Um Ambiente De Programação "Data Flow"

Miguel Menasche * Denis Maciel Maia**

RESUMO

Este trabalho, apresenta o projeto de um ambiente para programação "***data flow***", atualmente em desenvolvimento no Departamento de Engenharia Elétrica da PUC/Rio. Este ambiente é composto por: uma placa processadora "***data flow***" para ser utilizada em uma plataforma hospedeira tipo microcomputador compatível com PC; um montador assembler; um editor gráfico; um simulador/depurador e uma linguagem "***data flow***" de alto nível. O objetivo para o desenvolvimento deste ambiente é, inicialmente, o de ganhar familiaridade com uma filosofia diferente de processamento que explora bastante o paralelismo, além de estudar futuramente o comportamento desta em situações reais no tratamento de imagens.

ABSTRACT

This work describes a data flow programming environment that is being held at the Departamento de Engenharia Elétrica da PUC/Rio. This environment comprises: a data flow processor card to be mounted in a host microcomputer compatible PC; an assembler; a graphics editor; a simulator and debugger and a data flow high level language. The aim of this project is, in short term, to gain familiarity with a different processing philosophy that highly exploits parallelism, and subsequently, to study its behavior in real situations in the image processing field.

* Professor do Departamento de Engenharia Elétrica da PUC/Rio
Rua Marquês de S. Vicente 225, Rio de Janeiro RJ CEP 22453-900
Tel: (021) 529 9505

** Aluno de Engenharia de Computação, Bolsista de Iniciação Científica pelo CNPq.

Introdução

A busca pelo melhor desempenho nos sistemas computacionais tem tradicionalmente evoluído, através de três dimensões distintas e complementares: de um lado, o desenvolvimento tecnológico tem permitido a obtenção de componentes eletrônicos com tempos de chaveamento cada vez menores; de outro, a melhor compreensão dos processos envolvidos nesses sistemas tem levado à novas organizações de seus componentes internos, visando à otimização dos percursos a serem percorridos pelas informações; finalmente, a exploração das possibilidades de execução simultânea de diversas operações tem levado a índices crescentes de paralelismo nos seus mais diversos níveis.

O primeiro desses caminhos tem conseguido resultados extremamente expressivos ao longo desse meio século na história dos computadores eletrônicos. No entanto, o espaço que ainda pode ser percorrido através desta via parece ser limitado e, a persistir a mesma taxa de evolução nos tempos de chaveamento, prevê-se a chegada, dentro dos próximos trinta anos, à proximidade de limites físicos intransponíveis. Assim sendo, para continuar na busca incessante de melhores índices de desempenho é necessário explorar ao máximo as duas vias restantes as quais são intimamente interligadas e constituem o que se costuma denominar a arquitetura do sistema.

A arquitetura tradicional, conhecida como modelo de ***von Neumann***, na realidade segue uma proposta efetuada, ainda no século passado, por ***Babbage*** quando da apresentação da sua ***Analytical Engine***. Por este modelo, operações lógicas e aritméticas básicas devem ser efetuadas seqüencialmente sobre dados armazenados previamente em um conjunto de registros.

Ao longo deste primeiro meio século da existência dos computadores, este modelo tem sido bastante aperfeiçoado, mas a maioria esmagadora dos computadores modernos não se afasta demasiadamente desta concepção.

Apesar de neste período a imaginação dos projetistas e arquitetos não ter deixado de propor outras alternativas que procurassem explorar mais profundamente o paralelismo, diversas razões contribuíram para a manutenção maciça do modelo de ***von Neumann***: inicialmente, diversas propostas de uso de paralelismo foram efetuadas quando a tecnologia ainda não tinha condições para permitir sua implantação a um custo razoável; a adoção de paralelismo, em qualquer dos seus diversos níveis, implicava quase que invariavelmente em um custo muito superior oriundo da multiplicação do hardware necessário para viabilizá-lo; o modelo tradicional e a cultura existente amplamente difundidos permitiam resolver com

sucesso a maior parte dos problemas computacionais dentro de limites bastante aceitáveis; a adoção do paralelismo traz consigo uma complexidade maior e problemas novos que devem ainda ser pesquisados e resolvidos.

