O descritor da tarefa *T1* também é ilustrado juntamente com o RA da sua unidade *U1* de declaração.

Se uma ou mais tarefas dentre *T1*,...*Tn* não estiver INATIVA, então a exceção INITIATE-ERROR é levantada em *T0* e a inicialização não tem efeito.

Caso contrário, *T0* fica suspensa inicializando tarefas (SIT); os campos NTE e NTI passam a conter o valor *n*; são alocadas as pilhas para as tarefas com alocação dinâmica; para cada uma das *n* tarefas o endereço de sua pilha é salvo na pilha de *T0*; alguns campos do RAT destas tarefas são inicializados, dentre eles TQEI com o endereço da pilha de *T0*, IND que é utilizado para que se possa acessar o apontador para sua pilha salvo na pilha de *T0*, o campo TAET com zero, os apontadores APDT e AUDT para seu DT e para sua unidade de declaração respectivamente.

A figura 7 ilustra a situação logo após a execução da instrução INIT *n*. É possível que haja outras tarefas na FTE se *T0* estiver na FTE. A pilha de *T1* é mostrada em detalhes.

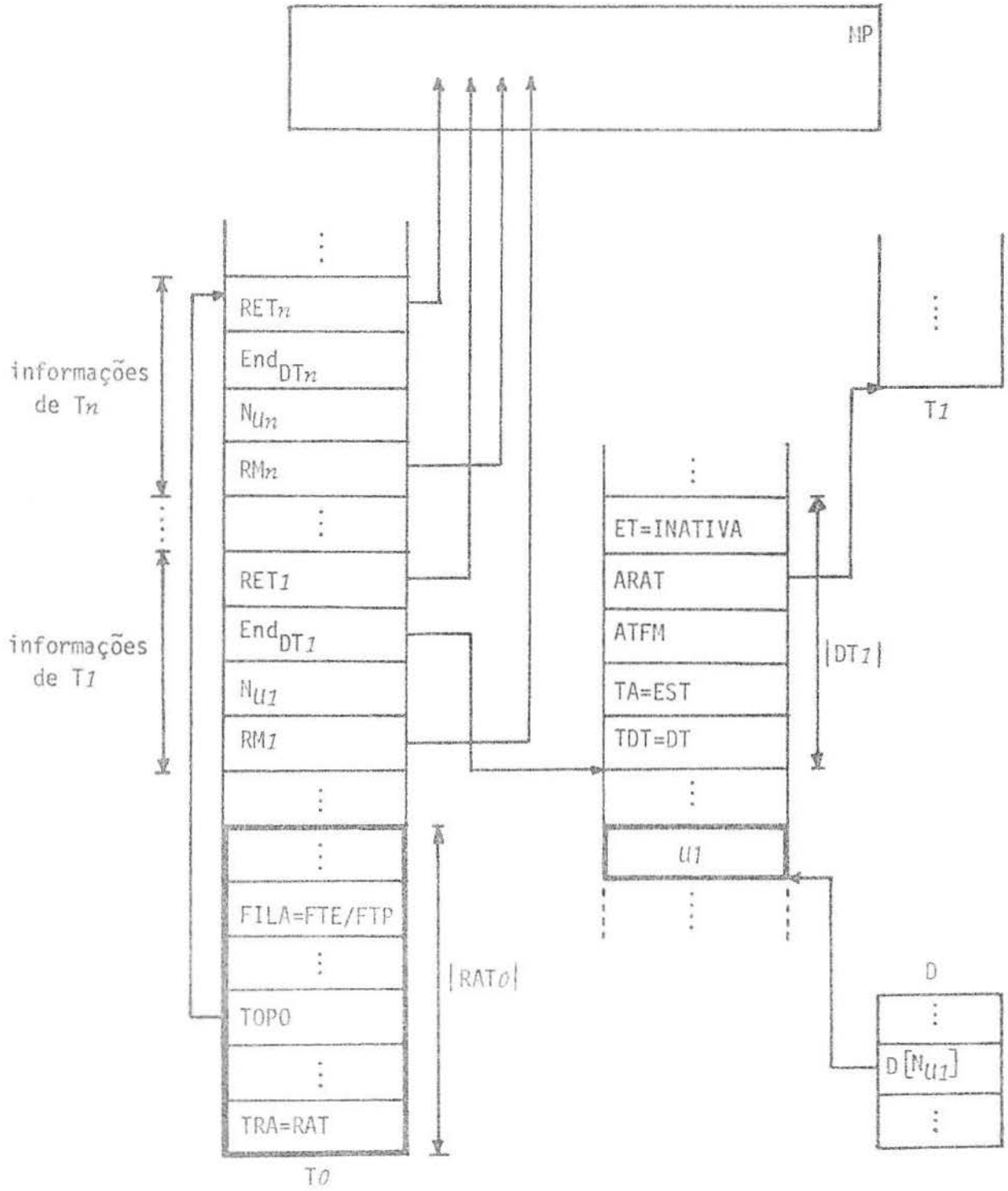


Figura 6 - Situação da tarefa T0 imediatamente antes da execução da instrução INIT n.

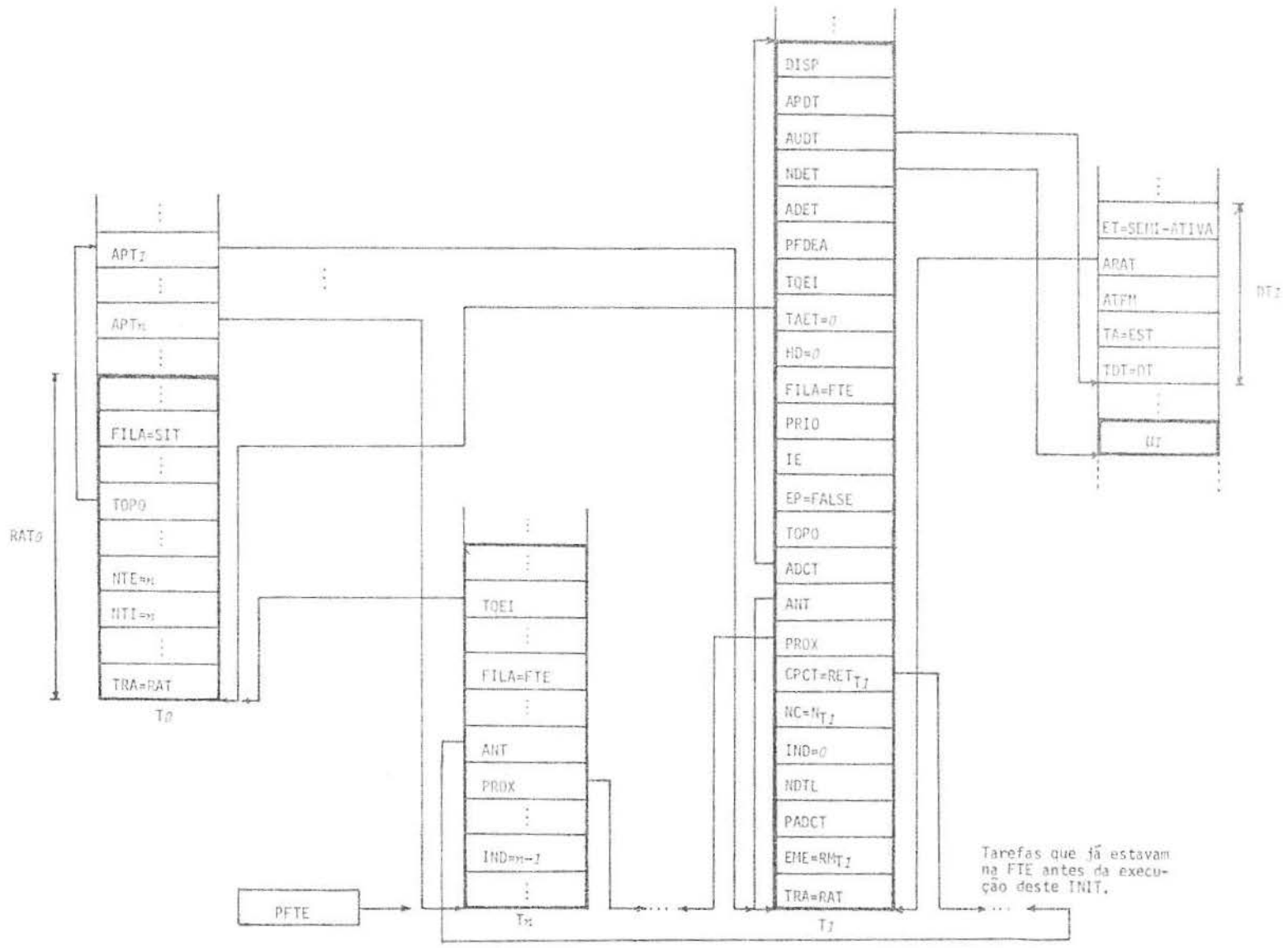


Figura 7 - Situação das tarefas T_0,..T_N logo após a execução da instrução INIT N.

Durante a elaboração das declarações de uma tarefa T*i* várias situações podem ocorrer. Uma exceção pode ser levantada em decorrência da execução de alguma instrução e a tarefa pode terminar antes de terminar a elaboração de suas declarações. Se isto ocorrer a instrução TNT*1* (término de tarefa) subtrai um do campo NTE da tarefa T*0* e faz o apontador APT*i* salvo na pilha de T*0* valer NIL. Assim, quando todas as tarefas terminarem a elaboração,T*i* não será ativada.

Como T*i* pode chamar subprogramas externos durante a elaboração, é possível que por intermédio destes subprogramas, T*i* chame uma entrada ou execute um comando *delay* ou inicialize outras tarefas. T*i* é retirada da FTE e inserida na FTER da entrada ou na FTD, ou fica suspensa inicializando tarefas, respectivamente.

A figura 8 mostra uma possível situação durante a elaboração das tarefas T*1*,...T*n*.A tarefa T*1* está na FTE e está sendo executada no momento; T*2* está na FTD e despertará quando RELOG for igual a 100.0;T*n* está em uma FTER de alguma entrada de uma tarefa ATIVA;T*3* já está elaborada e portanto está suspensa com o campo FILA igual a SEDA; a tarefa T*4* está INATIVA em decorrência de alguma exceção levantada durante sua elaboração; as demais tarefas estão já elaboradas como T*3*.

Os campos do RAT de T*3* marcados com X contêm valores irrelevantes. Os marcados com * contêm valores relevantes,como por exemplo TAET que contêm o tempo que T*3* demora para se elaborar. Os campos NDET,ADET e NDTL foram inicializados pela instrução TNEL (término de elaboração) e determinam as regiões RDTL e RDET. O campo IND não é mais necessário e passa a ser utilizado como HD que é inicializado como zero. O campo PFDEA é inicializado com NIL pois

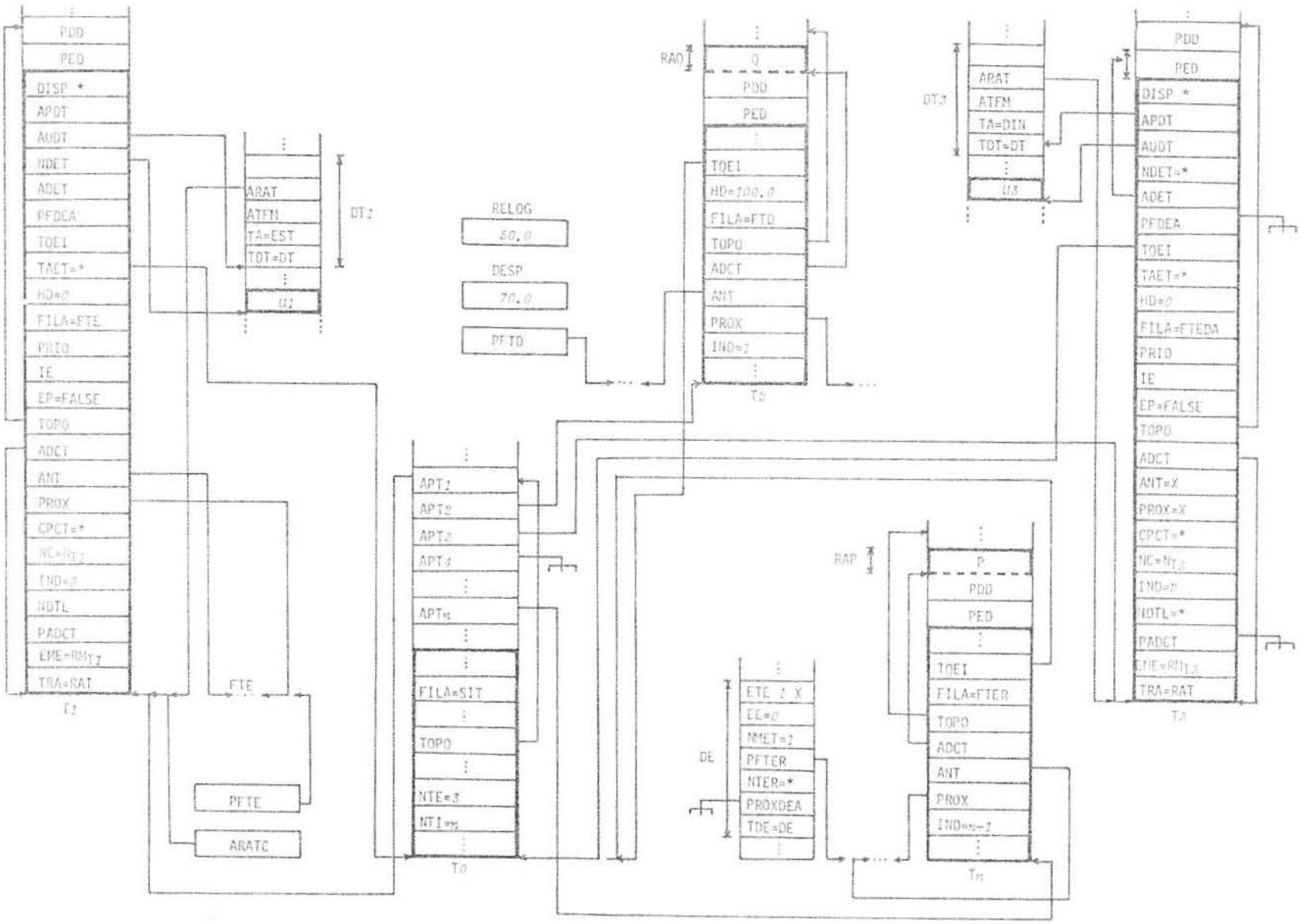


Figura 8 - Uma possível situação das tarefas T0..Tn durante a elaboração das tarefas T1..Tn.

T3 não tem entradas abertas. Finalmente, PED e PDD já estão alocadas.

O valor de NTE de T0 é três, indicando que ainda existem três tarefas em elaboração (T1, T2 e Tn). Quando a última tarefa terminar sua elaboração quer tornando-se INATIVA, quer já elaborada, o valor de NTE passa a ser zero e a instrução INIT1 ou TNEL ativa todas as tarefas já elaboradas colocando-as,juntamente com T0, na FTP correspondente à prioridade de T0.

O campo PRIO das tarefas já elaboradas é então inicializado com esta prioridade. O campo NTLA do RA da unidade de declaração de cada uma das tarefas é acrescido de um, indicando que existe mais uma tarefa local ativa. O SCHEDULER é então chamado para escolher uma tarefa para execução.

7.10 - Término de Tarefas.

Para implementar o término de uma tarefa são necessárias duas instruções:TNT1 e TNT2.

Toda tarefa termina com ou sem exceção por estas instruções. A existência ou não de exceção pendente quando a tarefa termina é irrelevante, pois a exceção não é propagada.

