AMBIENTE PARA PROGRAMAÇÃO PARALELA NO PROCESSADOR i860

Cristiana Bentes Seidel

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

Aprovada por:

Prof. Édil Severiano Tavares Fernandes, Ph. D.
(presidente)

Prof. Valmir Carneiro Barbosa, Ph. D.

Prof. Cláudio Luís Amorim, Ph. D.

Prof. Ageu Cavalcante Pacheco Junior, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
JULHO DE 1991

SEIDEL, CRISTIANA BENTES

Ambiente para Programação Paralela no Processador i860 [Rio de Janeiro] 1991

XI, 117 p., 29.7 cm, (COPPE/UFRJ, M. Sc., ENGENHARIA DE SISTEMAS E COMPUTAÇÃO, 1991)

TESE – Universidade Federal do Rio de Janeiro, COPPE

1 – Sistemas Operacionais 2 – Processamento Paralelo

I. COPPE/UFRJ II. Título(Série).

A Paulo e a meus pais

Agradecimentos

Ao professor Edil S. T Fernandes pelo interesse e dedicação com que orientou este trabalho.

Ao professor Valmir Carneiro Barbosa pela contribuição que deu ao trabalho na ausência do professor Edil.

Ao grupo de processamento paralelo da COPPE/UFRJ, Ricardo Citro, Eliseu, Alberto, José Queiroz, Juliana, Júlio, Delfim, Emerson, Edu e Adilson pela construção da máquina e pelo auxílio constante na utilização da mesma.

A Gilberto Yamashiro pelo desenvolvimento do montador e pela ajuda na utilização do i860.

A meus pais Vera e José Flávio a quem devo esta realização, pelo carinho e pelo apoio em todos os momentos da minha vida.

E a meu marido Paulo um agradecimento muito especial para quem compartilhou dos bons e dos maus momentos, transmitindo todo amor, amizade e compreensão de que necessitei.

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

Ambiente para Programação Paralela no Processador i860

Cristiana Bentes Seidel

Julho de 1991

Orientador: Edil Severiano Tavares Fernandes
Programa: Engenharia de Sistemas e Computação

Essa tese descreve o desenvolvimento do Núcleo de um Sistema Operacional para o processador i860 da Intel. O Núcleo tem como objetivo principal tornar homogêneo o ambiente para programação paralela de cada nó de processamento de uma arquitetura com topologia hipercúbica. Essa máquina, desenvolvida na COPPE-UFRJ, inclui em cada um de seus nós um processador T-800 da Inmos e um i860. O ambiente de programação paralela desenvolvido reproduz no i860 as facilidades para processamento paralelo implementadas diretamente pelo nível de μ-código do T-800.

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

Environment for Parallel Programming on i860 Processor

Cristiana Bentes Seidel

July, 1991

Thesis Supervisor: Edil Severiano Tavares Fernandes

Department: Programa de Engenharia de Sistemas e Computação

This thesis describes the development of an Operating System's Kernel for the Intel i860 processor. The main goal of the Kernal is to create a homogeneous environment suitable for parallel programming in the processing nodes of a hypercube machine wich was developed at COPPE-UFRJ. Each processing node of this hypercube includes an Inmos T -800 and an Intel i860 processors. The environment developed in the i860 provides the same facilities as those realized by the T-800's μ-code.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

A necessidade de resolver classes de problemas num curto intervalo de tempo constitui um grande desafio para a Ciência da Computação. A medida que aumenta a complexidade dos programas de aplicação, precisamos de máquinas mais rápidas. Avanços tecnológicos proporcionaram um aumento expressivo no desempenho dos computadores, entretanto, a tecnologia esbarra em um obstáculo: a velocidade da luz. Como seria possível, dada esta limitação, reduzir ainda mais o tempo de execução dos programas de aplicação? A resposta a essa pergunta está no processamento paralelo.

O processamento paralelo explora a seguinte estratégia: se não podemos aumentar a desempenho individual de um componente do sistema, o desempenho do sistema como um todo pode ser melhorado substancialmente se utilizarmos diversos componentes operando em paralelo.

Em sistemas de computação, o processamento paralelo pode estar presente em diversos níveis. Por exemplo, é possível projetar um processador com diversas unidades funcionais operando em paralelo. Em um nível mais alto, podemos ter uma arquitetura constituída de diversos processadores interconectados que podem executar, simultaneamente, diferentes partes de uma mesma tarefa. Este tipo de arquitetura é denominado de **multiprocessador**.

Multiprocessadores podem ser diferenciados, primariamente, pela forma de conexão entre os elementos de processamento e a memória, sendo classificados

como forte ou fracamente acoplados.

Máquinas fortemente acopladas são caracterizadas pela existência de uma memória global, ou seja, a memória é compartilhada por todos os elementos de processamento. Nessas máquinas, a comunicação entre os elementos de processamento pode ser levada a cabo através de primitivas do tipo **P** e **V**.

Nas máquinas fracamente acopladas a memória encontra-se distribuída pelos diversos elementos de processamento, cada elemento possui o seu segmento de memória privativo e troca informações com os outros elementos através de mensagens. A conexão dos elementos de processamento é realizada por ligações (*links*) dedicadas, por barramentos globais ou por outras estruturas formando uma rede de interconexão.

Dentre as diversas topologias de rede de interconexão podemos destacar a topologia hipercúbica [9,14,15] que é capaz de mapear inúmeras estruturas como: grade, anel e árvore.

Essa forma de interconexão é caracterizada pela presença de $N = 2^d$ elementos de processamento, onde cada elemento é conectado a outros d elementos adjacentes. Os elementos de processamento são vistos como nós, ou vértices, de um hipercubo em um espaço d-dimensional, a conexão dos nós, ou *links*, são as arestas do cubo. A Figura I.1 ilustra algumas topologias hipercúbicas, onde os círculos representam os elementos de processamento e as arestas os *links*; d representa a dimensão do hipercubo.