Com o desenvolvimento das técnicas de integração em alta escala (**VLSI**), o primeiro motivo deixou de existir, o segundo perdeu um peso relativo considerável, os problemas computacionais correntes passaram a se ressentir das limitações das arquiteturas tradicionais, mas a inércia para adotar uma nova maneira de pensar aliada à existência de uma cultura de paralelismo pouco difundida têm impedido uma adoção mais rápida de modelos alternativos.

Dentre os modelos de arquiteturas alternativos, surge na década de 70, o modelo denominado "***data flow***" proposto por ***Jack Dennis***, no qual as operações lógicas e aritméticas não mais seriam executadas seqüencialmente seguindo uma ordem previamente especificada pelo programa ("***control flow***"), mas sim, uma ordem definida pela disponibilidade dos dados ("***data flow***") onde a medida em que dados são gerados é também habilitada a execução de todas as operações que deles dependem. A execução dessas operações pode ser simultânea ou em qualquer ordem dependendo exclusivamente da disponibilidade de unidades de processamento livres naquele momento. Em consequência, tal tipo de máquina não mais possui um registro que aponta para a instrução do programa a ser executada (contador de programa). O próprio conceito de programa deve ser revisto. Em uma máquina "***data flow***" um programa não é mais fornecido sob a forma de uma lista de instruções a serem executadas seqüencialmente mas sim fornecido sob a forma de um grafo no qual devem ser explicitadas as relações de dependência ou precedência na execução das instruções. Pares de operações que não estejam especificadas em uma relação de precedência poderão ser executados em paralelo.

Até poucos anos atrás, protótipos de máquinas "***data flow***" tinham existência apenas em centros de pesquisa, dos quais os mais famosos foram os do MIT e o da Universidade de Manchester.

Apesar dos primeiros microprocessadores baseados na arquitetura "***data flow***" terem surgido em 1984, com o lançamento comercial pela **NEC** do **μPD7281**, possibilitando a construção de máquinas com esta filosofia a baixo custo, o seu uso tem sido ainda bastante restrito evidenciando as dificuldades de adoção de um novo paradigma computacional.

Neste sentido, a fim de aumentar nossa familiaridade com uma nova filosofia de programação, iniciamos no Departamento de Engenharia Elétrica da PUC-Rio o desenvolvimento de um ambiente de programação "***data flow***" que consiste em uma placa

processadora utilizando o **μPD7281** da **NEC** para ser utilizada em um computador hospedeiro do tipo PC compatível, um mini assembler "***data flow***" (**MAD**), um editor gráfico para grafos "***data flow***", um simulador e depurador de programas e futuramente o desenvolvimento de uma linguagem de mais alto nível.

O objetivo principal deste projeto, é o de dispor de uma plataforma "***data flow***" na qual seja possível ganhar experiência com a realização de programas e eventuais problemas inerentes a uma arquitetura paralela de uso ainda pouco difundido.

No estado atual, o projeto deste ambiente (**dfEnv**) já teve implementadas sua placa processadora (**dfP**), seu mini assembler (**dfA**) ou **MAD** e o seu editor gráfico (**dfG**), este ainda com alguns problemas que necessitam ser corrigidos. O simulador/depurador (**dfSim**) encontra-se em fase de desenvolvimento e o projeto para a linguagem de alto nível (**dfL**) ainda não foi iniciado.

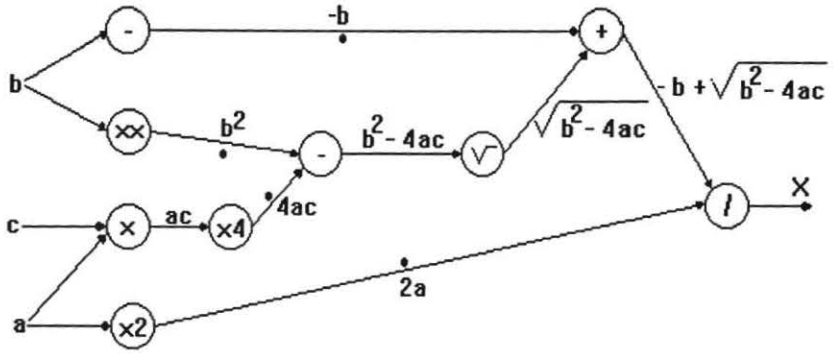
Neste trabalho, procuraremos de um lado apresentar características do projeto desenvolvido e em desenvolvimento, assim como mostrar opções efetuadas em função de trabalharmos com uma arquitetura alternativa, em geral, e em função do chip **μPD7281**, em particular.