Uma tarefa Ti pode estar ATIVA ou SEMI- ATIVA quando executa TNT1. Se está SEMI-ATIVA ela está na FTE, de onde é retirada. O seu estado passa a INATIVA. O apontador APTi salvo na pilha da tarefa T0 que a inicializou passa a valer NIL. O campo NTE de T0 é subtraido de um e se com isto passar a valer zero então todas as tarefas já elaboradas e inicializadas por T0 são ativadas. Se o

tipo de alocação de Ti for dinâmico, sua pilha é desalocada. O SCHEDULER é então chamado para escolher outra tarefa para execução.

Estando ATIVA, as seguintes operações são executadas por TNT1: a tarefa Ti é suspensa (retirada da FTP onde se encontra); utilizando os campos ADET e NDET a exceção TASKING-ERROR é propagada para as tarefas que estão esperando rendezvous com entradas de Ti; se Ti não tiver tarefas locais ativas (NTLA≠0), então TNT2 é executada em seguida. Em caso contrário, seu estado passa a ser SEMI-INATIVA e Ti fica suspensa com ativação pronta para terminar (FILA=SAPT) e só será reativada para executar TNT2 quando NTLA passar a valer zero.

A instrução TNT2 executa as seguintes operações: utilizando-se do campo PLSE é propagada a exceção TASKING-ERROR para todas as tarefas que estão executando subprogramas visíveis externamente de Ti (vide seção 11); o campo NTLA da unidade U_i de declaração de Ti é subtraído de um.

Se com isto NTLA passar a valer zero, e se a tarefa S onde U_i está ativada estiver suspensa com ativação pronta para terminar (FILA=SAPT), então S é colocada na FTP correspondente à sua prioridade para que U_i seja terminada. O estado de Ti passa a ser INATIVA. Se o tipo de alocação de Ti for dinâmico, sua pilha é desalocada. O SCHEDULER é então chamado para escolher outra tarefa para execução.

Note que U_i pode eventualmente ser uma tarefa (a própria tarefa S), e neste caso, se sua FILA for igual a SAPT e NTLA for zero,então S é colocada na FTP correspondente à sua prioridade para que a sua instrução TNT2 seja executada.

8 - COMANDO *delay*.

Um comando

delay e;

onde *e* é uma expressão do tipo predefinido TIME, é traduzido por:

$[c]^+$	/código para a expressão *e* do *delay*
ECDL	/"executa" comando *delay*

A tarefa é inserida na FTD, ficando dormente durante um período de tempo *e*. Ela será "acordada" antes deste período se receber a exceção FAILURE ou se for abortada.

Se o valor de *e* for igual a zero, a tarefa não é inserida na FTD e segue sua execução normalmente. A rigor, numa implementação real a tarefa somente deveria ser inserida na FTD se *e* fosse maior que o tempo de execução das operações que a tornaram dormente.

9 - ENTRADAS

Nesta seção são estudadas as entradas, o comando *accept* e o comando *select*.

A declaração de uma entrada tem como resultado a inclusão do nome da entrada, do nome de seus parâmetros e dos tipos destes na tabela de símbolos do compilador e da geração de código que monta um descritor para esta entrada.

O compilador associa a cada entrada E um endereço que corresponde à posição de seu descritor na pilha da tarefa T onde E é declarada. A entrada E é dita local à tarefa T.

A entrada E só pode ser chamada quando T está ATIVA. Em caso contrário,a exceção TASKING-ERROR é levantada na unidade *U* que chamou a entrada, no ponto da chamada.

Uma entrada pode estar aberta ou fechada. Inicialmente as entradas estão fechadas. Uma entrada E torna-se aberta quando um comando *accept* é executado para ela, permanecendo neste estado até que ela seja chamada.

Enquanto E está aberta, T fica suspensa esperando que E seja chamada por uma tarefa Q. Quando se diz que Q chama E subentende-se que a unidade *U* (bloco , subprograma ou a própria tarefa Q) ativada em Q chama a entrada E. Quando E é chamada por Q e T está ATIVA, E pode estar:
ABERTA - A chamada é dita aceita. A tarefa Q é suspensa e T é reativada (colocada na sua FTP) e a execução dos comandos da entrada tem inicio. Durante a execução da entrada diz-se que T e Q estão em "rendezvous". Terminado o

"rendezvous" Q é reativada e pode continuar a sua execução a partir da chamada da entrada e T a partir do comando seguinte ao último comando da entrada. FECHADA - A tarefa Q é suspensa e inserida na fila de tarefas esperando "rendezvous" (FTER) da entrada E. A tarefa Q é colocada no fim da FTER , ficando nesta fila até que (a)-ela seja a primeira da fila e que (b)-a entrada E se torne aberta, quando então Q é retirada da FTER e a chamada é aceita.

Como podem existir vários comandos *accept* para uma mesma entrada, é necessário guardar para qual deles a entrada está aberta. Esta informação , chamada de endereço de transferência da entrada (ETE), é armazenada no descritor da entrada e consiste do endereço do código gerado para o *accept* que a abriu.

Como num comando *select* podem existir vários *accept* para uma mesma entrada,esta entrada pode ficar aberta para mais de um *accept*. Neste caso é necessário que se guarde vários ETEs ao invés de um único. O compilador no seu primeiro passo conta, para cada entrada, o número máximo de comandos *accept* para esta entrada em cada comando *select* da tarefa. O maior destes máximos (um para cada *select*) é escolhido como número máximo de endereços de transferência(NMET) da entrada. Assim, o descritor de cada entrada conterá um vetor ETE de NMET posições.

9.1 - Estados de uma Entrada.

Uma entrada pode estar fechada ou aberta. Quando uma tarefa T é ativada, todas suas entradas estão fechadas. Uma entrada E torna-se aberta quando um comando

accept E;

é executado e E ainda não foi chamada. Esta entrada permanece aberta até que ela seja chamada e o processo de aceitação da chamada tenha início.

Como um comando *select* pode conter mais de um *accept* para uma mesma entrada , esta entrada pode ficar aberta para mais de um *accept*. Quando isto ocorrer uma entrada fica n-aberta, sendo n o número de comandos *accept* executados para esta entrada num mesmo *select*.

O estado da entrada é armazenado no campo EE de seu descritor. Quando EE é maior que zero a entrada está EE-aberta (ou simplesmente aberta), estando fechada em caso contrário.

9.2 - Descritores de Entradas.

Da mesma forma que para tarefas, existem dois tipos de descritores de entradas. Um para família de entradas (DFE) e outro para entradas simples (DE).

DE

ETE NMET
:
ETE[1]
EE
NMET
PFTER
NTER
PROXDEA
TDE=DE

DFE

LSUPE
LINFE
ADEF
TDE=DFE

TDE - tipo de descritor de entrada (DE/DFE):

PROXDEA - próximo descritor de entrada aberta.

NTER - número de tarefas esperando "rendezvous" com esta entrada.

PFTER - primeiro da fila de tarefas esperando "rendezvous" com esta entrada.

NMET - número máximo de endereço de transferência desta entrada.

EE - estado da entrada. Se EE maior que zero a entrada está aberta.Caso contrário, está fechada.

ETE[i] - i-ésimo endereço de transferência da entrada.

ADEF - apontador para o primeiro DE da família de entradas.

LINFE - limite inferior do índice da família de entradas.

LSUPE - limite superior do índice da família de entradas.

Os descritores de membros de família de entradas são os mesmos que os de entradas simples.

O endereço textual de uma entrada é o endereço de seu descritor: nível da tarefa T onde ela é declarada, deslocamento do descritor em relação à base do RA de T (N_T,d_{DE}).

Uma entrada pode ser referida em duas situações distintas: *(a)*referências locais - em comandos *accept* e *select* e quando o seu atributo COUNT é investigado. Estas referências ocorrem somente quando a tarefa T está em execução. *(b)*referências externas - em chamadas a esta entrada.

O endereço efetivo da entrada é portanto carregado na pilha de duas formas diferentes, conforme a situação em que ela é referida. Nas referências locais,a

tarefa T está em execução e seu RAT é apontado por ARATC. Assim sendo, o endereço efetivo da entrada é dado por: ARATC+d_{DE}. O endereço efetivo da entrada é carregado pela instrução:

CEEL d_{DE} /carrega endereço de descritor de entrada local.

Nas referências externas, é necessário que T esteja ATIVA para que a entrada possa ser referida. Como a única forma de se referir externamente à uma entrada é através de uma chamada a esta entrada, a instrução que carrega o endereço da entrada externa é específica para esta situação.

CEEE d_{DE} /carrega endereço de entrada externa.

As entradas que pertencem à famílias também podem ter referências locais ou externas. O cálculo de seu endereço efetivo depende do seu índice na família, que é calculado e carregado na pilha antes da instrução que carrega o endereço da entrada. Este índice tem que estar dentro dos limites do intervalo da família.

Nas referências locais, o endereço efetivo de um membro de família é carregado por:

$[c]^+$ /código que calcula e carrega o índice da entrada.

CFEL d_{DFE} /carrega endereço de membro de família de entrada local.

Nas referências externas, a instrução que carrega o endereço efetivo para membros de família de entradas também é específica para chamadas de entrada.O endereço do descritor da tarefa T e o valor do índice da entrada já estão carregados na pilha.

CFEE d_{DFE} /carrega endereço de membro de família de entrada externa.

9.3 - Registro de Ativação de Entrada(RAEN).

O registro de ativação de entradas é ilustrado na figura 1 juntamente com os outros registros de ativação existentes.

Descrição dos campos:

TRA - tipo do registro de ativação. Identifica o registro de ativação, portanto é igual a RAEN. Este campo é utilizado pela instrução ABRT. (vide seção 10)

EME - endereço do código do manipulador de exceções da entrada. O manipulador de exceções de uma dada entrada é sempre a sua instrução FREN, que é a última instrução do código para comando *accept*.

ADCTA- este campo é utilizado para guardar o valor de ADCT anterior à ativação do RAEN. Ele é utilizado para restaurar o contexto quando a entrada termina sua execução.

ATCE - apontador para a tarefa que chamou esta entrada. É utilizado para levantar a exceção TASKING-ERROR na tarefa que chamou a entrada e para reativar esta tarefa quando a entrada termina sua execução.

ARPE - apontador para a região onde estão alocados os parâmetros efetivos da entrada.

9.4 - Declaração de Entradas.

A declaração de uma entrada (ou família) pode ocorrer na parte visível de uma tarefa T ou na parte declarativa do seu corpo. Esta entrada é então chamada de entrada local à tarefa T. Em ambos os casos, a compilação da declaração de

uma entrada tem como efeito a inclusão do seu nome, de seus parâmetros formais e dos tipos destes na tabela de símbolos. O tipo da entrada (simples ou família) também é guardado.

O compilador associa à cada entrada o endereço textual de seu descritor, que é alocado na PED de T (mesmo quando a entrada é declarada na parte visível).

Os endereços dos descritores das entradas de uma tarefa são determinados pelo compilador de tal forma que eles ficam agrupados na RDET, como descrito na seção 7.5.

O código gerado para a declaração de uma entrada é composto de duas partes: *(a)* código que elabora os tipos dos parâmetros e que calcula os valores iniciais para os "default parameters". *(b)* código que monta o descritor da entrada ou família de entradas.

Caso seja necessário alocar espaço para descritores de tipos ou inicializações de parâmetros, este espaço é alocado na PED de T, se a entrada é declarada no corpo da tarefa ou,se é declarada na parte visível, na PED e PDD da unidade on de a parte visível de T é declarada. (vide 7.6)

A parte *(b)* é sempre gerada no corpo de uma tarefa. A parte *(a)* pode ser gerada na especificação ou no corpo, de acordo com o lugar onde a entrada é declarada.

Código para declaração de entrada simples no corpo da tarefa.

[c]* /parte*(a)*

MDEN d_{DE},K /monta descritor de entrada simples - parte*(b)*.

Código para declaração de família de entradas no corpo da tarefa.

[c]* /parte *(a)*

cdi /carrega descritor de intervalo de família - parte *(b)*.

MDFE d_{DFE},K /monta descritor de família de entradas - parte*(b)*.

K - valor de NMET desta entrada.

Os códigos para entradas ou famílias declaradas na parte visível são os mesmos sem a parte *(a)*.

9.5 - Código para Chamada de Entradas.

Para que uma entrada E local a uma tarefa T seja chamada por uma tarefa Q é necessário que os parâmetros efetivos de E sejam elaborados e que T esteja ATIVA. Estas operações, a chamada propriamente dita e as operações de retorno dos parâmetros são efetuadas pelas instruções abaixo.

Código para chamada de entrada membro de família.

$[c]^+$ /1-carrega endereço do DT da tarefa onde a entrada está declarada

$[c]^+$ /2-código que calcula a expressão do índice da entrada

MPEN K /3-marca pilha para chamada de entrada

[c]* /4-elaboração dos parâmetros efetivos

CFEE d_{DFE} /5-carrega endereço de membro de família de entrada externa

CHEN /6-chama a entrada

[c]* /7-retorno dos parâmetros

RPFR /8-restaura a pilha após fim de "rendezvous"

Código para chamada de entrada simples.

Este código é o mesmo que para família de entradas, exceto as linhas 2 e 5:

```
RSRV    1     /2-reserva uma posição na pilha
CEEE    d_DE  /5-carrega endereço de entrada externa
```

K - Tamanho da parte estática dos parâmetros da entrada

d_{DFE} - Deslocamento do DFE em relação à base do RA de T

d_{DE} - Deslocamento do DE em relação à base do RA de T

9.6 - Código para o Comando *accept*.

Um comando *accept* para uma entrada E é traduzido para as instruções abaixo:

```
        [c]+        /carrega endereço do descritor de E que é local
        ABRE        /abre a entrada
        ACEN   RM   /aceita a entrada
        [c]+        /comandos da entrada
RM:     FREN        /fim de rendezvous
```

9.7 - Código para Comando *select*.

Um comando *select* pode ou não conter a cláusula *else*. Não tendo a cláusula *else* ele pode conter alternativas selecionáveis com comandos *delay*. O código para o *select* é determinado conforme ele tenha ou não cláusula *else*

Código para *select* sem cláusula *else* com n alternativas selecionáveis.

```
        ISLT    RB /inicia comando select
        [cas]+     /código para as alternativas selecionáveis
RAn:    SELD       /seleciona entrada ou possível delay
RB:     FSLT       /fim de select
        c          /instrução seguinte ao select
        :
        :
```

Código para *select* com cláusula *else* com n alternativas selecionáveis.

```
        ISLT    RB  /inicia comando select
        [caa]+      /código para a alternativa accept
RCE:    [c]*        /código para os comandos da cláusula else
        DSVS    RB  /desvia para instrução seguinte ao select
RAn:    SELE    RCE /seleciona entrada com cláusula else
RB:     FSLT        /fim de select
        c           /instrução seguinte ao select
        :
        :
```

O código para cada alternativa selecionável (cas) pode ser código para alternativa *accept* (caa) ou código para alternativa *delay* (cad).

A ordem das alternativas selecionáveis em um *select* é relevante, pois cada alternativa faz referências à rótulos da alternativa seguinte. Assim, a descrição de caa e cad vem acompanhada de um índice i que identifica a posição da alternativa no *select*. A primeira alternativa tem índice zero.

Código para a i-ésima alternativa *accept* (caa).

```
RA_i:  [[c]]+          /código para a expressão guardiã
       DSVF   RA_i+1]
       [c]+            /carrega endereço do DE da entrada que é local à tarefa
                       /corrente
       ABRS   RA_i+1   /abre entrada em select
       [c]+            /carrega endereço do DE da entrada que é local à tarefa
                       /corrente
       ACEN   RM_i     /aceita chamada de entrada
       [c]*            /código para a parte crítica da entrada
RM_i:  FREN            /fim de rendezvous
       [c]*            /código para a parte não-crítica
       DSVS   RB       /desvia para a instrução seguinte ao select
```

Código para a i-ésima alternativa *delay* (cad).

```
RA_i:  [[c]]+          /código para a expressão guardiã desta alternativa
       DSVF   RA_i+1]
       [c]+            /código para a expressão do valor do delay
       MADL   RA_i+1   /marca alternativa delay
       [c]*            /código para a parte não crítica
       DSVS   RB       /desvia para a instrução seguinte ao select
```

Não é necessário gerar o rótulo RA0 na primeira alternativa, pois ela não é referida. A última alternativa faz referência à RAn que rotula a instrução SELD ou SELE.