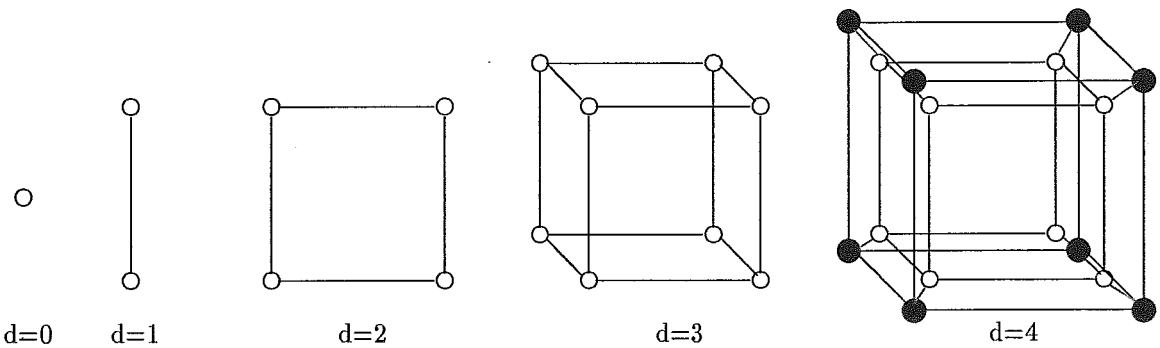

Figura I.1: Definição recursiva da topologia hipercúbica

Como parte do projeto de pesquisa em processamento paralelo, foi desenvolvida, pelo grupo de processamento paralelo da COPPE/UFRJ, uma máquina paralela, denominada de NCP-1. É um multiprocessador do tipo MIMD [11] que incorpora diversos nós de processamento interconectados segundo uma topologia hipercúbica (sendo possível a reconfiguração para outras topologias). A Figura I.2 mostra a topologia do protótipo da máquina que está em funcionamento.

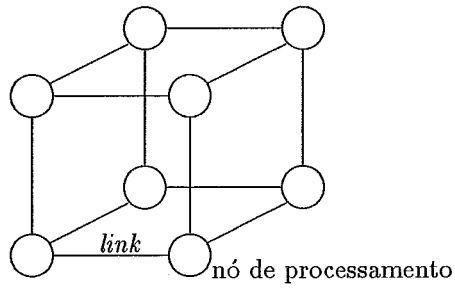

Figura I.2: Topologia da máquina hospedeira

A memória do sistema encontra-se distribuída através da rede e a comunicação entre os nós se dá diretamente através dos *links* que formam a rede de interconexão.

Cada nó de processamento pode ser considerado como um computador. Ele é constituído de dois processadores: um processador i860 da Intel [19,20] e um processador T-800 da família Transputer da Inmos [35,36]. A Figura I.3 mostra o diagrama do nó de processamento.

Conforme ilustrado na Figura I.3, existem três segmentos de memória: um segmento de memória dinâmica local do T-800, um segmento de memória estática compartilhado pelos dois processadores e um segmento de memória dinâmica em que uma parte é compartilhada pelos dois processadores e a outra é memória local do i860. A comunicação entre os processadores se dá através dos dois segmentos de memória compartilhada; a memória estática (de acesso mais rápido) pode ser utilizada para agilizar a troca de mensagens.

As arestas do hipercubo correspondem aos *links* do *Cross-Bar-Switch*. É através desse módulo que podemos configurar a rede de interconexão dos nós de processamento. As mensagens procedentes de outros nós são passadas do *Cross-*

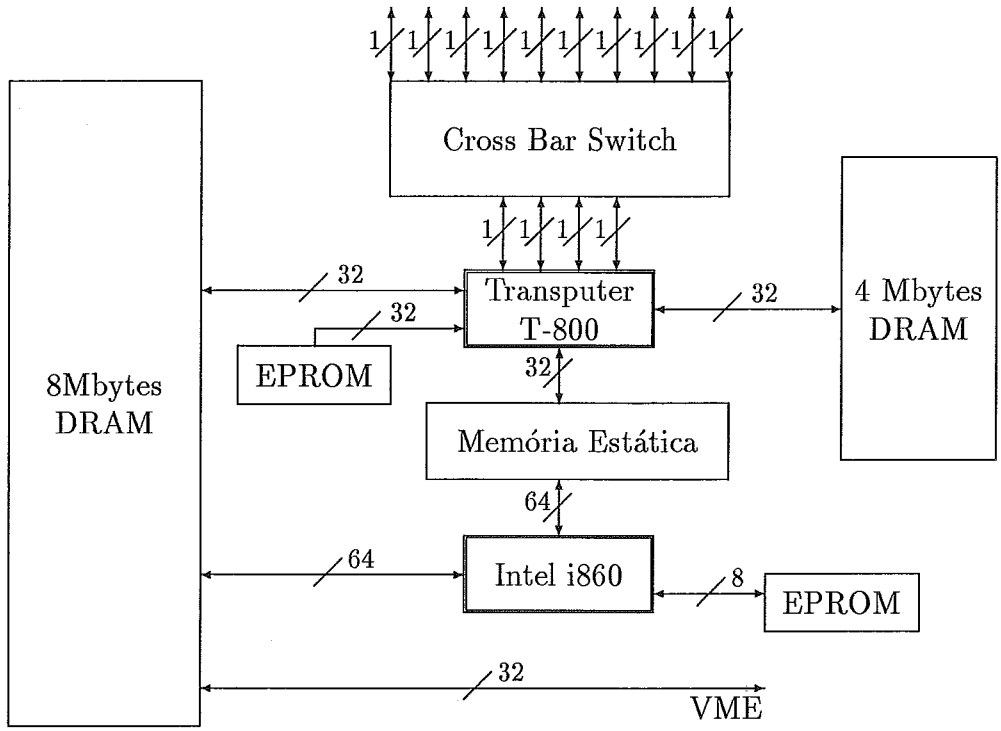

Figura I.3: Nó de processamento da máquina hospedeira

Bar-Switch para o Transputer através de um dos quatro *links* desse processador.