1. Princípios de Uma Arquitetura "Data Flow"

Nesta seção faremos uma breve introdução à filosofia "***data flow***" com o objetivo apenas de apresentar a nomenclatura utilizada durante o decorrer deste trabalho.

Como visto, no capítulo introdutório, máquinas "***data flow***" não mais têm o seu fluxo de instruções definidos a priori por um programador através de um programa que consiste em uma lista seqüencial de instruções básicas a serem realizadas pela unidade de processamento. Ao invés disso, instruções são executadas pelas unidades de processamento interno, a medida que os dados necessários para a sua execução venham se tornando disponíveis. Ao programador cabe a tarefa de apresentar ao computador "***data flow***" apenas um mapa de precedências entre operações, indicando aquelas que necessariamente dependem de resultados anteriores. Assim, para executar uma expressão do tipo $\mathbf{X := (-b + \sqrt{(b^2 - 4ac)})/2a}$ em uma máquina tradicional, seria necessário executar 9 operações aritméticas em uma seqüência semi-arbitrária. Em uma máquina "***data flow***", estas mesmas 9 operações podem ser executadas com um certo grau de paralelismo, por exemplo: a operação **-b** pode ser executada simultaneamente com a operação $\mathbf{b^2}$, que também pode ser executada simultaneamente com a operação **2a**, e assim, sucessivamente com uma série de outras operações. Entretanto, a

operação de subtração na expressão b^2 - 4ac somente pode ser executada quando as operações b^2 e 4ac estiverem integralmente concluídas. Para informar à uma máquina "*data flow*" as múltiplas relações de precedência e de paralelismo entre as diversas operações, uma forma que parece ser bastante natural é a da sua apresentação através de um grafo orientado, onde os seus nós, denominados *atores*, representam as operações e os seus arcos representam as relações de precedência. A este grafo denomina-se *grafo de fluxo*. Assim para a expressão acima o seguinte *grafo de fluxo* deveria ser produzido:



A disponibilidade de um dado em um determinado instante de tempo é representada por pequenas marcas ao lado de um arco e denominadas *fichas*. Assim, a figura acima mostra o *grafo de fluxo* para a representação das relações de precedência para a obtenção de **X**, no instante em que os valores **-b**, b^2, **4ac** e **2a** já foram calculados. Uma operação somente pode ser iniciada quando todos os arcos que chegam ao nó que corresponde a esta operação contiverem suas respectivas fichas. A figura acima permite ainda observar o ***grau máximo de paralelismo*** assim como o ***caminho crítico*** para a obtenção do valor **X**. O ***grau máximo de paralelismo*** é 4, pois é o maior número de operações paralelas que podem ser realizadas em um determinado instante e o ***caminho crítico*** é aquele situado entre a entrada **c** e a saída **X**.

Pode-se ainda concluir que em uma máquina "*data flow*", somente será possível beneficiar-se do ***grau máximo de paralelismo*** de uma determinada computação se esta máquina contiver um número de elementos processadores em quantidade superior ou igual a este grau.

2. A Placa Processadora dfP

Foi objetivo, durante a fase de desenvolvimento desta placa, poder dispor de uma plataforma confiável e bastante comum em qualquer laboratório universitário. Assim, foi excluída, logo no início, a possibilidade de construção de um computador "data flow" completo com sistema operacional, entrada e saída próprios que demandariam mais tempo de projeto e construção e poderiam trazer problemas suplementares não relacionados com o paralelismo propriamente dito. A disponibilidade crescente de computadores compatíveis com **PC** em diversos dos nossos laboratórios, praticamente definiu este tipo de computador como hospedeiro para uma placa que deveria funcionar com a filosofia "***data flow***" construída em volta do microprocessador **μPD7281** da **NEC**. O próprio processador, **μPD7281**, por construção, já prevendo a sua utilização juntamente com uma CPU hospedeira contribuiu para a cristalização desta decisão. Desse modo pode-se aproveitar todo o potencial dos computadores **PC** compatíveis para entrada/saída, programação, depuração e teste sem que necessariamente a placa **dfP** esteja efetivamente conectada ao mesmo. A placa **dfP** somente se torna necessária quando for preciso rodar os programas "***data flow***" propriamente ditos.