9.8 - Montagem dos Descritores de Entradas.

O descritor de uma entrada é montado quando da sua declaração. A montagem de um DFE (que é efetuada pela instrução MDFE) consiste em atribuir aos campos LSUPE e LINFE o valor do DI da família previamente carregado na pilha; em atribuir a ADEF o valor corrente de TOPO e em fazer TDE igual a DFE.

Para cada membro da família é alocado (a partir da posição apontada por TOPO) e montado um DE.

O tamanho de um DE depende do valor NMET da entrada. O valor de NMET para cada membro de uma família pode ser diferente. No entanto, optou-se em tomar um único valor de NMET para todos os membros de uma família. O valor é escolhido pelo compilador como sendo o máximo entre os NMETs da família.

Sempre que necessário, o tamanho ocupado por um DE será referido por |DE|.

A montagem de um DE consiste em inicializar os campos TDE com DE; NTER com zero; PFTER com NIL; EE com zero e NMET com o valor K que é fornecido pelo compilador como parâmetro das instruções MDEN e MDFE.

9.9 - Chamada de Entradas.

Do ponto de vista da tarefa Q que chama uma entrada E declarada em T, a entrada é como que um subprograma . A chamada é efetuada e após algum tempo retorna e Q segue sua execução.

Quando E é chamada, seus parâmetros são avaliados e alocados na pilha de Q. Para que a chamada seja realmente efetuada é necessário que T esteja ATIVA e que T e Q não sejam a mesma tarefa (isto pode ocorrer e causaria um "dead-lock" em T).

A alocação dos parâmetros segue o esquema apresentado na seção 4 . Como neste caso não existe um RA a ser ativado, utiliza-se um pseudo RA composto de um único campo PADCTA. Este campo é utilizado para guardar o valor anterior de PADCT.

Como não há mudança de contexto (ADCT, D e NC conservam seus valores), o acesso a estes parâmetros é feito utilizando-se o valor de PADCT de Q ao invés de um dos valores do vetor D.

A figura 9 ilustra a situação da tarefa T imediatamente após a execução da instrução CHEN (chamada de entrada) efetuada pela unidade *U* ativada em Q e antes da chamada ser aceita.

A tarefa Q é a única tarefa na FTER da entrada E. Se E estava aberta quando Q a chamou, então o processo de aceitação já teve inicio. A tarefa T, que estava na FTD ou suspensa esperando chamada de entrada, foi reativada (colocada numa FTP). A próxima instrução a ser executada por T será ACEN, que aceitará a chamada a E. Como outras tarefas podem ser executadas antes que o SCHEDULER transfira a execução para T, a instrução ACEN testa se existe tarefa na fila FTER da entrada E, pois Q pode ter terminado sua execução neste interim devido a um comando *abort* executado por uma destas outras tarefas.

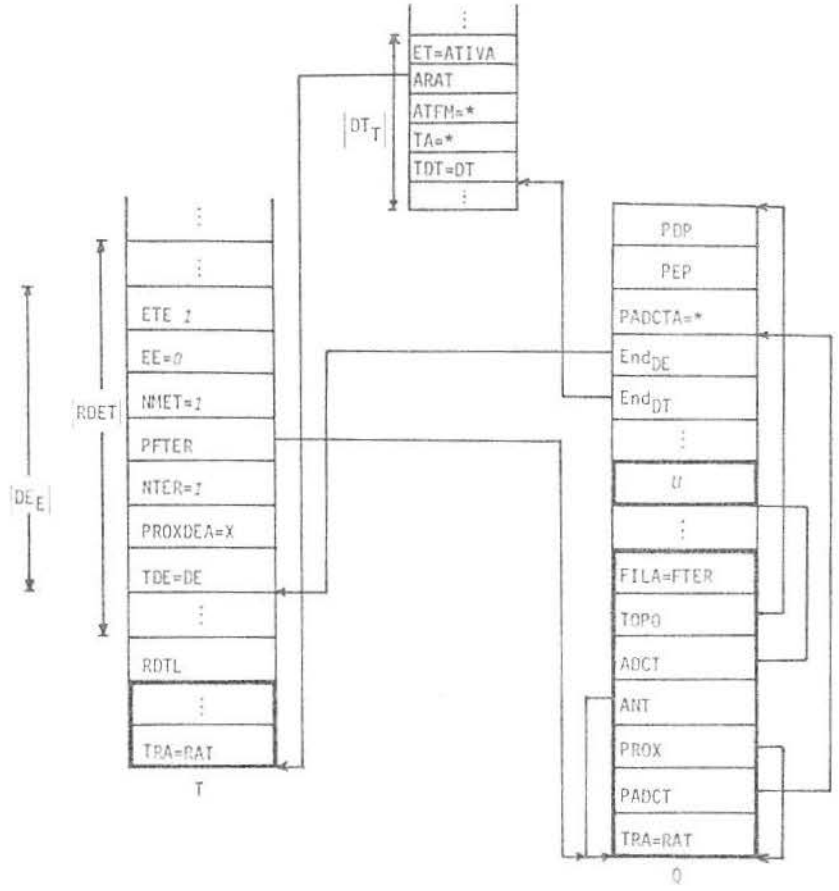


Figura 9

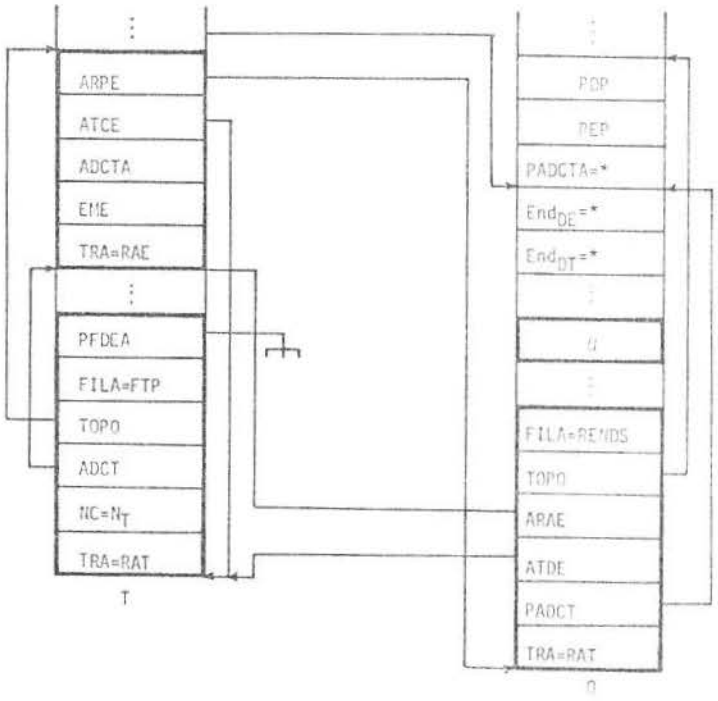


Figura 10

Quando T é reativada, todas suas entradas são fechadas. Se acontecer da FTER de E estar vazia quando ACEN é executada, a exceção TASKING-ERROR é levantada em T, da mesma forma que quando Q termina durante o "rendezvous" (ou seja, após a chamada ter sido realmente aceita).

Se E estava fechada quando Q a chamou, a situação de Q e T é a mesma mostrada na figura 10 . A tarefa T está ATIVA e com FILA igual a qualquer dos valores possíveis menos FTE e SEDA.

9.10 - Aceitação de Chamada de Entrada e "rendezvous".

Quando a tarefa T executa um *accept* para a entrada E, a FTER de E pode estar:
VAZIA - A entrada é aberta, seu descritor inserido na FDEA e T fica suspensa esperando chamada de entrada.
NÃO VAZIA - O processo de aceitação tem início.

Por processo de aceitação entende-se o período que a tarefa T passa desde que a instrução ACEN termina sua execução. A execução de ACEN não implica obrigatoriamente em aceitação e início de "rendezvous", como descrito na seção 9.9.

Quando uma chamada a E é realmente aceita, o RAEN de E é alocado no topo da pilha de T e inicializado, como ilustrado na figura 11 . O campo ATCE aponta para a pilha da tarefa Q, o campo ARPE para a região onde estão alocados os parâmetros de E (que é apontado pelo campo PADCT de Q). O valor de ADCT de T é salvo em ADCTA e o contexto é atualizado fazendo ADCT de T apontar para este RAEN. O campo EME passa a apontar para a instrução FREN da entrada, que é também o ME da entrada.

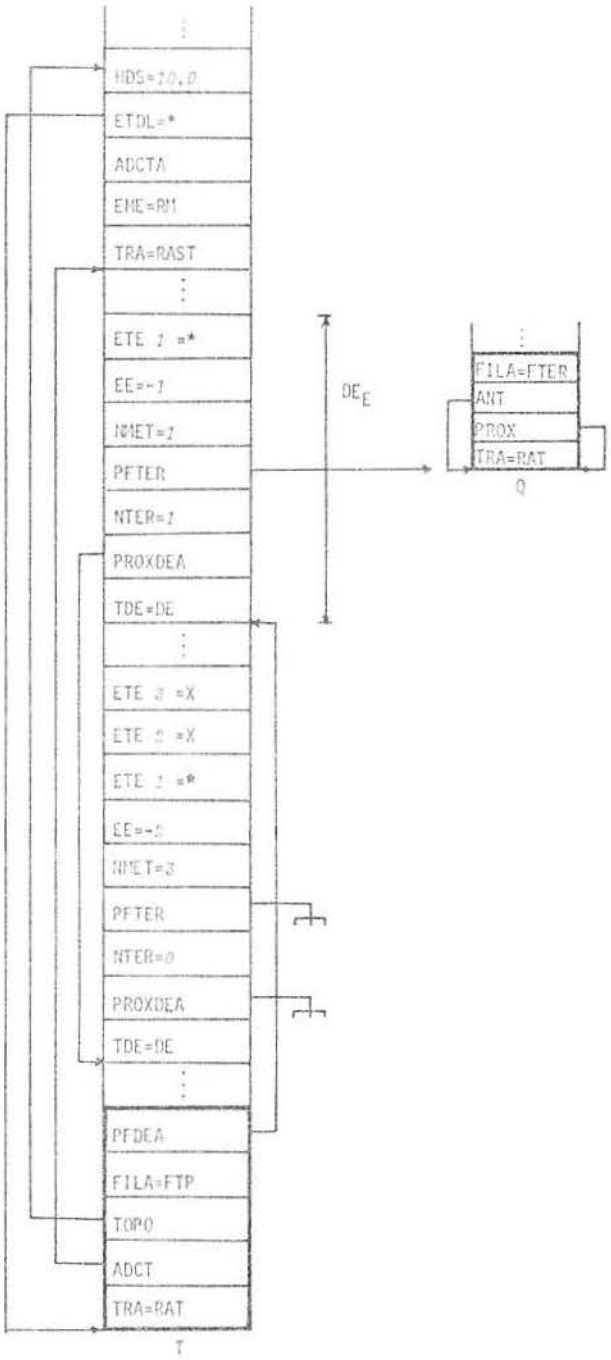


Figura 11 - Situação de uma tarefa T imediatamente após a aceitação de uma chamada de entrada

A tarefa Q é retirada da FTER de E ficando suspensa com FILA igual a RENDS(em "rendezvous"). Os campos PROX e ANT de Q são utilizados como ATDE e ARAE, que passam a apontar para a pilha de T e para o RAEN de E respectivamente.

As instruções de E começam então a ser executadas, as tarefas T e Q estão em "rendezvous".

Os parâmetros de E são acessados através de ARPE. O endereço efetivo de um parâmetro P de E, passado por cópia, é dado por ARPE(ADCT)+d_P, onde d_P é o deslocamento relativo de P à base do pseudo RA em Q. Os parâmetros passados por referência necessitam de uma indireção a mais.

Os campos ARAE e ATCE de Q são utilizados para levantar a exceção TASKING-ERROR em T quando Q é abortada.

Quando a instrução FREN (fim de "rendezvous") é executada, a tarefa Q é inserida na FTE ou numa FTP conforme o seu estado SEMI-ATIVO ou ATIVO respectivamente. O RAEN é desativado e se T está com exceção pendente, a exceção é levantada em Q.

O valor de PADCT salvo em PADCTA é restaurado e CPCT é atualizado com o endereço do ME de *U*. A execução de T é desviada para o ME da unidade onde se encontra o comando *accept*. No exemplo ilustrado pela figura 11 , esta unidade é a própria tarefa T.

9.11 - Processo de Seleção

A implementação do comando *select* utiliza de um RA. Este registro de ativação RAST, é ilustrado na figura 1 e seus campos são descritos a seguir.

TRA - tipo de registro de ativação. Portanto, igual a RAST.

EME - endereço do ME do *select*, que é sempre o endereço da instrução FSLT gerada para este *select*.

ADCTA - valor anterior de ADCT.

ETDL - endereço de transferência de *delay* em *select*.

HDS - hora de despertar em *select*.

A execução de um *select* é efetuada em dois estágios: *(a)* determinação das alternativas abertas e *(b)* seleção.

Durante o primeiro estágio, as alternativas são abertas ou não, dependendo do valor de suas expressões guardiãs. As alternativas com comando *accept* (para uma entrada E) são abertas através da inserção do descritor de E na FDEA. Se E já estava na FDEA, então é porque E está n-aberta (com n>1). Esta informação é armazenada no DE de E, decrementando de um o valor de EE e salvando em ETE[-EE] o valor do endereço de transferência para cada alternativa *accept* (para a entrada E) aberta. Quando um *select* inicia sua execução, todas as entradas estão fechadas (com EE=0). Desta forma, as entradas estão fechadas à vista das outras tarefas e assim o controle de qual entrada é selecionada fica a cargo do próprio *select* (através das instruções SELE e SELD).Se estas entradas fossem abertas com valores positivos de EE e se ocorresse uma chamada à entrada E durante o estágio *(a)*, esta chamada seria aceita e a execução do

select não terminaria.

Dentre as alternativas *delay* abertas, somente aquela que implicar no menor tempo de dormência (também chamado valor do *delay*) é considerada na seleção.

No estágio *(a)* a alternativa de menor valor de *delay* tem seu endereço de transferência salvo em ETDL e seu valor salvo em HDS.

Como a tarefa que está executando o estágio *(a)* de um *select* pode ser interrompida (durante a avaliação das expressões guardiãs e dos valores dos comandos *delay*), o campo CPCT desta tarefa não pode ser utilizado para armazenar o endereço de transferência de *delay*.

O campo HDS é utilizado no lugar de HD, pois um comando *delay* pode ser executado como efeito colateral da avaliação das expressões (guardiãs e de valores de *delay*) no estágio *(a)*.

São proibidas nesta implementação, a execução de comandos *accept* e *select* como efeito colateral da avaliação das expressões no estágio *(a)*. A execução destes comandos nestas situações não fariam mesmo muito sentido e complicariam ainda mais a implementação do comando *select*.

Findo o estágio *(a)* as alternativas selecionáveis abertas estão guardadas nos campos PFDEA (alternativas *accept*), HDS e ETDL (alternativa *delay*).

A seleção pode ser feita de duas maneiras, conforme o *select* tenha ou não cláusula *else*.

Se o *select* contiver cláusula *else*, a seleção é feita pela instrução SELE. Se não houver alternativa aberta, a execução é desviada para os comandos da cláusula *else*. Em caso contrário, os valores dos campos EE das entradas na FDEA são transformados em valores positivos e os valores de EE das entradas com possibilidade de "rendezvous" imediatos são somados. Assim, obtem-se o número total de endereços de transferência com possibilidade de "rendezvous" imediato. Este número corresponde ao número de alternativas *accept* abertas que podem ser selecionadas imediatamente.