Além dessa rede de interconexão (controlada diretamente pelos processadores T-800), a arquitetura possui uma outra via alternativa conectando todos os nós de processamento. Essa via de comunicação é um barramento VME interconectando os processadores i860 do sistema.

Nessa estrutura, cada nó de processamento pode executar processos provenientes de um mesmo programa ou não. Por exemplo, partes de um mesmo programa podem estar sendo executadas por uma coleção de processadores T-800 e i860.

Como geralmente ocorre com as máquinas paralelas, um dos maiores problemas para explorar adequadamente os recursos do NCP-1 reside na dificuldade de programá-lo [12]. A tarefa de desenvolvimento de programas (explorando o poder computacional da máquina subjacente) será bastante facilitada se dispusermos de um ambiente adequado à programação paralela.

O objetivo fundamental desse trabalho é o de tornar homogêneo o

ambiente de programação paralela em cada um dos nós do hipercubo. Para tal, especificamos e implementamos o Núcleo de um Sistema Operacional que executa no processador i860 e provê, além das primitivas para comunicação e sincronização de processos em execução nos diversos processadores do sistema, procedimentos para reproduzir no processador i860 as facilidades para processamento paralelo que são diretamente realizadas pelo nível de μ-código do T-800. Assim, fica facilitada a tarefa de migração de processos entre os dois tipos de processadores, bem como a execução de processos cooperantes.

Essa tese está organizada em seis capítulos. No Capítulo II é apresentado o processador i860 da Intel no qual o Núcleo do Sistema Operacional irá executar.

O Capítulo III descreve o ambiente que o Núcleo deverá reproduzir. Serão vistas as principais características da linguagem Occam e como elas são implementadas no processador T-800 da Inmos.

No Capítulo IV descrevemos a estrutura geral do Núcleo, destacando as facilidades que ele oferece ao usuário, como é realizado o escalonamento de processos e como o Núcleo pode ser ativado.

O Capítulo V está diretamente ligado à implementação do Núcleo no processador i860, nele é encontrada a estrutura de dados utilizada, e todas as primitivas que compõem o Núcleo. Para cada primitiva apresenta-se o respectivo algoritmo. No final deste Capítulo são expostas as medidas de desempenho das primitivas.

No Capítulo VI apresentamos nossas conclusões do trabalho e mostramos algumas possíveis evoluções futuras.

Capítulo II

O Microprocessador i860

O microprocessador i860 da Intel [19,20] é um processador RISC de 64 bits, com arquitetura paralela utilizando técnicas de *pipeline*. Sua arquitetura inclue em um único circuito integrado operações com inteiros, operações com ponto-flutuante, operações gráficas, suporte para gerenciamento de memória e duas memórias *caches*, uma para dados e outra para instruções. A incorporação de todas as unidades funcionais em um único *chip* reduz o *overhead* de comunicação entre *chips*.

II.1 Arquitetura

A arquitetura do i860 está ilustrada na Figura II.1. A CPU i860 contém as seguintes unidades funcionais:

. Unidade central para operações em inteiros

. Unidade de gerenciamento de memória

. Unidade de controle de ponto-flutuante

. Unidade de soma de ponto-flutuante

. Unidade de multiplicação de ponto-flutuante

. Unidade gráfica

. Memória *cache* de instruções

. Memória *cache* de dados

. Unidade de controle do barramento e das memórias *caches*

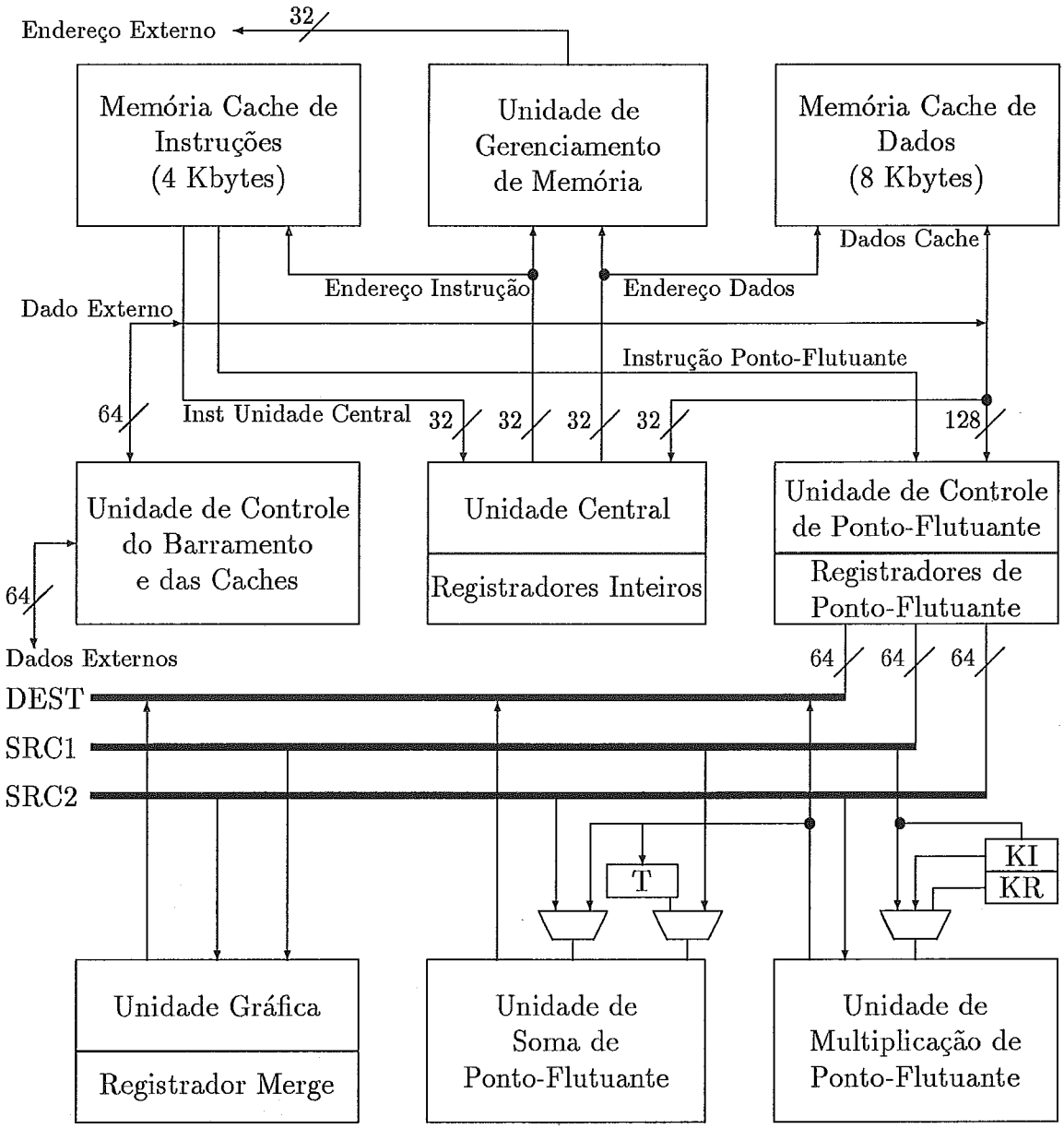

Figura II.1: O processador i860

II.2 Tipos de dados

O microprocessador i860 provê operações atuando sobre dados inteiros e em ponto-flutuante. Basicamente, operações com inteiros empregam operandos de 32 bits,

embora existam algumas operações que manipulam operandos de 64 bits. As instruções do tipo *load* e *store* podem referenciar operandos de 8, 16, 32, 64 ou 128 bits. Operações com ponto-flutuante manipulam números de 32 ou 64 bits. Instruções gráficas manipulam *arrays* de *pixels* constituídos de 8, 16 ou 32 bits.

Um inteiro é representado na forma de complemento a dois e os números reais podem estar representados em precisão simples ou em precisão dupla, de acordo com o padrão IEEE 754.

II.3 Memória Local

Conforme mostra a Figura II.2, o microprocessador i860 possui os seguintes registradores:

a. Um "arquivo" de registradores inteiros : são trinta e dois registradores de 32 bits (r0 a r31) utilizados para cálculo de endereços e em operações escalares. O registrador **r0** retorna zero quando lido, independentemente do que for armazenado nele;

b. Um "arquivo" de registradores de ponto-flutuante : são trinta e dois registradores de 32 bits, denominados de **f0** a **f31** e utilizados nas operações em ponto-flutuante. Estes registradores podem ser operados individualmente como trinta e dois registradores de 32 bits, como dezesseis registradores de 64 bits, ou como oito registradores de 128 bits. Os registradores **f0** e **f1** retornam zero quando lidos;

c. Seis registradores de controle (PSR, EPSR, DB, FIR, DIRBASE e FSR): estes registradores são acessíveis somente por instruções do tipo *load* e *store*. O registrador PSR(*Processor Status Register*) contém informações referentes ao estado do processo corrente. O registrador EPSR(*Extended Processor Status Register*) contém informações adicionais do estado do processo corrente, além de outras informações, como por exemplo, tipo do processador e tamanho da memória *cache*. O registrador DB(*Data Breakpoint*) é usado na geração de

um *trap* quando o processador acessa um operando cujo endereço está armazenado nesse registrador, permitindo a implementação de facilidades para a monitoração/depuração de programas. O registrador FIR(*Fault Instruction Register*) armazena o endereço da instrução que causou o *trap*. O registrador DIRBASE(*DIRectory BASE*) contém informações de controle para a *cache*, para a tradução de endereços e para as opções do barramento. Finalmente o registrador FSR(*Floating-point Status Register*) contém informações referentes ao estado do processo corrente quando ocorre um *trap* por erro em uma operação de ponto-flutuante e sobre os modos de arredondamento;

d. Quatro registradores de propósito especial (KR, KI, T e Merge). Os registradores KR, KI e T são utilizados por operações duais, conforme será visto posteriormente. O registrador Merge é utilizado pela unidade gráfica.

II.4 Endereçamento

A memória é acessada em unidades de bytes com um espaço de memória virtual paginado de 2^{32} bytes. Instruções e dados podem localizar-se em qualquer posição do espaço de endereçamento desde que respeitado o alinhamento. Dados só podem ser acessados em endereços múltiplos de seu tamanho. Por exemplo, o endereço de um dado de dois bytes deve ser divisível por dois, o de um dado de quatro bytes deve ser divisível por quatro e assim por diante.

Normalmente, dados com mais de um byte são armazenados no formato *little endian*, isto é, o byte menos significativo localizado no endereço de memória mais baixo. Opcionalmente, pode-se selecionar dinamicamente, por *software*, o formato *big endian*, onde o byte mais significativo encontra-se no endereço de memória mais baixo. O acesso ao código é feito sempre por endereçamento *little endian*.