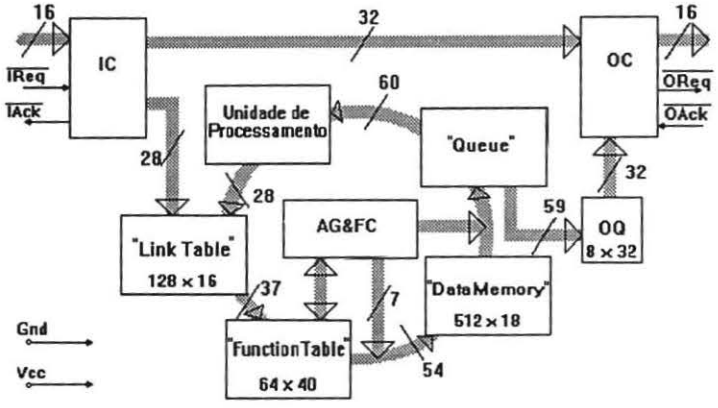
Para melhor compreensão desta placa e do ambiente proposto, torna-se necessário um conhecimento prévio do processador **μPD7281**.

2.a. O processador μPD7281

O processador **μPD7281**, também denominado de **Image Pipelined Processor**, é um dispositivo **VLSI**, **NMOS**, encapsulado em um circuito de 40 pinos, funcionando com uma única fonte de alimentação de 5 V e clock de 10 MHz. Foi projetado para ser utilizado juntamente com uma CPU hospedeira e para ser conectado com até outros 14 processadores **μPD7281** em cascata. Cada um dos processadores em cascata deve receber uma identificação ou número de módulo que é fornecido através de seus barramentos de entrada, imediatamente após o sinal de **RESET** inicial. A CPU hospedeira recebe a identificação 0.

Cada **μPD7281** possui um "pipeline" circular por onde trafegam as ***fichas*** e que contém uma única unidade de processamento.

A seguinte figura mostra um diagrama em blocos de um processador **μPD7281**:



Fichas de 32 bits chegam em um **µPD7281**, através do barramento de entrada de 16 bits, são convertidas pelo controlador de entrada *IC* e, conforme seu destino seja o processador corrente ou um outro processador, encaminhadas para o "pipeline" interno ou para o controlador de saída *OC*.

O diagrama em blocos do processador **µPD7281** deixa visível o "pipeline" circular composto por cinco blocos. Três dos blocos que constituem o "pipeline" circular são constituídos por dispositivos de memória. Os dois primeiros, sob a forma de tabelas (***"link table"*** e ***"function table"***), servem para armazenar internamente o ***grafo de fluxo*** de um programa que deverá ser realizado pelo **µPD7281**. A ***"link table"***, responsável pelo armazenamento dos arcos do ***grafo de fluxo*** e a ***"function table"*** pelo armazenamento dos nós do mesmo grafo. O terceiro dispositivo de memória é a ***"data memory"***, responsável pelo armazenamento de constantes e de dados temporários correspondentes à disponibilidade de ***fichas*** que se encontram esperando a geração de outras ***fichas*** necessárias para a execução de uma operação. Quando todas as fichas necessárias para a execução de uma operação se encontram disponíveis, este conjunto de informações é empacotado em uma única ficha e depositado na ***"queue"*** que é uma fila de ***fichas*** aguardando por uma unidade de processamento. As fichas na ***"queue"*** são então encaminhadas para a unidade de processamento do mesmo processador ou para uma fila de saída *OQ* a fim de se dirigirem à saída ou a outro processador através do controlador de saída *OC*. O último elemento do "pipeline" circular é a unidade de processamento, responsável pela execução das operações propriamente ditas e pela geração de uma nova ***ficha*** resultado que deve ser encaminhada à ***"link table"*** para prosseguimento contínuo do procedimento.