Se este número N for igual a zero, então nenhum "rendezvous é possível no momento e a execução é desviada para os comandos da cláusula *else*.

Em caso contrário, uma das N alternativas é escolhida aleatoriamente e a primeira tarefa da FTER da entrada correspondente à alternativa escolhida terá sua chamada aceita.

Se o *select* não contiver cláusula *else*,então podem existir alternativas *delay* e a seleção é feita pela instrução SELD.

Não havendo alternativas *accept* abertas (quando PFDEA=NIL), então a exceção SELECT-ERROR é levantada. Em caso contrário, o número N acima referido é calculado e se for maior que zero, uma alternativa é selecionada aleatoriamente, como explicado acima.

Se N for igual a zero, então a tarefa corrente (a que está executando *select*) é alocada na FTD,ficando dormente se existir alternativa *delay* aberta ou, não havendo alternativa *delay* aberta, a tarefa passa a suspensa esperando chamada

de entrada (com FILA=SECE).

A existência de alternativa *delay* aberta pode ser verificada através do valor de HDS, que, sendo maior que zero, implica na existência de alternativa *delay* aberta e nesse caso o endereço de transferência do *delay* considerado está armazenado no campo ETDL.

O término da execução de um *select* é sempre executado pela instrução FSLT, mesmo quando ocorre uma exceção. Esta instrução desativa o RAST e se existir exceção pendente, as entradas na FDEA são fechadas e a execução é desviada para o ME da unidade que contém o comando *select*. As entradas são fechadas pois a exceção pode ter ocorrido durante o estágio *(a)* e certos descritores de entradas podem ter sido alterados.

9.12 Interrupções Transformadas em Chamadas de Entradas.

O ADA permite a associação de interrupções com entradas. A ocorrência destas interrupções são tratadas como uma chamada à entrada associada. Ou seja, quando a interrupção é aceita, a execução é desviada para o código que executa esta chamada. Este código é gerado pelo compilador em uma posição especial da MP, conforme apresentado em 9.5. Uma pseudo tarefa deve ser criada para fazer o papel da tarefa que chama a entrada.

10 - SUBPROGRAMAS DECLARADOS NA PARTE VISÍVEL DE UMA TAREFA

Nesta seção é estudada a implementação dos subprogramas declarados na parte visível de uma tarefa.

Estes subprogramas, por serem visíveis no escopo de declaração da tarefa T onde eles são declarados, requerem uma implementação mais complexa que a usualmente utilizada para subprogramas em linguagens como o PASCAL.

Estes subprogramas são chamados subprogramas externos (SE) e podem ser chamados tanto dentro do escopo de T (como qualquer outro subprograma), quanto fora deste escopo.

Estas chamadas são denominadas chamadas locais e externas, respectivamente.

Se estes subprogramas retornam valor, quer sejam funções ou procedimentos que retornam valor, eles são chamados de funções externas (FE).

Nas chamadas externas de SEs (e só nestas chamadas) é possível que a unidade onde ele está declarado (ou seja, a tarefa T) não esteja ativa, e assim o estado de T deve ser testado antes da chamada ser efetuada.

Como as declarações de T são visíveis no SE e não estão acessíveis quando o SE é chamado externamente, é necessário que se altere o contexto de tal forma a tornar acessíveis estas declarações.

Se a tarefa T termina e o SE está sendo executado externamente, então a exce-

ção TASKING-ERROR deve ser levantada na unidade que chamou este SE.Para que isto seja possível, a tarefa T deve manter uma lista de todas as ativações externas de seus SEs.

Nas chamadas locais, T está ativa, suas declarações são acessíveis e é impossível que T termine sem que suas ativações locais tenham terminado. Assim nenhuma das operações acima descritas para chamadas externas de SEs são necessárias quando as chamadas são locais.

Por uma questão de eficiência, as chamadas externas e locais são efetuadas de maneiras diferentes.

10.1 - Registro de Ativação de Subprogramas Externos (RASE).

Os campos de um RASE são:

TRA - Tipo do registro de ativação, portanto sempre igual a RASE.

EME - Endereço do manipulador de exceções de SE.

DA - Valor anterior de $D[N_{SE}]$.

NDTL - Número de descritores de tarefas locais (os descritores de membros de família não são contados).

NA - Nível anterior. Nível léxico da unidade que chamou o SE.

ER - Endereço de retorno. Contém o endereço para onde a execução deve ser desviada após a execução do SE.

PROXLSE - Apontador para o próximo RASE na lista das ativações externas de SE (LSE) da tarefa T onde o SE é declarado.

ANTLSE - Apontador para o RASE anterior na LSE de T.

DANT - Valor anterior de $D[N_T]$.

AP - Apontador para o RAT da tarefa sob cujo controle o SE chamado externamente está sendo executado.

TC - Tipo de chamada (EXTERNA/LOCAL). Este campo é utilizado para determinar quais operações devem ser executadas para o retorno do SE.

Os campos AP, DANT, ANTLSE e PROXLSE só são utilizados quando a chamada é externa.

10.2 - Código para Chamada de Subprograma Externo.

A seguinte seqüência de instruções é gerada:

```
[RSRV    |RF|]  /reserva |RF| posições na pilha para o resultado FE
MPSE     K,TCH  /marca a pilha para SE
[c]*            /código para a elaboração dos parâmetros
cds             /desvia para subprograma
[c]*            /operações de retorno dos parâmetros
```

Quando a chamada é local cds é substituído por:

```
DSUB     RSE    /desvia para subprograma
```

Sendo externa cds é substituido por:

```
DVSE     N_A,d_DT,RSE /desvia para subprograma externo
```

Onde: |RF| é o tamanho ocupado pelo resultado da FE.

K é o tamanho do RASE mais os tamanhos das partes PEP e PED.

TCH é o tipo de chamada, LOCAL ou EXTERNA.

RSE é o endereço do código gerado para o SE que está sendo chamado.

N_A, d_{DT} é o endereço textual do descritor da tarefa onde o SE está declarado.

A primeira instrução (RSRV |RF|) só é gerada quando a chamada é para um FE.

10.3 - Código para Subprogramas Externos.

Como o SE só é executado quando ele é chamado, a primeira instrução gerada é um desvio para a instrução seguinte ao seu código.

```
        DSVS    RB                      /desvia sempre
RSE:    ENSE    RM,N_T,d_DT,N_DTL       /entra em SE
        [c]*                            /código para as declarações do SE
        [c]*                            /código para os comandos do SE
R:      TNTA                            /testa o número de tarefas locais ativas
        RTSE                            /retorno de procedimento externo
RM:     [c]+                            /código para manipulador de exceções do SE
RB:     c                               /instrução seguinte ao SE
        :
```

Onde: N_T é o nível da tarefa T onde o SE está declarado.

d_{DT} é a posição do DT de T relativa ao RA da unidade onde T é declarado.

N_{DTL} é o número de descritores de tarefas locais, não incluíndo os descritores de membros de famílias.

10.4 - Chamada de Subprogramas Externos.

Se uma FE (ou qualquer função) é chamada, o espaço para seu resultado é reservado na pilha antes da alocação do RASE. Assim, quando a FE retorna, o seu resultado é deixado no topo da pilha . No mais, a chamada e o retorno de uma FE é o mesmo que para um SE.

Seja um subprograma externo SE declarado na tarefa T e chamado externamente pela unidade *U* ativada na tarefa Q. A tarefa T é declarada na unidade A de nível léxico N_A.

A figura 12 ilustra a situação da tarefa Q durante o processo de chamada a SE. O TOPO de Q aponta (linha cheia) para o fim da PEP após a execução de MPSE. Após a elaboração dos parâmetros, a parte PDP já está alocada e TOPO aponta como mostra a linha pontilhada.

O valor de ADCT não é alterado durante a elaboração dos parâmetros. A base do RASE é apontada pelo campo PADCT cujo valor anterior foi salvo no campo DA do RASE.

Note que, dadas as regras de visibilidade da ADA, se T é visível num dado ponto é porque A está necessàriamente ativa neste ponto. Assim o descritor de T pode ser investigado. O valor de $D[N_A]$, quando Q está sendo executada, é o endereço da base do RA de A.

Se a tarefa T estiver inativa,a exceção TASKING-ERROR é levantada em Q e a execução desviada para o ME de *U*.

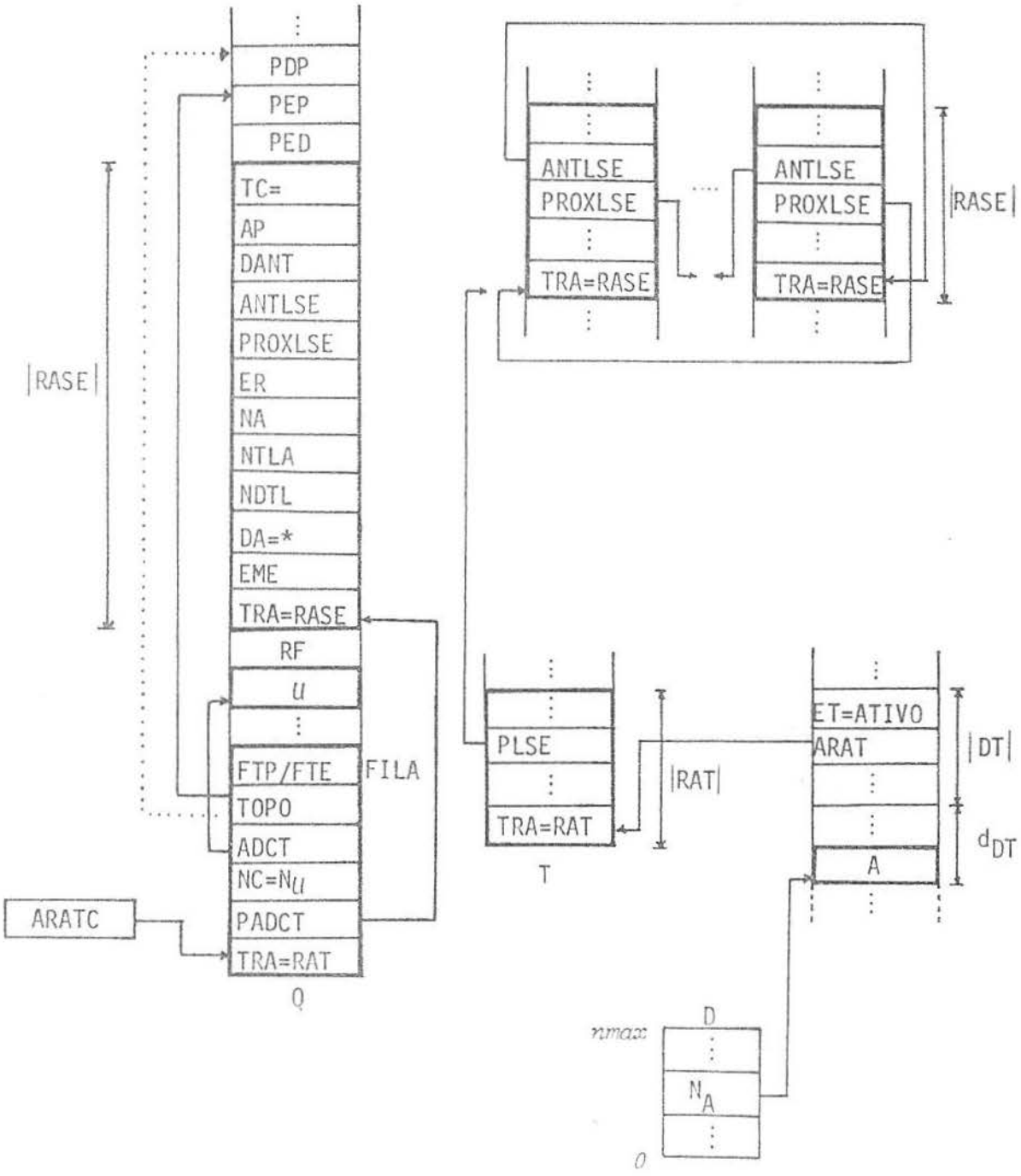

PDP
PEP
PED
TC=
AP
DANT
ANTLSE
PROXLSE
ER
NA
NTLA
NDTL
DA=*
EME
TRA=RASE
RF
u
FTP/FTE
TOPO
ADCT
NC=Nu
PADCT
TRA=RAT
FILA
|RASE|
ARATC
Q
ANTLSE
PROXLSE
TRA=RASE
|RASE|
PLSE
TRA=RAT
|RAT|
T
ET=ATIVO
ARAT
A
|DT|
dDT
nmax
D
NA
0

Figura 12

Em caso contrário, o SE é ativado com ADCT passando a apontar para a base de seu RASE, o valor anterior de PADCT (salvo no campo DA de RASE) é restaurado, o endereço de retorno é salvo no campo ER e a execução desviada para o código do SE.

A instrução ENSE é então executada. Os campos do RASE são inicializados. O endereço do manipulador de exceções do SE é guardado em EME. O valor de NC (nível corrente) de Q é atualizado com o N_{SE} e o nível de U é salvo no campo NA (nível anterior). No campo DA é salvo o valor de $D[N_{SE}]$ e $D[N_{SE}]$ passa a apontar para a base do RASE.

O campo NTLA é inicializado com zero e NDTL com o número de descritores de tarefas locais ao SE. O RASE é inserido na LSE de T através dos campos PROXLSE e ANTLSE. No campo DANT é salvo o valor de $D[N_T]$, e $D[N_T]$ passa a apontar para o RA da tarefa T . Em AP é salvo o endereço da base do RASE. Esta informação é utilizada para se levantar a exceção TASKING-ERROR em Q quando a tarefa T terminar e o SE ainda estiver ativado.

A figura 13 ilustra a situação imediatamente após a execução de ENSE,com TOPO apontando para o fim da PDP. Após a elaboração das declarações de SE,TOPO aponta para o fim da PDD (linha tracejada). O RASE está na lista de ativações externas de SEs de T.

A tarefa Q está sendo executada e o vetor de registradores D com seus valores correntes é também ilustrado. No RASE os valores denotados por D' são os valores de D antes da execução de ENPE.