Registradores Inteiros

31 0

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31

Registradores de Ponto-Flutuante

63 32 31 0

F1	F0
F3	F2
F5	F4
F7	F6
F9	F8
F11	F10
F13	F12
F15	F14
F17	F16
F19	F18
F21	F20
F23	F22
F25	F24
F27	F26
F29	F28
F31	F30

Registradores de Propósito Especial

KR	
KI	
T	
Merge	

Registradores de Controle

FSR
DB
DIRBASE
EPSR
PSR
FIR

Figura II.2: Conjunto de registradores do processador i860

II.5 Unidade Central

A unidade central realiza a busca de instruções inteiras e de ponto-flutuante. Ela é responsável pela execução de instruções do tipo: *load*, *store*, operações aritméticas com inteiros, operações de manipulação de bits, desvio do fluxo de controle, transferência de registradores inteiros para registradores de ponto-flutuante, teste e modificação dos registradores de controle, *flush* da *cache* e armazenamento de *pixel* (esta em conjunto com a unidade gráfica). Todas as instruções são de 32 bits. Somente as instruções *load* e *store* operam com a memória, todas as outras instruções operam com registradores. A maioria das instruções permite que o usuário especifique dois registradores fonte e um terceiro registrador destino onde será armazenado o resultado.

A característica principal da unidade central está na sua capacidade de executar a maioria das instruções em um mesmo ciclo de relógio. Ela possui um *pipeline* de quatro estágios: busca, decodificação, execução e escrita.

Algumas otimizações foram incluídas no conjunto de instruções de desvio. As instruções de desvio condicional podem ser executadas com ou sem "atraso". As instruções com atraso permitem que o processador execute a instrução localizada na posição de memória seguinte à instrução de desvio, enquanto o endereço alvo do desvio é buscado.

A unidade central (*Core Unit*) é responsável pela execução das instruções *load* e *store* de ponto-flutuante. Existem dois tipos de *load* para ponto-flutuante: FLD(*Floating-point Load*) e PFLD(*Pipelined Floating-point Load*). A instrução FLD transfere um dado da memória *cache* para um registrador de ponto-flutuante, ou traz o dado da memória, armazenando-o em uma linha da *cache* (se o dado não estiver na *cache*), e carrega-o para um registrador de ponto-flutuante. Um dado de 128 bits pode ser acessado, na *cache* de dados, em um ciclo de relógio. A capacidade de realizar um carregamento de um dado de 128 bits em um ciclo de relógio é fundamental para manter a unidade de ponto-flutuante executando continuamente.

Para acessar um dado muito grande (que não caiba na *cache*), a unidade central usa a instrução PFLD. O carregamento pelo *pipeline* transporta o dado diretamente para o registrador de ponto-flutuante sem colocá-lo na *cache* na ocorrência de uma falha. O *pipeline* de carregamento possui três estágios.

A unidade central provê, também, instruções *lock* e *unlock*. A instrução *lock* é utilizada para travar o barramento até a próxima instrução *unlock*. Entre uma instrução *lock* e a instrução *unlock* correspondente podem ser executadas no máximo 32 instruções.

II.6 Unidade de Gerenciamento de Memória

A unidade de gerenciamento de memória implementa as características básicas de memória virtual paginada incluindo a proteção de memória a nível de páginas. As técnicas de gerenciamento de memória são as mesmas dos microprocessadores 386 e 486 da Intel.

O processador realiza a tradução de endereços utilizando tabelas de páginas. São utilizados dois níveis hierárquicos de tabelas de páginas. No nível mais alto está o diretório de páginas que contém entradas para 1024 tabelas de páginas. A tabela de páginas, no segundo nível, endereça até 1024 páginas.

Para diminuir o *overhead* causado pelos acessos às tabelas de páginas, o processador guarda as informações de tradução referentes às 64 páginas mais recentemente utilizadas em uma memória associativa chamada de TLB (*Translation Lookaside Buffer*). Esta memória atua como uma *cache* de tradução de endereços. Quando o processador não encontra a informação de tradução para uma determinada página no TLB, ele procura esta informação na tabela de páginas, localizada na memória principal, e atualiza o TLB.

A proteção de memória é atingida através de dois bits da tabela de páginas. São eles W(*Writable*) e U(*User*). O processador realiza a proteção durante a tradução de endereços. Para proteger a área do sistema operacional de acessos indevidos de programas de usuários, o i860 utiliza o conceito de privilégio (ou modo

de operação). Este conceito é implementado atribuindo-se a cada página um dos dois níveis: Nível Supervisor(U = 0) e Nível Usuário(U = 1). Quando o processador está executando no Nível Supervisor, todas as páginas são endereçáveis, mas, quando ele está executando no Nível Usuário, apenas as que pertecem ao Nível Usuário são endereçáveis. No Nível Supervisor, todas as páginas podem ser lidas e apenas as que estiverem com W=0 podem ser alteradas (é importante observar que o i860 possui um modo que, estando no Nível Supervisor, permite que todas as páginas, seja qual for o valor de W, sejam modificadas). No Nível Usuário todas as páginas do processo corrente podem ser lidas e apenas as que estiverem com W=0 podem ser modificadas.

II.7 Unidade de Ponto-Flutuante

A unidade de ponto-flutuante pode operar com números reais com precisão simples e dupla. Ela possui duas unidades *pipeline* de três estágios, uma para soma outra para multiplicação.

As operações de soma e multiplicação podem ser executadas em dois modos: escalar e *pipelined.* No modo escalar novas operações de ponto-flutuante só podem ser iniciadas quando a operação anterior estiver terminada. Para cada operação de soma (independente da precisão) e para cada multiplicação com precisão simples são necessários três ciclos do relógio. A multiplicação com precisão dupla requer quatro ciclos do relógio.

No modo *pipelined* para cada nova operação o *pipeline* é avançado de um estágio. Desta forma, é possível produzir um resultado a cada ciclo.

No exemplo da Figura II.3, a unidade *pipeline* de soma inicia a soma de f2 com f7. Como esta é a primeira instrução da série e o *pipeline* ainda não está cheio, o resultado obtido é descartado (f0 é sempre zero). Em cada ciclo sucessivo, uma instrução de soma avança o *pipeline.* No quarto ciclo, o resultado da soma da primeira instrução estará disponível no estágio de resultado e é armazenado no registrador especificado pela quarta instrução.