No processo, ao longo do "pipeline", ***fichas*** são transformadas, empacotadas e desempacotadas assumindo formas e tamanhos diferentes conforme o estágio em que se encontram. Devido ao "pipeline", até cinco instruções podem estar sendo processadas internamente em paralelo a uma taxa de até 5 MIPS.

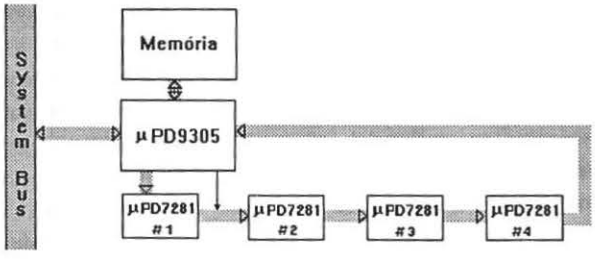
Em 1990 surgiu o **µPD72181,** sucessor do **µPD7281**, pino a pino compatível, mas com o dobro de sua velocidade e desempenho.

2.b. O projeto da placa dFP

Durante o projeto desta placa, foi estabelecido que a mesma deveria ser modular permitindo o acréscimo de mais processadores, se necessário, utilizando a capacidade inerente do **µPD7281** para ser interligado com outros em cascata. Assim a placa está provida de 4 soquetes de 40 pinos para serem ocupados por circuitos **µPD7281**, dos quais apenas um deve forçosamente ser ocupado.

Fora do **µPD7281**, o conceito de ***fichas*** é desconhecido, assim como internamente ao **µPD7281**, o conceito de endereços em memória também. Assim, torna-se necessário dispor de um circuito capaz de realizar esta conversão. Para isso a NEC providenciou juntamente com o **µPD7281**, um outro circuito integrado, denominado "**Memory Address Generator and Interface Controller** " - **MAGIC**, o **µPD9305**.

O projeto desenvolvido para a placa processadora **dFP**, envolve de 1 a 4 processadores **µPD7281**, um controlador **µPD9305** e uma memória RAM estática privativa de 128 K x 16 bits, conforme o diagrama em blocos a seguir:



O **µPD9305** é responsável pelo carregamento na memória, de um arquivo com um programa gerado em um computador PC. Em seguida este programa é transformado em uma sucessão de ***fichas*** pelo **µPD9305** que as encaminha para execução no interior dos diversos processadores **µPD7281** interligados em cascata.

A possibilidade de inserir diversos processadores em cascata neste placa permite que o seu desempenho máximo seja multiplicado pelo número de processadores efetivamente presentes.

Dependendo da aplicação, gargalos podem surgir, principalmente na comunicação com o barramento do sistema. Entretanto, para os fins propostos, esta é uma organização que se presta bastante bem para um ambiente de programação inicial.

3. O Montador MAD ou dFA

Apesar de sabermos que a **NEC** dispunha de um Assembler para este processador, tivemos uma grande dificuldade para adquirí-lo no mercado brasileiro. Assim, decidimos partir para a construção de uma versão própria. A construção deste montador foi realizada em **Turbo PASCAL** procurando seguir, na medida do possível, as especificações fornecidas pela **NEC** para o seu.

Um montador é, nada mais nada menos, do que um conversor direto de código que traduz palavras chaves de um arquivo texto para o seu respectivo código de máquina em binário. Além disso, um montador deve também resolver uma série de referências de endereços utilizadas pelas diversas variáveis de um programa.

As palavras chaves utilizadas para a programação de uma máquina "***data flow***" são aquelas necessárias à descrição do ***grafo de fluxo:*** seus nós (***atores***) e arcos. São estas as palavras que devem ser convertidas em binário pelo **dFA**.

O desenvolvimento de um montador para uma máquina "***data flow***", em geral, e para o **μPD7281**, em particular, apresenta especificidades que não são encontradas no caso de máquinas tradicionais. Em uma máquina "***data flow***", por exemplo, existe a necessidade de armazenar, sob alguma forma, o ***grafo de fluxo***. No **μPD7281**, isto é feito pela inicialização das tabelas internas "***link table***" e "***function table***".