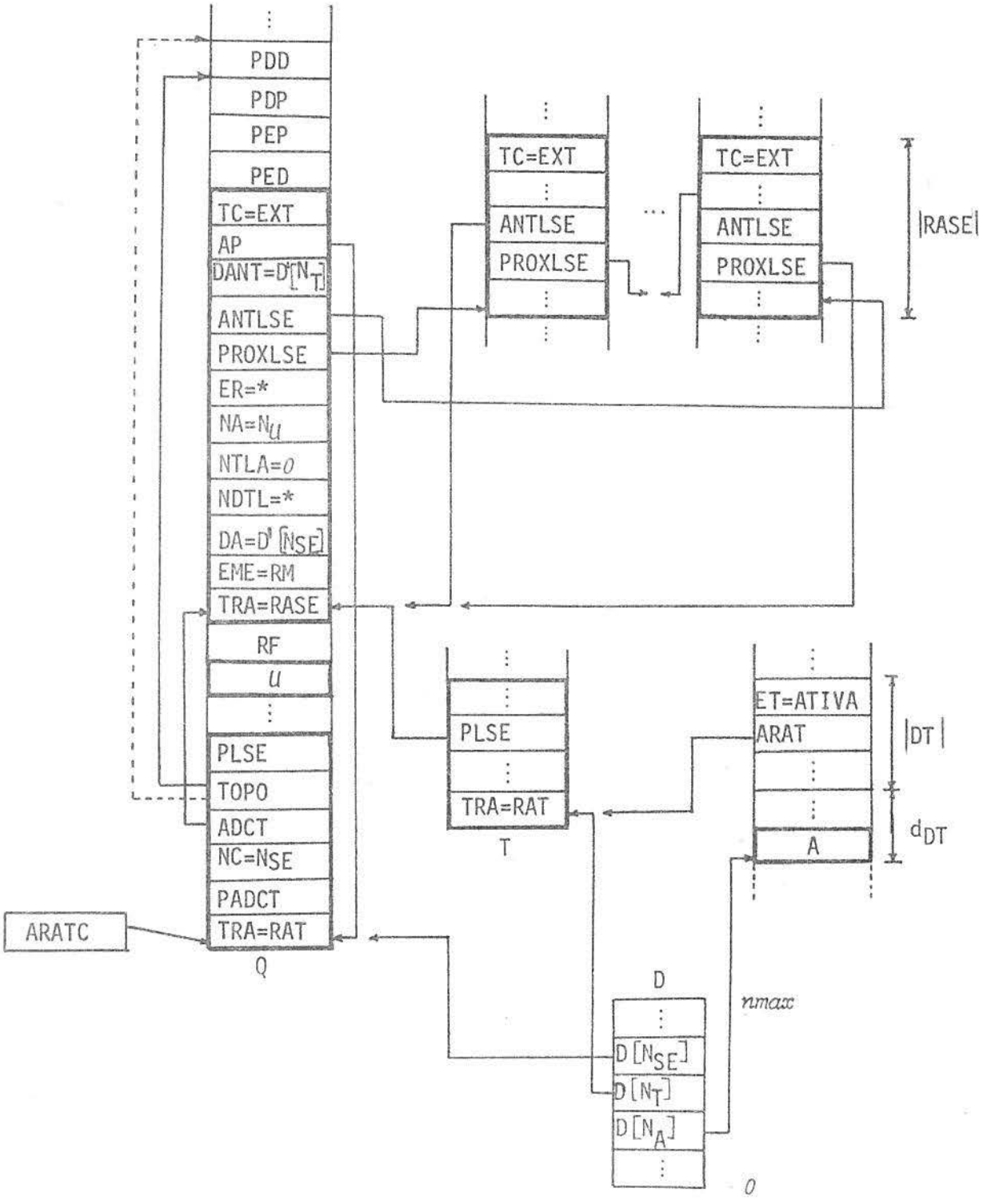

PDD
PDP
PEP
PED
TC=EXT
AP
DANT=D'[N_T]
ANTLSE
PROXLSE
ER=*
NA=N_u
NTLA=0
NDTL=*
DA=D'[N_SE]
EME=RM
TRA=RASE
RF
u
PLSE
TOPO
ADCT
NC=N_SE
PADCT
TRA=RAT
Q
ARATC
TC=EXT
ANTLSE
PROXLSE
TC=EXT
ANTLSE
PROXLSE
|RASE|
PLSE
TRA=RAT
T
ET=ATIVA
ARAT
A
|DT|
d_DT
D
D[N_SE]
D[N_T]
D[N_A]
nmax
0

Figura 13

Se a chamada é local, então o SE é declarado em Q, os campos DANT, PROXLSE e ANTLSE não são utilizados e $D[N_Q]$ não é alterado, pois já aponta para a base do RA de Q.

10.5 - Retorno de Subprograma Externo.

O retorno de um SE é implementado por duas instruções: TNTA e RTSE.

Se o SE tiver tarefas locais ativas (NTLA>0) então a tarefa Q é suspensa com ativação pronta para terminar (FILA=SAPT). Estas operações são efetuadas pela instrução TNTA. A tarefa Q fica nesta situação até que NTLA seja zero, quando Q é reativada.

Se a chamada foi externa (TC=EXTERNA), então o RASE é retirado da LSE de T, o valor de $D[N_T]$ salvo em DANT é restaurado.

As operações seguintes são comuns às chamadas locais e externas. Os campos NC, TOPO e ADCT do RA de Q são atualizados com N_U salvo em NA, com o valor de ADCT e com o valor de $D[N_U]$ que aponta para a base do RA de *U* respectivamente.

A execução é então desviada para o ME de *U* ou para o endereço de retorno do SE salvo no campo ER do RASE, dependendo se Q tem exceção pendente ou não. Se Q tem exceção pendente é porque uma exceção foi levantada no SE e não tratada localmente. Esta exceção é então propagada para *U*.

11 - COMANDO *abort*.

O comando *abort* termina incondicionalmente a execução de tarefas.

Um comando

abort T1,...Tn;

é traduzido por:

```
[[c]+        /carrega endereço do descritor de uma tarefa
     ]+      /carrega os endereços dos descritores das n tarefas
ABRT   n     /aborta tarefas
```

A instrução ABRT desativa as tarefas T1,...Tn que estiverem ativas.

O processo de desativação de uma tarefa T é um processo recursivo bastante complicado. É recursivo pois as tarefas locais ativas também devem ser desativadas.

A exceção TASKING-ERROR deve ser levantada nas tarefas que estão esperando "rendezvous" com entradas de T e nas tarefas que chamaram subprogramas de T visíveis externamente.

Todas as unidades ativadas em T também devem ser desativadas, ou seja, operações parecidas às efetuadas por suas instruções de retorno devem ser executadas. Se estas unidades contêm tarefas locais ativas, então estas tarefas devem ser desativadas.

Para efetuar estas desativações, foi projetada a primitiva DESATIVA(R_T,R_U) on-

de R_T aponta para o RA de uma tarefa T e R_U aponta para o RA de uma unidade U ativada em T. Esta primitiva desativa todas as unidades ativadas na tarefa T, até a ativação de U, inclusive.

Assim, uma tarefa T é desativada por:

DESATIVA(R_T,R_T);

Note que quando a exceção TASKING-ERROR é levantada em uma tarefa S que chamou um SE de T, podem existir várias ativações em S posteriores à ativação do SE. Todas estas ativações devem ser desativadas, inclusive a do SE. Assim, a exceção será tratada pelo ME da unidade que chamou o SE.

As mesmas desativações devem ser efetuadas quando a tarefa T termina normalmente e a exceção TASKING-ERROR é levantada nas tarefas que chamaram seus SEs

Na figura 14 a desativação da tarefa T implica em:

1 - Desativar as unidades A e B ativadas em T.
2 - Desativar a tarefa X que está ativa e é local à T.
3 - Desativar as unidades C, D e SE em S e levantar TASKING-ERROR em S (que será tratada por U).
4 - Desativar o RAT de T decrementando o campo NTLA da unidade de declaração de T.

Dada a extensão da descrição da primitiva DESATIVA ela não foi incluida no Apêndice I.

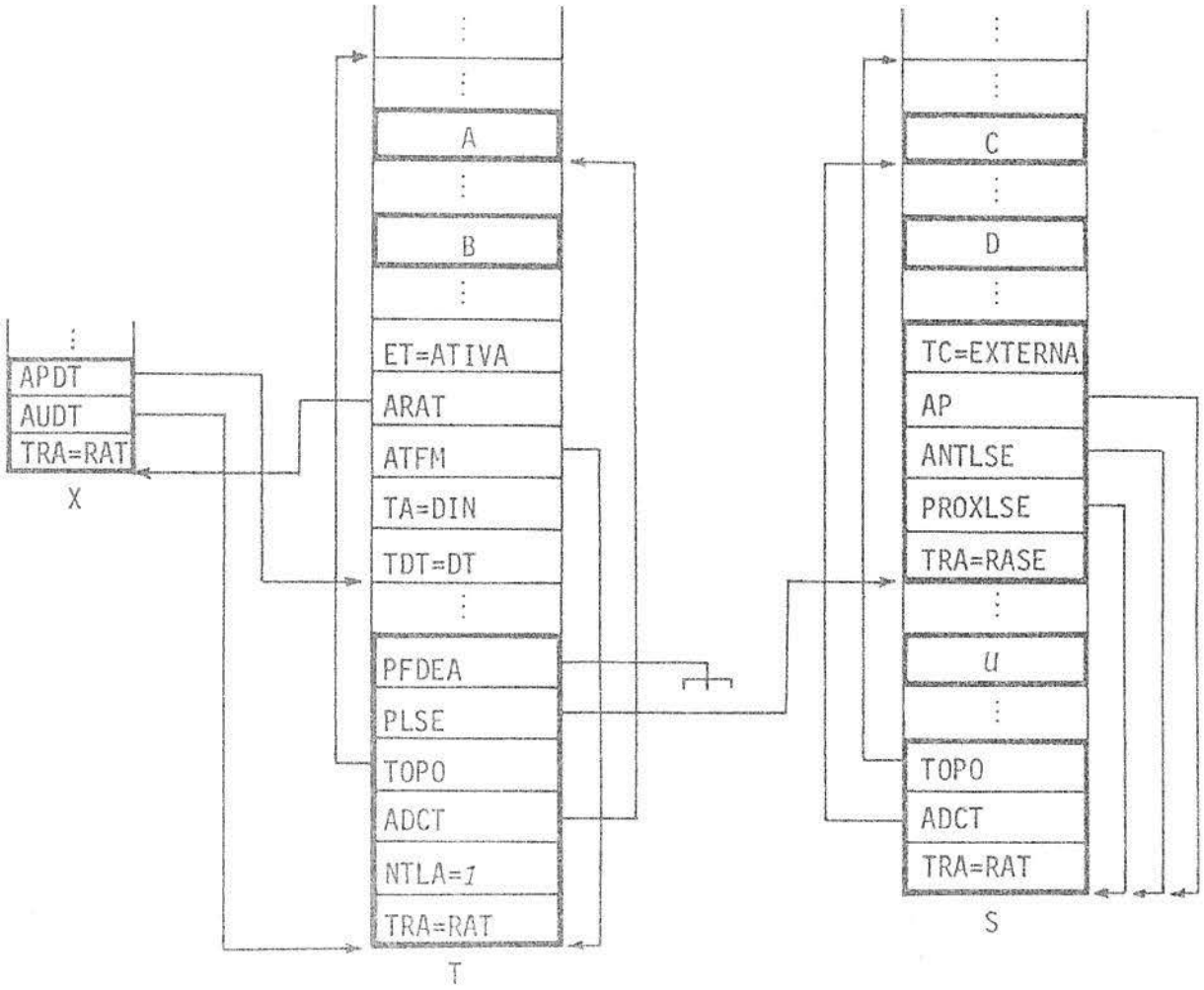

APDT
AUDT
TRA=RAT
X
A
B
ET=ATIVA
ARAT
ATFM
TA=DIN
TDT=DT
PFDEA
PLSE
TOPO
ADCT
NTLA=1
TRA=RAT
T
C
D
TC=EXTERNA
AP
ANTLSE
PROXLSE
TRA=RASE
u
TOPO
ADCT
TRA=RAT
S

Figura 14

CAPÍTULO III

CONCLUSÕES

Tentou-se projetar a MEADA de forma a obter como resultado uma implementação simples, segura e eficiente. A complexidade do sistema proposto advém das próprias características da ADA, principalmente do processamento paralelo e de suas primitivas de comunicação entre as tarefas dos subprogramas externos e das exceções e seu tratamento. Uma implementação da ADA sem tarefas seria bastante simples e semelhante ao PASCAL ou ALGOL 60.

A inclusão dos estados SEMI-ATIVO e SEMI-INATIVO, e o processo de desativação que simula os retornos das unidades ativas, colaboram bastante para a segurança deste sistema de execução.

O sistema apresentado pode ser melhorado em diversos pormenores, que o tornariam mais eficiente.

A implementação do comando *select* pode ser efetuada sem a restrição imposta neste trabalho, sendo porém menos eficiente.

O presente trabalho mostra a viabilidade da implementação da ADA. Dadas as características da própria linguagem, sua implementação em máquinas reais fica restrita às máquinas que contenham certas características mínimas. Dentre estas características citam-se: Relógio de tempo real e interrupção por tempo. Um certo número de registradores é aconselhável; nem todos os registradores da MEADA tem necessariamente que corresponder a registradores numa máquina real. Os registradores ESTPRO, CP, RELOG, DESP, HTIT e ARAC (os dois últi

mos quando existirem) devem corresponder a registradores da máquina real.Dada a freqüência com que são utilizados os registradores do vetor D e ARATC, é conveniente que eles correspondam a registradores na máquina real, pois isto aumentaria a eficiência da implementação.

Para completar este trabalho faltam, além dos comandos citados no capítulo I, a implementação dos tipos básicos da ADA (inteiros, reais e caracteres) que não apresentam maiores dificuldades.

Finalmente deve-se lembrar que o projeto do sistema de execução é apenas uma parte do trabalho de implementação da linguagem.