Conteúdo dos Registradores Antes da Soma

f2	2	f7	7	f12			
f3	3	f8	8	f13			
f4	4	f9	9	f14			
f5	5	f10	10	f15			
f6	6	f11	11	f16			

Sequência de Instruções	*Pipeline* de Soma			Destino
pfadd.ss f2, f7, f0	2+7	-	-	Nenhum
pfadd.ss f3, f8, f0	3+8	2+7	-	Nenhum
pfadd.ss f4, f9, f0	4+9	3+8	2+7	Nenhum
pfadd.ss f5, f10, f12	5+10	4+9	3+8	f12 ← 9
pfadd.ss f6, f11, f13	6+11	5+10	4+9	f13 ← 11
pfadd.ss f0, f0, f14	0	6+11	5+10	f14 ← 13

Figura II.3: Exemplo de utilização do *pipeline* de soma

Após a quarta instrução, obtemos um resultado por ciclo. Quando um número muito grande de operações é realizada, o *overhead* para preencher e esvaziar o *pipeline* torna-se desprezível. Caso apenas uma ou duas operações forem realizadas o modo escalar é mais indicado.

Os *pipelines* de soma e multiplicação permitem a sobreposição (*overlap*) de instruções sequenciais quando múltiplas instruções estão sendo executadas, ao mesmo tempo, em vários estágios diferentes. Contudo, o i860 vai além em termos de sobreposição de instruções. Ele possui um modo de processamento, chamado modo dual, que inicia duas instruções ao mesmo tempo: uma para a unidade central (*core unit*) e outra para a unidade de ponto-flutuante. No modo dual uma instrução tem 64 bits, com os 32 bits mais baixos representando a instrução de ponto-flutuante e os 32 bits mais significativos representam a instrução a ser executada na unidade central.

Dentro da unidade de ponto-flutuante existe, ainda, outra forma de sobreposição de instruções: as operações duais. Estas operações permitem que o usuário especifique que uma soma e uma multiplicação sejam executadas simultaneamente. Com apenas uma instrução de 32 bits é iniciada uma operação de soma em conjunto com uma operação de multiplicação. Apesar dessas duas operações

necessitarem de seis operandos, o formato da instrução especifica apenas três. Os outros operandos são providos pela utilização dos registradores de propósito especial KR, KI, e T.

Com o modo dual e as operações duais o processador é capaz de executar três operações ao mesmo tempo.

O exemplo seguinte mostra a aceleração obtida pelo uso do paralelismo na arquitetura.

$$for\ i := 1\ to\ 100\ do$$

$$X[i] := A[i] * B[i] + C$$

O código relativo ao corpo desse loop no modo escalar poderia ser:

$$FMUL \quad A[i],\ B[i],\ temp$$

$$FADD \quad temp,\ C,\ X[i]$$

Cada multiplicação e cada soma requerem três ciclos de relógio. Portanto este loop precisaria de um total de 600 ciclos de relógio.

Utilizando o *pipeline* e as operações duais, podemos obter um novo resultado a cada ciclo, se empregarmos a seguinte codificação:

$$M12TPM \quad A[i],\ B[i],\ X[i-6]$$

A instrução M12TPM realiza uma multiplicação com os dois primei-

ros operandos (A[i] e B[i]) e realiza uma soma com o resultado da multiplicação e o conteúdo do registrador T. O resultado da soma é armazenado no terceiro operando (X[i-6]). Devido aos três estágios do *pipeline* de soma e de multiplicação, o resultado disponível vem da operação que iniciou 6 ciclos de relógio anteriormente. Com a utilização do *pipeline* e a sobreposição das operações de soma e multiplicação o loop que no outro modo requeria 600 ciclos, nesse modo necessita de somente 100 ciclos de relógio (excluindo o preenchimento e esvaziamento do *pipeline*).

Apesar de ser uma máquina superescalar, o i860 é um processador bastante eficiente na realização de operações vetoriais. As características de *pipeline* das operações com a memória principal e das operações de ponto-flutuante permitem que um alto desempenho seja atingido. Dessa forma, um possível uso da conjunção T-800/i860 pode ser: o T-800 solicita ao i860 a execução de rotinas vetoriais.

II.8 Unidade Gráfica

A unidade gráfica possui uma lógica especial que manipula inteiros de 64 bits além de dar suporte a algoritmos para processamento gráfico em 3-D. Esta unidade pode operar em paralelo com a unidade central. Ela contém o registrador Merge e realiza múltiplas adições em inteiros que são armazenados nos registradores de ponto-flutuante.

II.9 Memória Cache

Além do TLB (*cache* utilizada para tradução de endereços), o i860 incorpora duas memórias *caches*: uma para o armazenamento de dados e outra para instruções. A memória *cache* de dados tem 8 Kbytes de capacidade e a memória *cache* de instruções 4 Kbytes, ambas possuem uma organização associativa por conjuntos, onde cada conjunto possui dois blocos de 32 bytes.

Ambas as *caches* utilizam endereços virtuais. Desta forma, dentro de um determinado contexto, cada endereço virtual só pode referenciar um endereço físico. Quando do acesso à *cache* de dados o TLB é consultado para fins de proteção

das páginas. Durante a troca de contexto a *cache* de instruções deve ser invalidada pelo programador e a *cache* de dados esvaziada (*flushed*).

A atualização dos dados na memória *cache* é feita segundo a política de *write-back*, ou seja, o processador faz as atualizações na *cache*, e a atualização da memória principal é retardada até que a linha alterada seja descartada da *cache*. Nesse momento, o dado atualizado é "escrito de volta" na memória principal. A utilização desta política impede a propagação de todas as escritas para o barramento externo, reduzindo assim a formação de um gargalo. Uma política mais sofisticada de coerência da *cache* deve ser implementada por *software*.

Cada processador pode armazenar na *cache* código e dados privativos de cada processo, enquanto que os dados compartilhados não podem ser armazenados em memória *cache*. Existe um bit (*cacheable bit*) na entrada da tabela de páginas que permite o controle, por *software*, do que é, ou não, armazenado nas *caches*.

