ANAIS

2.º SIMPÓSIO SOBRE DESENVOLVIMENTO DE SOFTWARE BÁSICO PARA MICROS

SBC / Sociedade Brasileira de Computação
– Comissão Especial para Linguagens e Sistemas de Programação
IME-USP / Instituto de Matemática e Estatística da Universidade de São Paulo
FAPESP / Fundação de Amparo à Pesquisa do Estado de São Paulo

São Paulo, 8 a 10 de dezembro de 1982

ANAIS

2º SIMPÓSIO SOBRE DESENVOLVIMENTO DE SOFTWARE BÁSICO PARA MICROS

SÃO PAULO, 8 A 10 DE DEZEMBRO DE 1982

SBC/SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
- COMISSÃO ESPECIAL PARA LINGUAGENS E SISTEMAS DE PROGRAMAÇÃO
IME-USP/INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO
FAPESP/FUNDAÇÃO DE AMPARO À PESQUISA DO ESTADO DE SÃO PAULO

COMISSÃO ORGANIZADORA:

V.W. Setzer, Coordenador (IME-USP)
S.W. Song (IME-USP)
W. Ruggiero (EPUSP)
M. Stanton (PUC-RJ)

3.3- Linguagem de Controle da Execução

Para controlar o funcionamento da "máquina", será provido uma linguagem de comando, que permita ao usuário um completo domínio sobre a execução de um microprograma.

A versão inicial da linguagem permitirá:

a) seleção de microprogramas para carga,

b) seleção do ponto de partida para execução de um microprograma, com o controle de parada sendo feito pelo número de referências ou pelo valor registrado em um recurso.

exemplo:

$EXEC 15 $REF micmem=10

inicia a execução de um microprograma a partir da posição 15 da microcmemória e é interrompida quando o número de referencias ao recurso MICMEM atingir 10 vezes.

c) inspeção e modificação do conteúdo de um recurso, permitindo que antes ou após o término da execução de um grupo de instruções de microprograma, o usuário possa ter conhecimento e/ou modificar o conteúdo de um recurso.

exemplos:

i) $DISPLAY alfa,micmem(115..200)

Um Simulador de Micromáquinas

por:

- Crediné Silva de Menezes
  Fundação Universidade do Amazonas
- Arndt von Staa
  Pontifícia Universidade Católica-RJ

Ric, Novembro/1982

1. Introdução

A crescente evolução dos conceitos de linguagens de programação, o crescimento constante do uso do computador na área de controle de processos e a difusão do uso de rede de computadores, são dentre tantos outros, um constante desafio aos projetistas de hardware.

É bem verdade que o surgimento de microcomputadores, fabricados em larga escala, veio dar nova vida às aplicações de controle de processos e rede de computadores, além de motivar o seu crescente uso. Existem entretanto aplicações onde o tempo de resposta é crítico, as quais são melhor resolvidas, e às vezes só resolvidas, com máquinas sob medida.

Procura-se hoje intensamente novas arquiteturas melhor adequadas à execução de programas redigidos em determinadas linguagens procedurais ou não e mesmo utilizando descrição de fluxo de dados como elementos de programação.

Existe pois um campo de pesquisa bastante vasto na área de projeto de hardware, em que pese saber-se da grande evolução da indústria de computadores nos seus trinta anos de vida. Evolução esta, é bem verdade, fortemente marcada pela tecnologia de componentes.

O projeto e construção de computadores, independente da classe de aplicações à que se destina, do porte e da tecnologia de componentes utilizada, é passível de uma divisão em fases bem definidas. Cada fase caracteriza-se por um conjunto de preocupações e decisões a serem tomadas. A seguir apresen-

tamos uma possível divisão em fases, assim como os principais aspectos envolvidos em cada uma delas:

a) Especificação de requisitos

- velocidade de processamento
- capacidade de armazenamento
- objetos a serem manipulados
- formas de manipular objetos

b) Especificação Funcional

- conjunto de instruções

c) Projeto Lógico

- memórias necessárias
- modos de endereçamento
- formato de instruções
- algoritmos para implementar o conjunto de instruções

d) Projeto Físico

- forma de implementar memórias (serviço, controle, armazenamento)
- forma de implementar unidade de controle
- forma de implementar unidade de processamento
- vias de dados

- largura das vias e memórias

c) Implementação

- criação de circuitos
- [illegible] componentes
- codificação de microprogramas
- verificação da implementação de hardware
- verificação da implementação do firmware

Das fases acima citadas, o projeto físico e a implementação, são as que mais sofrem influência do avanço da tecnologia, assim como carecem de melhores ferramentas de auxílio.

Já por volta de 1951, o professor Wilkes preconizava o uso de microprogramação como uma forma racional e confiavel de implementar unidades de controle. Apesar de simples e vantajosa, a idéia teve que esperar alguns anos até que o custo das memórias ROM tornasse o seu uso viável na implementação das memórias de controle. Hoje o processo é utilizado com sucesso pela maioria dos fabricantes, exceto os de microcomputadores, dado a larga escala de integração em que estes são produzidos.

Mais recentemente, a proliferação de blocos pre-construidos -- bit-slices -- veio provocar sensíveis mudanças na implementação de computadores. Estas famílias de blocos -- unidades de processamento, seqüenciadores de microprogramas, processadores de interrupções, entre outros -- em geral são

comercializados com vias de dados de quatro bits, com conexões que permitem cascateá-los para formar novos blocos de vias mais largas.