Alterações na "***link table***" e na "***function table***" e em particular suas inicializações devem ser efetuadas através de instruções próprias ao **μPD7281**. Isto significa a execução de um programa de inicialização que deve ocorrer previamente à execução do programa que está sendo montado. Em conseqüência, o montador **dFA** além de efetuar as conversões do código em assembly para linguagem de máquina deve também ser o responsável pela geração

automática do programa de inicialização das tabelas internas, também em linguagem de máquina.

Como a placa **dFP** poderá conter diversos processadores **µPD7281**, o programa (***grafo de fluxo***) poderá ser também distribuído através dos mesmos. É, portanto, tarefa do **dFA** o carregamento, nas respectivas tabelas internas, dos trechos específicos do ***grafo de fluxo*** que serão executados por cada um.

Entre as características suplementares apresentadas pelo montador **dfA**, existe a possibilidade, opcional, de geração de um arquivo texto com um mapa do conteúdo e do espaço livre das tabelas existentes nas três memórias internas, "***link table***", "***function table***" e "***data memory***", assim como de um mapa das variáveis utilizadas.

4. O Editor Gráfico dFG

Programar em baixo nível em um computador "***data flow***", consiste portanto na concepção de um ***grafo de fluxo*** o qual deve ser devidamente traduzido e compreendido pela máquina para determinar as diversas possibilidades permitidas de paralelismo.

No caso do **µPD7281**, este ***grafo de fluxo*** é originalmente convertido manualmente, para uma descrição textual de nós e arcos que constituem a linguagem assembly. Só então, o montador **dFA** converte esta descrição textual para uma seqüência de zeros e uns que corresponde ao programa em linguagem de máquina.

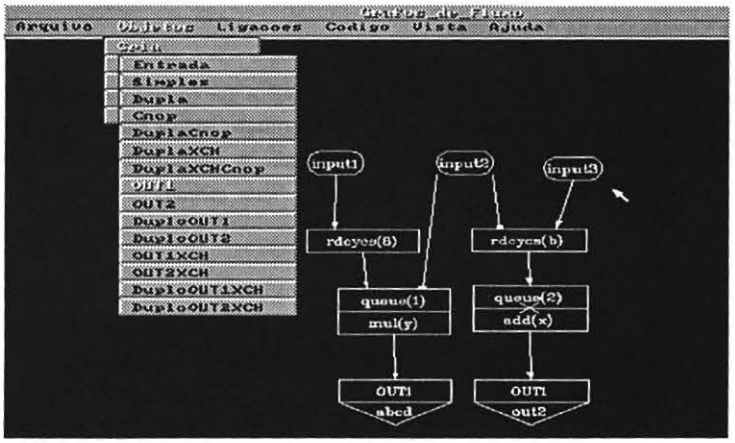
Este processo pode, sem dúvida, ser automatizado. Com o avanço recente dos dispositivos gráficos dos computadores do tipo PC, interfaces gráficas para programas que antigamente poderiam ser muito lentas, passaram a ser viáveis.

A idéia é que o usuário projete seu programa desenhando, diretamente na tela do computador, o ***grafo de fluxo*** para o seu problema através de um editor gráfico de grafos de fluxo **dFG** e que em seguida este converta automaticamente o grafo editado em um arquivo texto na linguagem assembly pronto para ser montado pelo **dFA**.

No caso do **µPD7281**, este editor deve ter uma complexidade adicional, pois os nós do ***grafo de fluxo*** não são uniformes conforme aparecem na primeira figura deste trabalho, mas podem assumir diversos formatos em função das instruções a que se referem.

O projeto deste editor gráfico foi realizado em **C**, utilizando-se de uma interface gráfica desenvolvida na **PUC** do Rio, o **INTGRAF**, que emprega o padrão gráfico internacional **GKS**. Como vantagem suplementar ele pode rodar em máquinas **SUN** com poucas adaptações.

A figura a seguir, mostra uma tela do **dFG**, ilustrando alguns dos possíveis tipos de nós utilizados:



5. Simulador/Depurador dFSim

A possibilidade de validar programas "*data flow*" sem a necessidade de possuir a placa **dfP**, foi um dos motivos principais que levaram ao desenvolvimento do **dfSim**.