Capítulo III

A Linguagem Occam e o Transputer

Visando facilitar a apresentação das características do Núcleo, vamos descrever o ambiente para programação paralela que ele deve reproduzir. Neste capítulo faremos uma breve descrição da linguagem Occam [26] e como algumas de suas construções são implementadas pelo processador T-800 da linha Transputer da Inmos.

III.1 Uma Máquina Occam

A especificação da linguagem Occam e dos processadores da linha Transputer se deu de uma maneira um tanto peculiar. As características de *hardware* do Transputer foram escolhidas para dar suporte ao modelo Occam de concorrência e o seu conjunto de instruções foi projetado para otimizar a execução de programas Occam [29]. Apesar do Transputer ser capaz de executar código derivado de programas escritos em C, FORTRAN, Pascal ou em *Assembly*, ele é uma "máquina Occam". Em outras palavras, o Transputer incorpora em *hardware* o modelo Occam de programação, com diversas construções da linguagem sendo diretamente realizadas pelas primitivas do processador.

III.2 A Arquitetura do Transputer

O termo **Transputer** se refere a uma família de dispositivos VLSI programáveis incluindo controladoras de discos, processadores de ponto-flutuante, processadores gráficos, dispositivos de processamento de sinal e processadores de propósito geral de 16 e 32 bits [36]. Estamos interessados na arquitetura do processador T-800 de propósito geral de 32 bits. A Figura III.1 mostra os principais componentes do processador T-800.

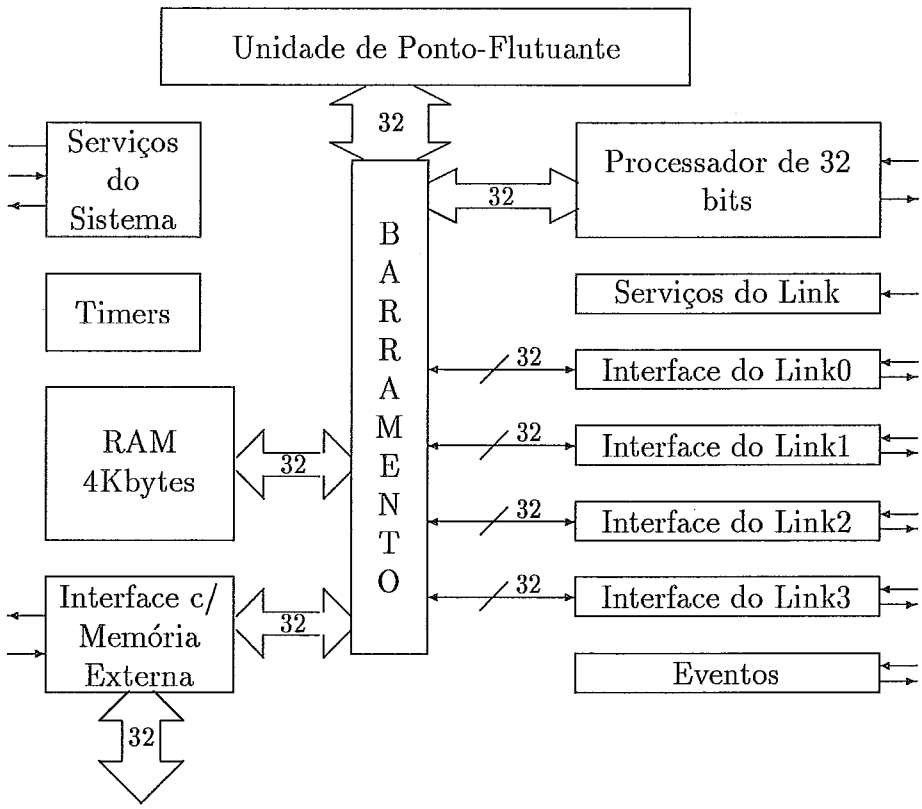

Figura III.1: O processador T-800

O processador contém uma memória interna de 4K bytes e quatro *links* para comunicação direta com outros Transputers. O barramento é conectado a uma memória externa e é implementado de modo que o espaço de endereçamento da memória externa seja uma continuação do da memória interna.

Os quatro *links* estão conectados ao processador central via quatro *interfaces de link*. Essas interfaces podem, idependentemente, manipular os *links* de comunicação (incluindo o acesso direto à memória). Por conseguinte, o T-800 pode se comunicar simultaneamente através dos quatro *links* (em ambas as direções) e realizar processamento interno. A comunicação é feita no modo ponto-a-ponto.

III.3 A Linguagem Occam

O modelo básico de um programa Occam é o de um conjunto de processos concorrentes que se comunicam. A linguagem Occam foi influenciada pela linguagem CSP (*Communicating Sequential Processes*) de C.A.R. Hoare [17]. A linguagem CSP se baseia na seguinte idéia: computações paralelas podem ser implementadas por um número de máquinas sequenciais que se comunicam umas com as outras através de uma rede de canais.

III.3.1 Processos Primitivos

Em Occam o elemento fundamental de um programa é o processo. Por definição, um processo inicia, realiza algumas ações e termina. Um programa Occam consiste de vários processos concorrentes. Existem apenas cinco processos primitivos, são eles:

- STOP
- SKIP
- Atribuição
- Entrada
- Saída

STOP

É um processo que não realiza nenhuma ação e nunca termina. É utilizado comumente para casos de erro, impedindo que os processos seguintes sejam executados.

SKIP

Também não realiza nenhuma ação. É um processo que está sempre pronto para executar, terminando imediatamente após seu início. Representa um processo nulo e é usado quando nenhuma ação deve ser realizada.

ATRIBUIÇÃO

A atribuição, em Occam, é identica à atribuição nas linguagens C, Pascal ou BASIC:

$$V := s;$$

onde V é uma variável e s é uma expressão do mesmo tipo de V. Após este comando, V conterá o valor de s.