APÊNDICE I

INSTRUÇÕES E PRIMITIVAS DA MEADA

```
instrução ABRE; (* abre entrada *)
inicio (* o endereço do DE da entrada está no topo da pilha *)
  H0:=(TOPO); (* End_DE *)
  se PFTER(H0)=NIL então
    inicio (* não tem nenhuma tarefa esperando rendezvous com a entrada *)
      EE(H0):=1;
      ETE[1](H0):=CP;
      PROXDEA(H0):=NIL;
      PFDEA:=H0;
      SUSPENDE(PFTP[PRIO]);
      FILA:=SECE; (* suspensa esperando chamada de entrada *)
      TAET:=TAET+RELOG;
      SCHEDULER;
    fim;
fim;
```

```
instrução ABRS R; INTERROMPÍVEL (* abre entrada em select *)
inicio (* R é o endereço da próxima alternativa selecionável.
          O endereço do DE da entrada está no topo da pilha *)
  H0:=(TOPO); (* End_DE *)
  TOPO:=TOPO-1;
  se EE(H0)=0 então (* o DE desta entrada ainda não está na FDEA *)
    inicio (* insere na FDEA *)
      PROXDEA(H0):=PFDEA;
      PFDEA:=H0;
    fim;
  EE(H0):=EE(H0)-1;
  ETE[EE(H0)](H0):=CP; (* salva o EE-ésimo endereço de transferência da
                          entrada *)
  CP:=R; (* desvia para próxima alternativa do select *)
fim;
```

```
instrução ABRT N; (* aborta tarefas *)
inicio (* os endereços dos DTs das N tarefas estão carregados na pilha *)
  TAET:=TAET+RELOG;
  para H0:=1 até N faça
  inicio
    H1:=(TOPO); TOPO:=TOPO-1; (* H1 contém o endereço do DT da tarefa T *)
    se ET(H1) em [ATIVO, SEMI-INATIVO] então
      (* executa os retornos incondicionais de todas as unidades ativadas em
         T e o término incondicional de T *)
      DESATIVA(ARAT(H1),ARAT(H1));
  fim;
  SCHEDULER;
fim;
```

```
primitiva ACEITARENDS(var CF,N); (* aceita rendezvous em select *)
inicio (* a entrada com possibilidade de rendezvous imediato que contiver o
          N-ésimo ETE será selecionada. CF - cabeça da FDEA da tarefa *)
  repita
    se NTER(CF)>0 então (*é possível rendezvous imediato com esta entrada*)
      inicio
        H0:=CF; H1:=EE(CF);
        N:=N-EE(CF);
      fim;
    EE(CF):=0; (* fecha todas as entradas *)
    CF:=PROXDEA(CF);
  até N≤0;
  (* a entrada cujo DE é apontado por H0 é a escolhida *)
  (* fecha as demais entradas *)
  enquanto CF≠NIL faça
  inicio
    EE(CF):=0;
    CF:=PROXDEA(CF);
  fim;
  (* desvia para o ETE escolhido da entrada *)
  CP:=ETE [H1+N](H0);
fim;
```

```
instrução ACEN RM; (* aceita chamada de entrada *)
inicio (* RM é o endereço da instrução de fim de rendezvous que faz também o
          papel do ME da entrada.
          O endereço do DE da entrada está carregado no topo da pilha *)
  H0:=(TOPO); (* End_DE *)
  TOPO:=TOPO-1;
  H1:=PFTER(H0); (* tarefa para a qual a chamada está sendo aceita *)
  se H1=NIL então
    inicio (*a tarefa que chamou a entrada recebeu exceção ou foi abortada*)
      EP:=TRUE; (* levanta exceção na tarefa corrente *)
      IE:=TASKING-ERROR;
      CP:=EME(ADCT);
    fim;
  NTER(H0):=NTER(H0)-1;
  (* inicializa o RAEN *)
  TRA(TOPO):=RAEN;
  EME(TOPO):=RM;
  ADCTA(TOPO):=ADCT;
  ATCE(TOPO):=H1;
  ARPE(TOPO):=PADCT(H1);
  (* atualiza a situação da tarefa que chamou a entrada *)
  RETIRA(H1,PFTER(H0)); FILA(H1):=RENDS;
  ATDE(H1):=ARATC; (* a mesma posição do campo PROX *)
  ARAE(H1):=ADCT; (* a mesma posição do campo ANT *)
  (* atualiza a tarefa corrente *)
  ADCT:=TOPO; TOPO:=TOPO+|RAEN|;
fim;
```

```
primitiva ATIVATF(End_RAT); (* ativa as tarefas já elaboradas inicializadas
                               pela tarefa T *)
inicio (* End_RAT é o endereço da pilha da tarefa T *)
  (* o tempo de execução desta instrução é contado como sendo de T *)
  H1:=RELOG;
  H0:=PRIO(End_RAT);
  (* ativa todas as tarefas inicializadas por T *)
  para H2:=1 até NTI(End_RAT) faça
  inicio
    H3:=(TOPO(End_RAT)); (* apontador para uma das tarefas inicializadas *)
    se H3≠NIL então (* testa se a tarefa está elaborada ou se já terminou *)
      inicio
        NTLA(H3):=0; (* número de tarefas locais ativas é igual a zero *)
        INSERETF(H3,PFTP[H1]);
        FILA(H3):=FTP;
        PRIO(H3):=H0;
        (* lista das ativações externas de  subprogramas externos começa
           vazia *)
        PLSE(H3):=NIL
      fim;
    (* retira o apontador para a tarefa já ativada do topo da pilha de T *)
    TOPO(End_RAT):=TOPO(End_RAT)-1;
  fim;
  INSERETF(End_RAT,PFTP[H1]); (* insere T na FTP correspondente *)
  FILA(End_RAT):=FTP; (* adiciona a TAET o tempo de execução desta instrução *)
  TAET(End_RAT):=TAET(End_RAT)+RELOG-H0
fim;
```

```
instrução CEEE d_DE; (* carrega endereço de descritor de entrada externa *)
inicio
  H0:=(PADCT-2); (* End_DT da tarefa onde a entrada está declarada *)
  se ET(H0)≠ATIVA então
    inicio
      EP:=TRUE;
      IE:=TASKING-ERROR;
      CP:=EME(ADCT);
    fim;
 (PADCT-1):=ARAT(H0)+d_DE; (* End_DE *)
fim;

instrução  CEEL d_DE; (* carrega endereço de descritor de entrada local *)
inicio  (* a entrada está declarada na tarefa corrente *)
  TOPO:=TOPO+1;
  (TOPO):=ARATC+d_DE;
fim;
```

```
instrução CFEE d_DFE; (* carrega endereço de membro de família de entrada
                         externa *)
inicio
  H0:=(PADCT-2); (* End_DT da tarefa onde a entrada está declarada *)
  H1:=(PADCT-1); (* índice da entrada na família *)
  se ET(H0)≠ATIVA então
    inicio (* levanta exceção na tarefa corrente *)
      EP:=TRUE;
      IE:=TASKING-ERROR;
      CP:=EME(ADCT); (* desvia para o ME da ativação corrente *)
    fim;
  (* a tarefa está ATIVA *)
  H2:=ARAT(H0)+d_DFE; (* End_DFE *)
  se (H1<LINFE(H2)) ou (H1>LSUPE(H2)) então (* índice fora dos limites *)
    inicio (* levanta exceção *)
      EP:=TRUE;
      IE:=RANGE-ERROR;
      CP:=EME(ADCT);
    fim;
  (* carrega o endereço do DE *)
  (PADCT-1):=ADEF(H2)+(H1-LINFE(H2))*|DE|;
fim;
```

```
instrução CFEL d_DFE; (* carrega endereço de membro de família de entrada
                         local *)
inicio
  H0:=(TOPO); (* índice da entrada na família *)
  H1:=ARATC+d_DFE; (* End_DFE *)
  se (H0<LINFE(H1)) ou (H0>LSUPE(H1)) então (* índice fora dos limites *)
    inicio  (* levanta exceção *)
      EP:=TRUE;
      IE:=RANGE-ERROR;
      CP:=EME(ADCT);
    fim;
  (* carrega o endereço do DE *)
  (TOPO):=ADEF(H1)+(H0-LINFE(H1))*|DE| ;
fim;
```

```
instrução CHEN; (* chamada de entrada *)
inicio (* os endereços do DT da tarefa T onde a entrada está declarada e do
          DE estão nas posições PADCT-2 e PADCT-1 respectivamente *)
  H0:=(PADCT-1); (* End_DE *)
  H1:=(PADCT-2); (* End_DT *)
  se (ET(H1)=ATIVA) e (ARATC≠ARAT(H1)) então
    inicio (* a entrada pode ser chamada *)
      (* a tarefa corrente é suspensa e colocada na fila da entrada *)
      se FILA=FTP então
        SUSPENDE(PFTP[PRIO])
      senão
        SUSPENDE(PFTE);
      INSERETF(ARATC,PFTER(H0));
      FILA:=FTER; NTER(H0):=(H0)+1;
      se EE(H0)>0 então
        inicio (* a entrada está aberta e T deve ser colocada numa FTP *)
          H2:=ARAT(H1); (* End_RAT *)
          se FILA(H2)=FTD então RETIRAFTD(H );
          INSERETF(H2,PFTP[PRIO(H2)] );
          CPCT(H2):=ETE[GERANÚMERO(EE(H0))](H0);
          (* fecha as entradas de T pois a chamada será aceita assim que T
             for executada *)
          FECHAENT(PFDEA(H2));
        fim;
      TAET:=TAET+RELOG;
      SCHEDULER;
    fim
```

```
  senão (* ou T não está ATIVA ou está chamando entrada própria *)
    inicio (* a tarefa corrente recebe exceção *)
      EP:=TRUE;
      IE:=TASKING-ERROR;
      CP:=EME(ADCT);
    fim;
fim;
```