Foi estabelecido que uma exigência para este simulador seria a de explorar ao máximo a capacidade gráfica do ambiente PC a fim de permitir uma melhor compreensão das operações internas da placa processadora.

Espera-se assim que o **dfSim** permita uma melhor compreensão da filosofia "***data flow***", em particular da arquitetura da **dfP**, através de uma visualização gráfica, em qualquer ciclo do(s) microprocessador(es), de todas as estruturas internas do(s) mesmo(s).

Em vista disso, foi decidido implementar este simulador no ambiente operacional MS-Windows. Além dos motivos acima que contribuíram para esta decisão, o ambiente Windows ainda possui as seguintes características favoráveis:

. Ser um sistema com alta portabilidade;

. Ser um sistema que utilize toda memória disponível do computador;

. Ser um sistema orientado a objetos, que é uma orientação indicada para futuras reutilizações do sistema.

5.a. A Visualização da Arquitetura

Cada microprocessador "***data flow***" possui estágios que contêm informações sobre o programa que está sendo processado. Como toda informação interna é importante para uma melhor compreensão da arquitetura, o **dfSim** permite a visualização do conteúdo dos seguintes estágios: "***link table***"; "***function table***"; "***data memory***"; "***queue***" e "***output queue***";

Além destes estágios o **dfSim** permite a visualização gráfica de um diagrama em blocos contendo os cinco estágios do "**pipeline**" interno com um dos estágios sempre marcado. Esta marcação corresponde ao estágio da ***ficha*** que será visualizada.

A visualização da ***ficha*** não será feita de forma binária, mas sim de forma interpretada, na qual sua informação é decomposta em campos decodificados antes de ser mostrada ao usuário. Esta é uma característica interessante porque a ***ficha*** transforma-se conforme muda de estágio, modificando a quantidade e o tipo de informação nela contida.

O mais importante desta visualização interna de um microprocesador "***data flow***" é a proposta de se poder visualizar graficamente informação de vários microprocessadores "***data flow***" ao mesmo tempo. Por isto, propôs-se que o **dfSim** trabalhasse em um **modo MDI** (**M**ultiple **D**ocument **I**nterface).

5.b. O Ambiente MDI

Este modo consiste em uma janela principal, no caso o **dfSim**, funcionando como ***mesa de trabalho*** onde estão todos seus documentos, no caso os microprocessadores do **dfEnv**, em forma de ***janelas filhas*** gerenciadas pela janela principal.

Esta opção foi adotada prevendo uma maior flexibilidade na visualização das informações, que podem ser muitas caso estejamos trabalhando com mais de um microprocessador "*data flow*".

No caso de um programa "*data flow*" utilizar mais de um microprocessador o procedimento de leitura do **dfSim** faz uma varredura em todo o arquivo que o especifica, a fim de determinar o número de microprocessadores que serão visualizados.

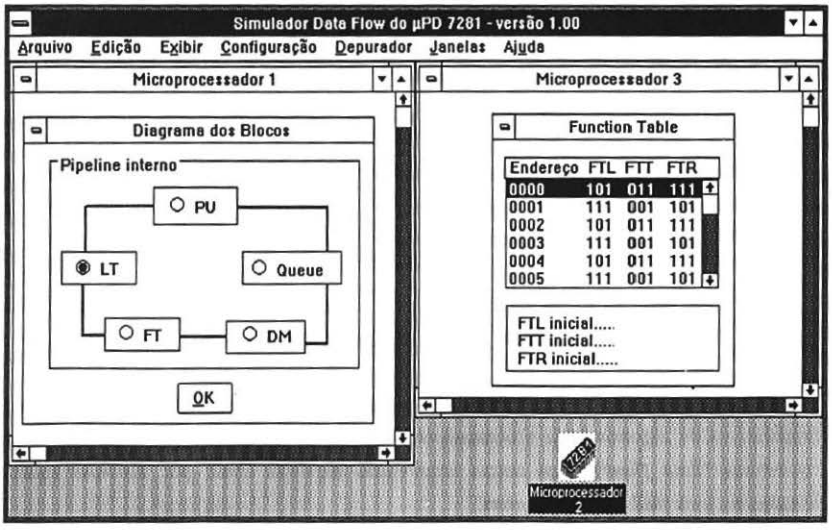
Outras características convenientes em um **ambiente MDI** são:

. Poder transformar um microprocessador em um ícone (minimização), que em qualquer momento pode ser restaurado, ou seja, voltar a forma de janela;

. Poder enquadrar todos os microprocessadores de forma que todos são visualizados, sem nenhuma sobreposição, ocupando todo o espaço da janela principal;

. Poder posicionar os microprocessadores, em qualquer momento, em cascata.

A figura abaixo mostra uma possível tela do dfSim apresentando três microprocessadores: o primeiro com seu diagrama em blocos, o segundo minimizado sob a forma de um ícone e o terceiro exibindo sua "*function table*".



6. Linguagem de Alto Nível dFL

Em um ambiente de programação "***data flow***" seria muito importante poder dispor também de uma linguagem de alto nível.

Sabemos da existência de algumas linguagens de alto nível, especificamente desenvolvidas para máquinas "***data flow***". É nossa intenção dotar também este ambiente com uma linguagem de mais alto nível que permita expressar diversas das características de programação de maneira adaptada aos processadores "***data flow***", em geral, e ao **μPD7281**, em particular.

Os passos iniciais para o estudo e desenvolvimento desta linguagem, no entanto, não foram ainda iniciados.

7. Comentários Finais

A busca de modelos alternativos que favoreçam o paralelismo assim como o lançamento pela NEC do **μPD7281**, levou-nos a estudar mais detalhadamente a filosofia proposta por Jack Dennis no inicio da década de 70, denominada de "***data flow***". De todas as filosofias propostas que procuram explorar o paralelismo, esta talvez seja aquela que mais se afasta do modelo tradicional, conhecido como modelo de von Neumann.

Programar em uma máquina "***data flow***" exige uma maneira bastante diferente de raciocínio daquela habitualmente utilizada.

A necessidade de ganhar experiência com esta filosofia de programação levou-nos ao desenvolvimento de um ambiente composto por uma placa processadora, um montador, um editor gráfico, um simulador/depurador e, futuramente, uma linguagem de alto nível.

Embora ainda não operacional, o desenvolvimento deste ambiente já nos permitiu ganhar familiaridade com o **μPD7281**, que era um dos nossos objetivos.

Como continuação deste trabalho, pretendemos efetuar uma série de estudos comparativos de algoritmos de processamento de imagem em arquiteturas tradicionais, em arquiteturas "***data flow***" e em outras arquiteturas paralelas como por exemplo aquelas baseadas em uma malha de ***transputers***.

Referências

Manuais da NEC sobre o **μPD7281** e o **μPD9305**.

"Data Flow Languages"
William B. Ackerman
Computer, fevereiro de 1982, pp. 15-25.

Proposta de Projeto de Coprocessador "Dataflow".
Eliseu Monteiro Chaves Filho, Miguel Menasche
Nota Interna DEE - PUC/Rio, janeiro de 1989.

Proposta de um Montador (Assembler) para a Placa Co-Processadora "Data Flow".
Alexandre Nigri, Marcus Vinicius Ítala Ferreira.
Nota Interna DEE - PUC/Rio, julho de 1990.

Proposta de um Ambiente de Programação Data Flow dFPE.
Átila Leão Flores Xavier
Nota Interna DEE - PUC/Rio, Outubro de 1990.

Agradecimentos:

Os autores gostariam de expressar seus agradecimentos a:

CNPq e à FAPERJ pelos auxílios concedidos sob forma de bolsas de Iniciação Científica e de Pesquisa.

Todos os alunos e ex alunos que participaram deste projeto:

- Eliseu Monteiro Chaves Filho.
- Átila Leão Flores Xavier.
- Marcus Vinicius Itala Ferreira
- Alexandre Nigri.

COBRA Computadores pela colaboração no fornecimento dos componentes necessários à construção da placa processadora.

Professor Raul Queiroz Feitosa pelas valiosas sugestões incorporadas a este projeto.