ENTRADA e SAÍDA (Comunicação)

As primitivas de entrada e saída da linguagem implementam um mecanismo troca de mensagens através de um **canal**. Canais são entidades de um

programa Occam que se parecem com variáveis. Um canal tem um identificador, um tipo, e deve ter sido declarado antes do uso. Um canal conecta exatamente dois processos em apenas uma direção, uma conexão bidirecional requer dois canais, um para cada direção. Os comandos de saída (envio de uma mensagem) e entrada (recebimento de uma mensagem) são da forma:

ch ! s – saída

ch ? V – entrada

onde ch é um canal, s uma expressão, e V uma variável. Esses comandos especificam, respectivamente, "envie o valor da expressão s pelo canal ch", e "receba pelo canal ch um valor e o armazene em V". Combinando os comandos acima teremos o seguinte efeito:

$V := s$; (estando V em um processo e s em outro)

O compartilhamento de variáveis globais é proibido em Occam. A única maneira possível de se transferir um valor de um processo para outro é através de um canal.

A troca de mensagens é feita de modo bloqueante. Se um canal é utilizado para entrada em um processo X e para saída em um outro processo Y, a comunicação só é realizada quando ambos os processos estiverem preparados para trocar uma mensagem, ou seja, o processo Y espera até que X esteja preparado para receber a mensagem, e vice-versa (i.e., o processo X espera até que Y esteja preparado para enviar a mensagem).

Canais são utilizados também para implementar a interação de um programa Occam com o ambiente externo. Como em qualquer linguagem de alto nível, as operações de E/S dependem dos detalhes da implementação da linguagem. Entretanto, o modelo básico de E/S em Occam é muito simples. Um programa que deseja se comunicar com a *VDU* e com o teclado, por exemplo, deve utilizar dois canais. Através de um desses canais o programa envia (executando um comando de saída) caracteres para a *VDU* e, através do outro canal o programa recebe (executando um comando de entrada) os caracteres do teclado.

Existem canais especiais para temporização que retornam o valor do relógio local de tempo real. Estes canais são declarados como *TIMER* e cada processo só pode declarar um canal desse tipo. Sobre canais *TIMER* é possível a executar apenas comandos de entrada. Este canal tem como característica principal o fato de estar sempre preparado para saída (i.e., um processo nunca fica bloqueado quando realiza uma entrada em um canal *TIMER*). Para que um processo fique bloqueado esperando por um intervalo de tempo, a linguagem Occam provê a seguinte construção:

chrel ? AFTER T

onde *chrel* é um canal *TIMER* declarado previamente, e T um número inteiro. O processo que executar esse comando de entrada sobre *chrel* ficará bloqueado até que o valor do relógio de tempo-real seja maior ou igual ao conteúdo de T.

III.3.2 Construções

Processos primitivos podem ser combinados formando uma construção. Uma construção também é um processo e pode ser usada como componente de outra construção. Além das construções usuais IF, CASE e WHILE, existem outras três classes de construções:

Sequencial

A definição da construção Sequencial é:

```
SEQ
  P1
  P2
  P3
  ...
```

Os processos componentes (`P1, P2, P3, ...`) da construção são executados sequencialmente. Cada componente é iniciado depois que o outro termina, e a construção termina quando o último componente terminar.

Paralela

A definição da construção Paralela é:

```
PAR
  P1
  P2
  P3
  ...
```

Os processos componentes (`P1, P2, P3, ...`) da construção são executados concorrentemente. A construção termina quando todos os processos componentes terminarem.

Alternativa

A definição da construção Alternativa é:

```
ALT
  G1
    P1
  G2
    P2
  G3
    P3
  ...
```

onde `Gi` é um **guarda** e `Pi` um subprocesso chamado de **lista de comandos**, o conjunto {`Gi`,`Pi`} é chamado de **comando guardado.**

A construção ALT deriva dos comandos guardados de Dijkstra [7]. Ela introduz na linguagem o não-determinismo na comunicação. Através dessa construção, é possível selecionar qual o processo concorrente (dentre diversos) que trocará mensagem com o processo executando a construção.

A sintaxe de um guarda é:

guarda = comando de entrada
| (expressão booleana) & comando de entrada
| (expressão booleana) & SKIP

O guarda é dito **pronto** quando a expressão booleana é verdadeira e o comando de entrada associado pode executar. É importante observar que o processo SKIP está sempre pronto para executar, e que um comando de entrada só estará preparado para a troca de mensagens quando existir um outro processo

preparado para realizar a saída correspondente. O comando de entrada pode representar também a espera por um intervalo tempo, neste caso, o guarda estará **pronto** após o transcurso intervalo de tempo (i.e., o *delay*) especificado.

A execução da construção ALT inclui a escolha de um comando guardado (cujo guarda esteja **pronto**) e a execução da respectiva lista de comandos. Quando não há nenhum guarda **pronto**, o processo ALT fica suspenso até que o estado de um dos guardas seja modificado. Quando há mais de um guarda **pronto** a escolha é arbitrária (i.e., a definição da linguagem não especifica nenhum critério de seleção).

III.4 A Implementação da Linguagem Occam no Transputer

O Transputer contém facilidades para prover uma implementação eficiente do modelo Occam de concorrência e comunicação. Essas facilidades incluem:

- Suporte para processos sequenciais;
- Suporte para processos concorrentes;
- Suporte para comunicação;
- Suporte para implementação da construção ALT.

III.4.1 Suporte para Processos Sequenciais

Para execução de processos sequenciais são usados seis registradores do processador:

- ponteiro para área de trabalho - aponta para a área de memória onde as variáveis locais do processo estão armazenadas;
- ponteiro de instrução - aponta para a próxima instrução a ser executada;
- registrador de operandos - utilizado na formação de operandos de instruções (imediatos ou endereços);