instrução CMFT N_u,d_{DFT}; INTERROMPÍVEL (* carrega endereço de descritor de membro de família de tarefas *)

```
início (* o índice do membro da família está carregado no topo *)
  H0:=(TOPO);
  H1:=D[Nu]+dDFT; (* endereço efetivo do DFT *)
  se (H0<LINFF(H1)) ou (H0>LSUPF(H1)) então
    início (* índice fora dos limites, levanta RANGE-ERROR *)
      EP:=TRUE;  IE:=RANGE-ERROR;
      CP:=EME(ADCT); (* desvia para o ME local *)
    fim;
  (TOPO):=ADEF(H1)+H0+|DE|;
fim;
```

```
primitiva CONTANPR (CF); (* conta o número de possíveis rendezvous imediatos
                            com entradas na FDEA *)
inicio (* CF - cabeça da fila de descritores de entradas abertas *)
  H0:=0; (* número de rendezvous  possíveis *)
  repita
    (* transforma os valores de EE em valores positivos *)
    EE(CF):=-EE(CF);
    (* testa se existe tarefa esperando rendezvous com a entrada *)
    se NTER(CF)>0 então H0:=H0+EE(CF);
    CF:=PROXDEA(CF);
  até CF=NIL;
  retorna H0;
fim;


instrução CRCT K; INTERROMPÍVEL (* carrega constante *)
inicio
  TOPO:=TOPO+1;
  (TOPO):=K
fim;

instrução CREN N, d; INTERROMPÍVEL (* carrega endereço *)
inicio
  TOPO:=TOPO+1;
  (TOPO):=D[N]+d;
fim;
```

```
instrução DIME; INTERROMPÍVEL (* desvio indireto em manipulador de exceções *)
inicio
  CP:=(CP+IE);
fim;

instrução DSUB R; (* desvia para subprograma *)
início (* R é o endereço do código do subprograma *)
  ER(PADCT):=CP (* salva o endereço de retorno *)
  (* atualiza ADCT e restaura o valor de PADCT *)
  ADCT:=PADCT; PADCT:=DA(ADCT);
  CP:=R; (* desvia para o subprograma *)
fim;

instrução DSVF R; INTERROMPÍVEL (* desvia se conteudo do topo é FALSE *)
inicio
  TOPO:=TOPO-1;
  se (TOPO+1)=FALSE então CP:=R;
fim;

instrução DSVS R; INTERROMPÍVEL (* desvia sempre *)
inicio
  CP:=R;
fim;
```

```
instrução DVSE N_A,d_DT,R; (* desvia para subprograma externo *)
início (* (N_A,d_DT) - endereço textual do descritor da tarefa T.
          R - endereço do código do subprograma externo *)
  ER(PADCT):=CP; (* salva o endereço de retorno *)
  se ET(D[N_A]+d_DT) em [ATIVO, SEMI-INATIVO] então
    início
      ADCT:=PADCT; PADCT:=DA(ADCT);
      CP:=R; (* desvia para o subprograma *)
    fim
  senão (* a tarefa T não está ativa e o SE não pode ser chamado *)
    início (levanta exceção na unidade que está chamando o SE
      PADCT:=DA(APDCT);
      EP:=TRUE; IE:=TASKING-ERROR;
      CP:=EME(ADCT); (* desvia para o ME *)
    fim;
fim;


instrução ECDL; (* "executa" comando delay *)
inicio (* o valor de delay (tempo de dormência) está no TOPO da pilha *)
  HD:=(TOPO);
  TOPO:=TOPO-1;
  CPCT:=CP; (* salva endereço de transferência do delay *)
  (* insere a tarefa corrente na FTD  se o valor do delay for maior que zero *)
  INSEREFTD;
  TAET:=TAET+RELOG; (* atualiza o tempo acumulado de execução da tarefa *)
  SCHEDULER;
fim;
```

```
instrução ENSE RM,N_T,d_DT,N_DTL; (* entra em subprograma externo *)
início (* RM é o endereço do ME do subprograma externo (SE).
          N_T é o nível da tarefa onde SE está declarado.
          d_DT é a posição do DT de T relativa ao RA da unidade onde T foi
          declarado.
          N_DTL é o número de descritores de tarefas locais, não incluindo
          os descritores de membros de famílias *)
  EME(ADCT):=RM;
  NA(ADCT):=NC;
  NC:=N_T+1; (* N_SE=N_T+1 *)
  DA(ADCT):=D[NC];
  D[NC]:=ADCT;
  NTLA(ADCT):=0;
  NDTL(ADCT):=N_DTL;
  se TC(ADCT)=EXTERNA então
    início (* insere RASE na LSE da tarefa T onde SE foi declarado *)
      INSERETF(ADCT,PLSE(D[N_T]));
      (* salva D[N_T] e atualiza o contexto no nível N_T *)
      DANT(ADCT):=D[N_T];
      D[N_T]:=ARAT(D[N_T-1]+d_DT);
      (* guarda valor de ARATC em AP que será utilizado se T terminar antes
         de SE retornar *)
      AP(ADCT):=ARATC;
    fim;
fim;
```

```
primitiva EXECUTA(T,IT);
inicio
  (* dependendo da implementação faz HTIT:=RELOG + IT *)
  (* atualiza ARATC com o endereço do RA da tarefa *)
  ARATC:=T;
  (* atualiza o vetor D *)
  para H0 até nmax faça D[H0] :=DISP[H0];
  (* atualiza TAET para guardar o tempo acumulado de execução da tarefa *)
  TAET:=TAET-RELOG;
  (* desvia para a primeira instrução a ser executada da tarefa *)
  CP:=CPCT;
fim;
```

```
primitiva FECHAENT(var CF); (* fecha todas as entradas de uma dada tarefa *)
inicio (*CF-cabeça da fila dos descritores de entradas abertas de uma tarefa*)
  repita
    EE(CF):=0; (* fecha a entrada *)
    CF:=PROXDEA(CF); (* pega o próximo DE *)
  até CF=NIL;
fim;
```

```
instrução FREN; (* fim de "rendezvous" *)
inicio
  Hø:=ATCE(ADCT); (* End_RAT da tarefa que chamou a entrada *)
  (* insere a tarefa na sua fila de execução *)
  se ET(APDT(Hø))=ATIVA então
    inicio
      INSERETF(Hø,PFTP[PRIO (Hø)]); FILA(Hø):= FTP;
    fim
  senão
    inicio
      INSERETF(Hø,PFTE); FILA(Hø):=FTE;
    fim;
  (* desativa e desaloca o RAEN *)
  TOPO:=ADCT;
  ADCT:=ADCTA(ADCT);
  se EP=TRUE então
    inicio (* levanta exceção na tarefa que chamou a entrada *)
      se IE=FAILURE então
        IE(Hø):=TASKING-ERROR
      senão
        se (EP(Hø)≠TRUE) ou (IE≠FAILURE) então IE(Hø):=IE;
      EP(Hø):=TRUE; CPCT(Hø):=EME(ADCT(Hø));
      PADCT(Hø):=(PADCT(Hø)); (* atualiza PADCT para seu valor anterior *)
      (* desvia para o ME da ativação corrente *)
      CP:=EME(ADCT);
    fim;
fim;
```

```
instrução FSLT; (* fim de comando select *)
inicio
  TOPO:=ADCT;
  ADCT:=ADCTA(TOPO); (* desativa o RAST *)
  se EP=TRUE então
    inicio (* fecha as entradas e desvia para ME da ativação corrente *)
      FECHAENT(PFDEA);
      CP:=ENE(ADCT);
    fim;
fim;
```

```
instrução INIT N; (* inicializa N tarefas *)
inicio (* legenda
          Posição na pilha das informações de cada tarefa a ser inicializada
          (TOPO) - Endereço de entrada na tarefa T.
          (TOPO-1) - Endereço do descritor de T(DT) relativo à base do RA de
                     U onde T está declarada.
          (TOPO-2) - Nível de U.
          (TOPO-3) - Endereço do ME de T.
          H1 - Usado como endereço do DT.
          H2 - Usado como cabeça da fila das tarefas que serão colocadas na
               FTE no fim desta instrução.
          H3 - Endereço da pilha de T. *)
  H2:=NIL;
  para H0:=1 até N faça
  inicio
    H1:=D[TOPO-2]+(TOPO-1); (* END_DT=D[N_UT]+d_DT *)
    se ET(H1)≠INATIVA então
      inicio (* tentativa de inicializar tarefa ativa *)
        (* levanta exceção na tarefa que está executando o INIT *)
        EP(ADCT):=TRUE;
        IE(ADCT):=INITIATE-ERROR;
        (* desativa as tarefas colocadas na fila apontada por H2 desalocando
           as pilhas das tarefas com alocação dinâmica *)
        enquanto H2≠NIL faça
          inicio
            H3:=H2;
            H2:=PROX(H2); (* retira a primeira da fila *)
```

```
        ET(APDT(H3)):=INATIVA;
        se TA(APDT(H3))=DIN então DESALOCPIL(H3);
        ARAT(APDT(H3)):=NIL;
      fim;
    (* desvia para o ME na tarefa corrente *)
    CP:=EME(ADCT);
  fim;
se TA(H1)=DIN então
  inicio (* aloca uma pilha para a tarefa *)
    H3:=ALOCPIL;
    ARAT(H1):=H3;
  fim
senão H3:=ARAT(H1);
APDT(H3):=H1;
(* coloca a tarefa na FTE atualmente apontada por H2 *)
INSERETF(H3,H2);
FILA(H3):=FTE;
(* inicializa os campos do RAT *)
TRA(H3):=RAT;
EME(H3):=(TOPO-3);
IND(H3):=H0-1;
NC(H3):=(TOPO-2)+1;
CPCT(H3):=(TOPO);
ADCT(H3):=H3;
TOPO(H3):=H3+|RAT|;
EP(H3):=FALSE;
HD(H3):=0;
```

```
    TAET(H3):=0;
    TQEI(H3):=ARATC;
    AUDT(H3):=D[(TOPO-2)];
    (* inicializa o vetor DISP da tarefa T. Basta copiar os valores de D do
       nível 0 até o nível N_U e DISP_T[N_U+1] (N_U+1 é o nível de T) com o ende-
       reço da sua pilha *)
    para H4:=0 até (TOPO-2) faça DISP(H3)[H4]:=D[H4];
    DISP(H3)[(TOPO-2)+1]:=H3;
    (* desempilha as informações desta tarefa *)
    TOPO:=TOPO-4;
  fim;
  (* suspende a tarefa corrente *)
  SUSPENDE(PFTP[PRIO]);
  FILA:=SIT;
  NTE:=N; NTI:=N; (* número de tarefas por inicializar *)
  INSEREFF(H2,PFTE);
  (* guarda apontadores para as pilhas das tarefas por elaborar *)
  para H0:=1 até N faça
  inicio
    TOPO:=TOPO+1;
    (TOPO):=H2;
    H2:=PROX(H2);
  fim;
  TAET:=TAET+RELOG; (* atualiza TAET da tarefa corrente *)
  SCHEDULER;
fim;
```

```
primitiva INSEREFF(F1, var F2); (* insere uma fila F1 numa F2 *)
inicio (* a fila F1 nunca é vazia *)
  se F2≠NIL então
    inicio (* F2 não é vazia *)
      PROX(ANT(F1)):=F2; PROX(ANT(F2)):=F1;
      H0:=ANT(F1);
      ANT(F1):=ANT(F2); ANT(F2):=H0;
    fim;
  F2:=F1;
fim;
```

```
primitiva INSERETF(End_RAT, var CF); (* insere tarefa em fila *)
inicio (* End_RAT - endereço da pilha da tarefa. CF - cabeça da fila. *)
  se CF=NIL então
    inicio (* fila vazia *)
      PROX(End_RAT):=End_RAT;
      ANT(End_RAT):=End_RAT;  CF:=End_RAT;
    fim
  senão
    inicio (* a fila não está vazia *)
      PROX(End_RAT):=CF;
      ANT(End_RAT):=ANT(CF);
      PROX(ANT(CF)):=End_RAT;
      ANT(CF):=End_RAT;
    fim;
fim;
```

```
primitiva INSEREFTD; (* insere a tarefa corrente na FTD *)
inicio (* se o tempo de dormência for igual a zero a tarefa não é inserida *)
  se HD>0.0 então
    inicio
      se FILA=FTP então
        RETIRA(ARATC,PFTP[PRIO])
      senão
        RETIRA(ARATC,PFTE);
      FILA:=FTD;
      (* insere na FTD na posição correspondente ao valor de seu HD *)
      se PFTP=NIL então
        inicio
          PFTD:=ARATC; PROX:=ARATC;
          ANT:=ARATC; DESP:=RELOG+HD;
        fim
      senão
        inicio
          HØ:=PFTD;
          enquanto (HD>HD(HØ)) e (PROX(HØ)≠PFTD) faça HØ:=PROX(HØ);
          se  HD<HD(HØ) então
            inicio (* insere antes de HØ *)
              PROX:=HD(HØ); ANT:=ANT(HD(HØ));
              ANT(HD(HØ)):=ARATC; PROX(ANT):=ARATC;
              se HØ=PFTD então
                inicio (*a tarefa corrente é a 1a da FTD e DESP é atualizado*)
                  PFTD:=ARATC; DESP:=RELOG+HD;
                fim;
            fim
```

```
          senão
            início (* insere depois de H0 *)
              PROX:=PROX(H0);
              ANT:=H0;
              PROX(H0):=ARATC;
              ANT(PROX):=ARATC;
            fim;
        fim;
    fim;
fim;


instrução ISLT R; (* inicio de comando select *)
início (* R é o endereço da instrução FSLT deste select *)
  TRA(TOPO):=RAST;
  EME(TOPO):=R
  ADCTA(TOPO):=ADCT;
  HDS(TOPO):=-1.0;
  ADCT:=TOPO;
  TOPO:=TOPO+| RAST |;
fim;
```

```
instrução LVEF; (* levanta exceção FAILURE *)
inicio (* o endereço do DT da tarefa T que deve receber a exceção está no TO-
          PO da pilha *)
  H0:=(TOPO); TOPO:=TOPO-1;
  se ET(H0)=ATIVA então
    inicio
      H1:=ARAT(H0); EP(H1):=TRUE; IE(H1):=FAILURE;
      se H1=ARATC então (* está levantando FAILURE para si próprio *)
        CP:=EME(ADCT); (* desvia para ME local *)
      caso FILA(H1) seja
      FTD:  inicio (* T está dormente *)
              RETIRAFTD(H1); INSERETF(H1,PFTP[PRIO(H1)]);
              FILA(H1):=FTP; CPCT(H1):=EME(ADCT(H1));
            fim;
      FTER: inicio (* T chamou uma entrada E que ainda não foi aceita *)
              H2:=(PADCT(H1)-2); (* endereço do DE de E *)
              RETIRA(H1,PFTER(H2)); (* retira da FTER *)
              NTER(H2):=NTER(H2)-1; (* atualiza NTER *)
              INSERETF(H1,PFTP[PRIO(H1)]);
              FILA(H1):=FTP; CPCT(H1):=EME(ADCT(H1));
            fim;
      RENDS: inicio
              (* uma tarefa Q está executando uma entrada chamada por T. Bas
                 ta levantar TASKING-ERROR em Q que a instrução FREN  cuida
                 de colocar T na sua FTP *)
              H2:=ATDE(H1); (* endereço do RAT da tarefa Q *)
              EP(H2):=TRUE; IE(H2):=TASKING-ERROR;
```

```
            CPCT(H2):=EME(ARAE(H1));
          fim;
    SECE: inicio (* T está suspensa esperando chamada de entrada *)
            INSERETF(H1,PFTP[PRIO(H1)]);
            FILA(H1):=FTP; CPCT(H1):=EME(ADCT);
          fim;
    SIT:  (* T está inicializando tarefas e somente tratará a exceção após
             o término da elaboração das tarefas por ela inicializada *)
          CPCT(H1):=EME(ADCT);
    FTP:  (*basta atualizar CPCT com o endereço do ME da ativação corrente*)
          CPCT(H1):=EME(ADCT(H1));
    SAPT: (*T já está terminando e a exceção não lhe tem efeito*)
    fim; (* do comando caso *)
  fim;
fim;
```

```
instrução LVTE N_E; (* levanta exceção *)
inicio (* N_E é o número associado à exceção E *)
  EP:=TRUE;
  IE:=N_E;
  CP:=EME(ADCT);
fim;
```

```
instrução MADL R; INTERROMPÍVEL (* marca alternativa delay em select *)
início (* R é o endereço da próxima alternativa selecionável.
          O valor da expressão do delay está no topo da pilha *)
  H0:=(TOPO); (* valor do delay *)
  TOPO:=TOPO-1;  H1:=HDS(ADCT);
  se (H1=0.0) ou ((H1>0.0) e (H0<H1)) então
    início (* este delay é o de menor valor dentre os anteriores *)
      HDS(ADCT):=H0;(* HDS é atualizado com este valor *)
      (* salva em ETDL o endereço de transferência deste delay *)
      ETDL(ADCT):=CP
    fim;
  CP:=R; (* desvia para o código da próxima alternativa selecionável *)
fim;
```

```
instrução MDEN d_DE,K; INTERROMPÍVEL (* monta descritor de entrada *)
início (* d_DE é a posição do descritor da entrada relativa ao RA da tarefa
          corrente.
          K é o número máximo de endereços de transferência *)
  H0:=ARATC+d_DE; (* endereço efetivo de DE *)
  TDE(H0):=DE; NTER(H0):=0
  PFTER(H0):=NIL; NMET(H0):=K;
  EE(H0):=0;
fim;
```

```
instrução MDFE d_DFE,K; INTERROMPÍVEL (* monta descritor de família de entradas
                                         e um descritor para cada membro da fa-
                                         mília *)
início (* o DI da família está no topo da pilha. d_DFE é a posição do DFE rela-
          tiva ao RA da tarefa corrente. K é o número máximo de endereços de
          transferência *)
  H0:=ARATC+d_DFE; (* endereço efetivo de DFE *)
  LSUPE(H0):=(TOPO);  TDE(H0):=DFE;
  LINFE(H0):=(TOPO-1);
  TOPO:=TOPO-|DI|;
  ADEF(H0):=TOPO;
  para  H1:=LINFE(H0) até LSUPE(H0) faça
  início (* monta um DE para cada membro da família na PDD *)
    NTER(TOPO):=0; PFTER(TOPO):=NIL;
    NMET(TOPO):=K; EE(TOPO):=0;
    TOPO:=TOPO+|DE|;
  fim;
fim;
```

```
instrução MDTD d_DT; INTERROMPÍVEL (* monta DT com alocação dinâmica *)
início
  Ho:=ADCT+d_DT;
  TDT(Ho):=DT;
  TA(Ho):=DIN;
  ATFM(Ho):=ARATC;
  ET(Ho):=INATIVA;
fim;
```

```
instrução MDTE d_DT; INTERROMPÍVEL (* monta DT com alocação estática *)
início
  Ho:=ADCT+d_DT;
  TDT(Ho):=TD;
  TA(Ho):=EST;
  ATFM(Ho):=ARATC;
  ARAT(Ho):=ALOCPIL;
  ET(Ho):=INATIVA;
fim;
```

```
instrução MFTD d_DFT; INTERROMPÍVEL (* monta descritor de família de tarefas
          e descritores para cada membro da família com alocação dinâmica *)
inicio (* o DI da família está nas últimas posições da pilha *)
  H0:=ADCT+d_DT;
  TOPO:=TOPO-|DI|;
  TDT(H0):=DFT;
  ADTF(H0):=TOPO;
  LINFF(H0):=LINF(TOPO);
  LSUPF(H0):=LSUP(TOPO);
  (* monta um DT para cada membro da família *)
  para H1:=LINFF(H0) até LSUPF(H0) faça
  inicio
    IT(TOPO):=H1; (* índice desta tarefa na família *)
    TA(TOPO):=DIN;
    ATFM(TOPO):=ARATC;
    ET(TOPO):=INATIVA;
    TOPO:=TOPO+|DT|;
  fim;
fim;

instrução MFTE d_DFT; INTERROMPÍVEL (* monta descritor de família de tarefas
          e os descritores de tarefa para cada membro da família com aloca-
          ção estática *)
inicio (* o DI da família está nas últimas posições da pilha *)
  H0:=ADCT+d_DFT;
  TOPO:=TOPO-|DI|;
  TDT(H0):=DFT;
```

```
  LINFF(H0):=LINF(TOPO);
  LSUPF(H0):=LSUP(TOPO);
  (* monta um DT para cada membro da família *)
  para H1:=LINFF(H0) até LSUPF(H0) faça
  inicio
    IT(TOPO):=H1;
    TA(TOPO):=EST;
    ATFM(TOPO):=ARATC;
    ARAT(TOPO):=ALOCPIL;
    ET(TOPO):=INATIVA;
    TOPO:=TOPO+|DT|;
  fim;
fim;
```

```
instrução MPEN K; INTERROMPÍVEL (* marca pilha para chamada de entrada *)
inicio; (* K é o tamanho da parte estática dos parâmetros da entrada *)
  (TOPO+1):=PADCT;
  PADCT:=TOPO+1;
  TOPO:=TOPO+K+1;
fim;
```

```
instrução MPSE K,T; INTERROMPÍVEL (* marca pilha para chamada de subprograma
                                     externo *)
início (* K é o tamanho da parte estática das declarações e dos tarâmetros
          mais |RASE|.  T é o tipo da chamada LOCAL ou EXTERNA *)
  TRA(TOPO):=RASE; TC(TOPO):=T;
  (* salva PADCT para o caso desta chamada ser efetuada durante a elaboração
     dos parâmetros de outra chamada *)
  DA(TOPO):=PADCT; PADCT:=TOPO;
  TOPO:=TOPO+K;
fim;
```

```
primitiva PROPAGA1; (* levanta TASKING-ERROR nas tarefas que estão esperando
                       "rendezvous" com entradas da tarefa corrente *)
inicio
  H0:=ADET; (* apontador para a região de descritores de entradas da tarefa
               corrente *)
  para H1:=1 até NDET faça
  se TDE(H0)=DE então
    inicio (* descritor de entrada simples *)
      PROPAGA2(H0);
      H0:=H0+|DE|;
    fim
  senão
    inicio (* descritor de família de entradas *)
      PROPAGA3(H0);
      H0:=H0+|DFE|;
    fim;
fim;

primitiva PROPAGA2(End_DE); (* levanta TASKING-ERROR para as tarefas na FTER
                               desta entrada *)
inicio (* End_DE é o endereço do DE da entrada *)
  H0:=PFTER(End_DE);
  enquanto H0≠NIL faça
  inicio (* levanta a exceção, retira da FTER e insere numa FTP *)
    EP(H0):=TRUE;
    IE(H0):=TASKING-ERROR;
    CPCT(H0):=EME(ADCT(H0));
```

```
    PFTER(End_DE):=PROX(H0);
    INSERETF(H0,PFTP[PRIO(H0)]);
    FILA(H0):=FTP;
    H0:=PFTER(End_DE);
  fim;
fim;

primitiva PROPAGA3(End_DFE); (* chama PROPAGA3 para cada entrada da família *)
início
  H0:=ADEF(End_DFE);
  para H1:=LINFE(End_DFE) até LSUPE(End_DFE) faça
  início
    PROPAGA2(H0);
    H0:=H0+|DE|;
  fim;
fim;

primitiva PROPAGA4; (* levanta TASKING-ERROR nas tarefas que estão executando
                       SEs da tarefa corrente *)
início
  H0:=PLSE;
  enquanto H0≠ NIL faça
  início
    H1:=AP(H0);
    PLSE:=PROXLSE(H0);
    DESATIVA(H1,H0); (* termina todas as ativações em H1 até H0 inclusive *)
```

```
    (* propaga a exceção *)
    EP(H1):=TRUE;
    IE(H1):=TASKING-ERROR;
    CPCT(H1):=EME(ADCT(H1));
    H0:=PLSE;
  fim;
fim;
```

```
primitiva RETIRA(End_RAT, var CF); (* retira tarefa de uma fila *)
inicio (* End_RAT -endereço da pilha de tarefa
          CF - cabeça da pilha *)
  se PROX(End_RAT)=End_RAT então (* T é a única tarefa na fila *)
    CF:=NIL
  senão
    inicio
      PROX(ANT(End_RAT)):=PROX(End_RAT);
      ANT(PROX(End_RAT)):=ANT(End_RAT);
      se CF=End_RAT então  (* T era a primeira da fila *)
        CF:=PROX(End_RAT);
    fim;
fim;
```

```
primitiva RETIRAFTD(End_RAT); (* retira uma tarefa da fila de tarefas
                                 dormentes *)
inicio
  RETIRA(End_RAT,PFTD); HD(End_RAT):=0;
  se PFTD=NIL então (* a FTD ficou vazia *)
    DESP:=0
  senão (* atualiza o registrador DESP com o HD da primeira tarefa da FTD *)
    DESP:=HD(PFTD);
fim;

instrução RLVE; INTERROMPÍVEL (* relevanta exceção *)
inicio
  EP:=TRUE;
fim;

instrução RPFR; INTERROMPÍVEL (* restaura a pilha após fim de rendezvous *)
inicio
  TOPO:=PADCT-3; PADCT:=(PADCT);
fim;

instrução RSRV K; INTERROMPÍVEL (* reserva K posições de memória *)
início
  TOPO:=TOPO+K;
fim;
```

```
instrução RTSE; (* retorno de subprograma externo *)
início
  se TC(ADCT)=EXT então
    início (* a chamada foi externa *)
      (* D[NC-1] aponta para o RAT onde o SE foi declarado *)
      RETIRA(ADCT,PLSE(D[NC-1]));
      (* restaura o contexto do nível onde SE foi declarado *)
      D[NC-1]:=DANT(ADCT);
    fim;
  TOPO:=ADCT;
  D[NC]:=DA(ADCT);
  NC:=NA(ADCT);
  ADCT:=D[NC];
  se EP=TRUE então
    CP:=EME(ADCT) (* a exceção é propagada para a unidade que chamou SE *)
  senão
    CP:=ER(TOPO); (* retorno normal *)
fim;

primitiva SALVACTF; (* salva o contexto da tarefa corrente *)
início
  se FILA=FTP então
    SUSPENDE(PFTP PRIO )
  senão
    SUSPENDE(PFTE);
  TAET.*TAET+RELOG;
fim;
```

```
primitiva  SCHEDULER;
inicio
  se PFTE≠NIL então
     inicio
       (* coloca a tarefa apontada por PFTE no fim da FTE *)
       PFTE:=PROX(PFTE);
       (* dependendo da implementação determina um intervalo de tempo H0 *)
       EXECUTA (ANT(PFTE),H0)
     fim
  senão
     inicio (* determina a maior prioridade Hi cuja FTP é não vazia*)
       (* coloca a tarefa apontada por PFTP[Hi] no fim da FTP[Hi] *)
       PFTP[Hi]:=PROX(PFTP[Hi]);
       (* dependendo da implementação determina um intervalo de tempo H0 *)
       EXECUTA (ANT(PFTP[Hi]),H0)
     fim
fim;
```

```
instrução SELD; (* seleciona entrada ou possível delay *)
inicio
 (* os descritores das entradas abertas estão na FDEA e com EE<0.
    Os valores de EE são atualizados para -EE por CONTANPR *)
 se (PFDEA=NIL) então (* nenhuma entrada está aberta *)
   inicio (* levanta SELECT-ERROR *)
     EP:=TRUE; IE:=SELECT-ERROR;
     CP:=EME(ADCT); (* desvia para o ME local *)
   fim;
 (* número de entradas abertas com possibilidade de rendezvous imediato *)
 H0:=CONTANPR(PFDEA);
 se H0=0 então (* nenhum "rendezvous" é possível no momento *)
  se HDS(ADCT)>0 então (* "executa" o delay *)
    inicio (* a tarefa deve ser inserida na FTD *)
      HD:=HDS(ADCT);  CPCT:=ETDL(ADCT);
      INSEREFTD;
      (* salva o seu contexto e atualiza o tempo acumulado de execução *)
      para H0:=0 até nmax faça DISP[H0]:=D[H0];
      TAET:=TAET+RELOG;
      SCHEDULER;
    fim
  senão (*o select não tem alternativa com delay e a tarefa é suspensa*)
    inicio
      SUSPENDE(PFTP[PRIO]);
      FILA:=SECE; (* suspensa esperando chamada de entrada *)
      TAET:=TAET+RELOG; (* atualiza o tempo acumulado de execução *)
```

```
        SCHEDULER;
      fim;
  (* H0 "rendezvous" são possíveis. O H1-ésimo ETE é escolhido *)
  H1:=GERANÚMERO(H0); (* gera um número entre 1 e H0 inclusive *)
  (* aceita "rendezvous" para a entrada que contiver o H1-ésimo endereço de
     transferência na FDEA *)
  ACEITARENDS(PFDEA,H1);
fim;
```

```
instrução SELE R; (* seleciona entrada em select com cláusula else  *)
inicio (* R-endereço do código para a cláusula else *)
  (* os descritores das entradas abertas estão na FDEA e com EE 0.
     Os valores de EE são atualizados para -EE por CONTANPR *)
  se PFDEA=NIL então (* nenhuma alternativa está aberta *)
    CP:=R; (* desvia para o else *)
  (* número de entradas abertas com possibilidade de rendezvous imediato *)
  H :=CONTANPR(PFDEA);
  se H0=0 então (* nenhum rendezvous é possível no momento *)
    inicio (* o else é executado *)
      FECHAENT(PFDEA);
      CP:=R; (* desvia para o else *)
    fim;
  (* H0≠0 rendezvous são possíveis *)
  (* o H1-ésimo endereço de transferência é escolhido *)
  H1:=GERANÚMERO(H0);
  (* aceita rendezvous para a entrada que contiver o H1-ésimo endereço de
     transferência na FDEA *)
  ACEITARENDS(PFDEA,H1);
fim;
```

```
primitiva SUSPENDE(var CF); (* retira a tarefa corrente da fila F e salva seu
                               contexto *)
inicio
  RETIRA(ARATC,CF);
  CPCT:=CP;
  (* salva o vetor D em DISP *)
  para H0:=0 até nmax faça DISP[H0]:=D[H0];
fim;
```

```
instrução TNEL d_DET,N_DET,N_DTL; (* término de elaboração de tarefa *)
inicio (* d_DET - endereço de inicio da região dos descritores de entrada de T
                  relativo à base do RA de T.
          N_DET - número de descritores de entrada de T. Inclui os descritores
                  de famílias mas não os descritores dos membros da família.
          N_DTL - número de descritores de tarefas locais de T. Não inclui os
                  descritores de membros de famílias de tarefas. *)
  SUSPENDE(PFTE); (* suspende a tarefa corrente que está na FTE *)
  (* inicializa certos ítens do RAT *)
  PADCT:=NIL;
  se d_DET=0 então
    ADET:=NIL (* T não tem entrada *)
  senão (* ADET:=endereço efetivo da região dos descritores de entradas *)
    ADET:=ARATC+d_DET;
  FILA:=SEDA; (* suspensa já elaborada *)
  PFDEA:=NIL; (* todas as entradas estão fechadas *)
  NDET:=N_DET;
  NDTL:=N_DTL;
  (* atualiza TAET que vale TAET'(anterior)-RELOG'(do inicio desta execução),
     para TAET+RELOG(atual) que resulta em TAET'+RELOG-RELOG' *)
  TAET:=TAET+RELOG;
  NTE(TQEI):=NTE(TQEI)-1;
  se NTE(TQEI)=0 então ATIVATF(TQEI);
  SCHEDULER;
fim;
```

```
instrução TNT1; (* término de tarefa *)
inicio (* a tarefa pode estar ATIVA ou SEMI-ATIVA *)
  se ET(APDT)=SEMI-ATIVA então
    inicio (* a tarefa corrente é retirada da FTE e o apontador que TQEI (ta-
              refa que executou o initiate) tem para ela passa a valer NIL *)
      RETIRA(ARATC,PFTE);
      (TOPO(TQEI)-IND):=NIL;
      (* desativa a tarefa e se for o caso desaloca sua pilha *)
      ET(APDT):=INATIVA;
      H0:=TQEI; (* salva TQEI caso a pilha da tarefa seja desalocada *)
      se TA(APDT)=DIN então DESALOCPIL(ARATC);
      (* decrementa NTE da TQEI, se NTE ficar igual a zero então TQEI e as ta
         refas por ela inicializadas devem ser ativadas *)
      NTE(H0):=NTE(H0)-1;
      se NTE(H0)=0 então ATIVATF(H0);
      (* chama SCHEDULER para executar outra tarefa *)
      SCHEDULER;
    fim
  senão (* a tarefa está ATIVA e em uma FTP *)
    inicio
      SUSPENDE(PFTP[PRIO]);
      (* manda TASKING-ERROR para as unidades que chamaram entradas da tarefa
         corrente mas que ainda não foram atendidas *)
      PROPAGA1;
      se NTLA≠0 então
```

```
        inicio (* a tarefa tem tarefas locais ativas, ela ficará no estado
                  SEMI-INATIVA até que estas tarefas terminem, quando exe-
                  cutará TNT2 terminando *)
          ET(APDT):=SEMI-INATIVA;
          FILA:=SAPT;
          SCHEDULER;
        fim;
    fim;
fim;
```

```
instrução TNT2; (* término de tarefa *)
inicio (* manda TASKING-ERROR para as unidades que estão executando subpro-
          gramas externos desta tarefa *)
  PROPAGA4;
  (* atualiza a situação da unidade de declaração da tarefa corrente *)
  NTLA(AUDT):=NTLA(AUDT)-1;
  se (NTLA(AUDT)=0) e (FILA(ATFM)=SAPT) então
     inicio (* a tarefa "mãe" da tarefa corrente está suspensa e deve ser co
               locada na sua FTP para terminar a ativação corrente que está
               pronta para terminar *)
       INSERETF(ATFM,PFTP[PRIO(ATFM)]);
       FILA(ATFM):=FTP;
     fim;
  (* desativa e se for o caso desaloca sua pilha *)
  ET(APDT):=INATIVA;
  se TA(APDT)=DIN então DESALOCPIL(ARATC);
  (* chama o scheduler para executar outra tarefa *)
  SCHEDULER;
fim;
```

```
instrução TNTA; (* testa NTLA para ver se pode terminar *)
inicio
  se NTLA(ADCT)≠0 então
    inicio (* suspende a tarefa corrente que ficará com ativação
              pronta para terminar *)
      SUSPENDE(PFTP[PRIO]);
      FILA:=SAPT;
      SCHEDULER;
    fim;
fim;

instrução TRTE; INTERROMPÍVEL (* trata exceção *)
inicio
  EP:=FALSE;
fim;
```

APÊNDICE II

DICIONÁRIO DE ABREVIAÇÕES

ABREVIAÇÕES	DESCRIÇÃO	página
ABRE	instrução	999
ABRS	instrução	100
ABRT	instrução	101
ACEITARENDS	primitiva	102
ACEN	instrução	103
ADCT	Apontador Dinâmico do Contexto da Tarefa	44
ADCTA	Valor de ADCT anterior à aceitação da Chamada de uma Entrada	68
ADEF	Apontador para o primeiro descritor de entradas da família de entradas	66
ADET	Apontador para a região dos descritores de entrada da tarefa	45
ADTF	Apontador para os descritores de tarefas de uma fa - mília de tarefas	41
ALOCMVD	primitiva - Aloca posições na Memória de Variáveis Dinâmicas - Dependente da Implementação	19
ALOCPIL	primitiva - Aloca uma pilha na Memória de Pilhas - Dependente de Implementação	18
ANT	Aponta para tarefa anterior na mesma fila	44
ANTLSE	Registro de ativação anterior na Lista de chamadas externas de Subprogramas Externos	85
AP	Apontador para a base do registro de ativação no qual o subprograma externo está sendo executado	86
APDT	Apontador para o próprio Descritor de Tarefa	46
ARAC	Registrador da MEADA - aponta para o registro de a tivação corrente	16
ARAE	Apontador para o registro de ativação da entrada que foi chamada pela tarefa cujo RAT contém o ARAE	44
ARAT	Apontador para o registro de ativação da tarefa	42
ARATC	Registrador da MEADA - aponta para o RAT corrente	16
ARPE	Apontador para a região onde estão alocados os pa- râmetros da entrada	68

FTER	Fila de Tarefas Esperando Rendezvous com uma dada entrada	40
FTP	Fila de Tarefas Prontas segundo sua prioridade	14
GERANUMERO	primitiva - Retorna um número aleatório - Dependente da implementação	
HD	Hora do despertar de uma tarefa	45
HDS	Hora do despertar em *select*	80
HTIT	Registrador da MEADA - Hora do término de intervalo de tempo de execução de tarefa	16
IE	Identificação da exceção pendente	44
IND	Indice da tarefa - utilizado quando a tarefa está sendo inicializada	44
INIT	instrução	117
INSEREFF	primitiva	120
INSEREFTD	primitiva	121
INSERETF	primitiva	120
ISLT	instrução	122
IT	Indice da tarefa na família de tarefas	42
LINF	Limite inferior de um intervalo	27
LINFE	Limite inferior do índice da família de entradas	66
LINFF	Limite inferior do índice da família de tarefas	41
LSE	Lista de ativações externas de subprogramas externos de uma tarefa	85
LSUP	Limite superior de um intervalo	27
LSUPE	Limite superior do índice da família de entradas	66
LSUPF	Limite superior do índice da família de tarefas	41
LVEF	instrução	123
LVTE	instrução	124
MADL	instrução	125
MDEN	instrução	125
MDFE	instrução	126
MDTD	instrução	127
MDTE	instrução	127
ME	Manipulador de exceções	32

MFTD	instrução	128
MFTE	instrução	128
MP	Memória de Programas da MEADA	17
MPEN	instrução	129
MPIL	Memória de Pilhas da MEADA	17
MPSE	instrução	130
MRE	Memória de registradores especiais	13
MVD	Memória de Variáveis Dinâmicas da MEADA	18
NA	Nível da ativação anterior	85
NC	Nível da ativação corrente	44
NDET	Número de descritores de entradas locais da tarefa	45
NDTL	Número de descritores de tarefas locais	43
nmax	Número máximo de níveis léxicos - Determinado pela implementação	14
NMET	Número máximo de endereços de transferência de uma entrada	64
NTE	Número de tarefas em elaboração	44
NTER	Número de tarefas esperando rendezvous com uma entrada	66
NTI	Número de tarefas por inicializar num mesmo comando *initiate*	44
NTLA	Número de tarefas locais ativas	43
PADCT	Próximo valor de ADCT	43
PADCTA	Valor anterior de PADCT	75
PDD	Parte dinâmica das declarações de uma unidade	24
PDP	Parte dinâmica dos parâmetros de um subprograma ou entrada	24
PED	Parte estática das declarações de uma unidade	24
PEP	Parte estática dos parâmetros de um subprograma ou entrada	24
PFDEA	Primeiro na fila de descritores de entradas abertas	45
PFTD	Primeira tarefa na fila de tarefas dormentes	14
PFTE	Primeira tarefa da fila das tarefas em elaboração	14
PFTER	Primeira na fila das tarefas esperando rendezvous com determinada entrada	66

REFERÊNCIAS

[IC79a] - ICBIAH, J.D. et al. Preliminary ADA Reference Manual. ACM SIGPLAN Notices 14, 6 (July, 1979), part A.

[IC79b] - ICBIAH, J.D. et al. Rationale for the Design of the ADA Programming Language. ACM SIGPLAN Notices 14, 6 (July, 1979) / part B

[KO79] - KOWALTOWSKI, T. Implementação de Linguagens de Programação . São Paulo: Escola de Computação, Instituo de Matemática e Estatística, Universidade de São Paulo, 1979

[NO74] - NORI et al. The PASCAL 'P' Compiler : Implementation. (Relatório Técnico nº 10). Zürich: Institut fuer Informatik ETH,1974.

[OR73] - ORGANICK; E.I. Computer System Organization: The B5700, B6700 series. New York: Academic Press, 1973

[PA73] - PASKO, H.J. A Pseudo-Machine for Code Generation. (Dissertação de Mestrado). Toronto : Department of Computer Science , University of Toronto, 1973

[PR75] - PRATT, T.W. Programming Languages : Design and Implementation New York : Prentice-Hall, 1975.

[RA64] - RANDELL, B. & RUSSEL, L.J. ALGOL 60 Implementation. New York: Academic Press, 1964

[SA79] SANCHES, M.M. Portabilidade de Compiladores. (Dissertação de/ Mestrado). São Paulo: Instituto de Matemática e Estatística , Universidade de São Paulo, 1979

[TA80] - TAI, K & GARRARD K. Issues on the Implementation of ADA Tasking Facilities. (Comunicação Privada), Maio, 1979.

[WI75] - WIRTH, N. PASCAL - S : A Subset and its Implementation. (Relatório Técnico nº 12). Zürich : Institut fuer Informatik, ETH 1975.