Em que pesem as facilidades introduzidas por essas duas tecnologias, defronta-se o projetista de hardware com alguns grandes problemas. Destes problemas, podemos destacar:

a) o teste de microprogramas carece da existencia do hardware. Isto inibe o paralelismo de um projeto (hardware versus firmware);

b) a construção de protótipos de hardware, é feita através da fiação de componentes, processo este que dificulta e encarece o projeto;

c) a codificação de microprogramas, é feita em geral em linguagem de baixíssimo nível, metodo este que em geral produz microprogramas de baixa qualidade;

O presente trabalho apresenta os requisitos, a modelagem e o estágio atual de um Simulador de Micromáquinas. Este simulador faz parte do projeto de um Laboratório de Microprogramação do Departamento de Informática da PUC-RJ.

Tal simulador tem como função principal permitir a especificação e realização de micromáquinas em ambiente de software e tem como objetivo contribuir para:

- auxiliar a construção e validação de protótipos em ambiente ameno ao usc, longe de circuitos lógicos, fios, placas e outros apetrechos,

- a criação e teste de microprogramas para máquinas existentes, sem que essa precise ser desalocada de suas tarefas normais,

- facilitar o aprendizado de organização de computadores, rudimentos de microprogramação e algoritmos básicos de um computador.

Queremos ressaltar que o simulador que ora estamos projetando tem como base o conhecimento e a experiência adquiridos durante a construção de uma versão inicial rudimentar. Na primeira versão não se levou em conta facilidades de uso do simulador. A ênfase foi construir um programa flexível, que permitisse a simulação de diferentes micromáquinas. A flexibilidade foi obtida através de um projeto modular e uma linguagem de alto nível, os quais permitem que a alteração para simular uma outra micromáquina seja feita em um curto espaço de tempo.

2. Especificação de Requisitos

O sistema de simulação ora apresentado visa permitir que se especifique e exercite micromáquinas através da execução de microprogramas.

Obviamente o sistema terá de levar em conta que nem sempre esses programas estarão corretos, nem tampouco a especificação da máquina estará correta, ou ainda poderá ser introduzido erro ao passá-la para o sistema.

Para possibilitar que especificações e microprogramas possam ser verificados sem grandes atropelos, e com alto grau de confiabilidade, estabelecemos para o sistema os seguintes requisitos:

a) Facilidade para descrever micromáquinas - O sistema será provido de uma linguagem para especificar micromáquinas, flexível o bastante para permitir a sua utilização no projeto de diferentes micromáquinas. Esta linguagem será tal que exija o mínimo conhecimento da estrutura interna do sistema. O simulador levará em conta a existência de bit-slices e outros dispositivos utilizados na construção de micromáquinas.

b) Facilidade para incorporar novas funções eletrônicas - independente da modelagem e da implementação que se dê ao sistema, este será provido de mecanismo que possibilite ao usuário escrever e utilizar rotinas que implementem novas funções eletrônicas, tais como um multiplexador, um circuito decodificador ou um determinado bit-slice.

c) Facilidade para controlar o funcionamento da "máquina" - O sistema será dotado de uma linguagem que permita ao usuário controlar a execução de um microprograma. Tal linguagem será provida de mecanismo para carregar microprogramas, indicar pontos de parada durante a execução de um microprograma (break-points), mostrar o conteúdo de registradores e memórias, atribuir valor a registradores e memórias, retomar a execução a partir

de um dado endereço de micromemória e monitorar recursos da máquina.

d) Outras - O simulador levará em conta as características físicas dos componentes da micromáquina, para que tenha utilização como instrumento auxiliar na verificação do projeto. Deverá prover estatísticas, para auxiliar a avaliação e verificação do projeto, tais como controle de usos dos recursos(registradores, memórias e vias) e contabilizar o tempo de execução em número de ciclos da máquina.

3. Especificação Funcional

Diversos trabalhos existentes hoje, tratam de apresentar linguagens para descrever máquinas. Essas linguagens permitem uma completa descrição dos componentes da máquina e das funções desempenhadas por elas. Observa-se, entretanto, que pouca ênfase é dada a um ambiente para testes de microprogramas.

Seguindo uma linha diferente, no presente trabalho preocupamo-nos em criar um ambiente para acompanhar o exercício de microprogramas, submetidos a uma máquina especificada.

3.1. Modelagem

O modelo adotado para descrever micromáquinas, terá como base que uma micromáquina é um conjunto de dispositivos -- funções de via, funções eletrônicas e unidades de memória -- que operam sobre determinados recursos -- registradores, partes de registradores, ou vetor de registradores -- funcionando em determinados subciclos e controlados por código de operação oriundo de um determinado recurso (ver fig. 1).

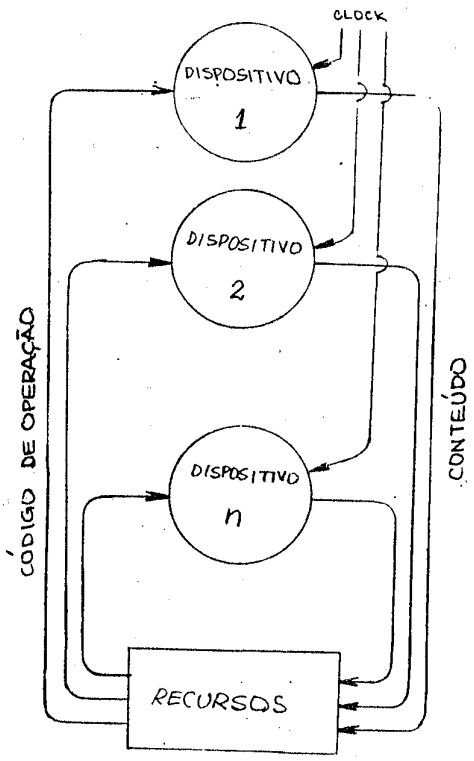


Fig. 1 - Abstração de uma Micromáquina

Funções de via, são aquelas que ligam dois ou mais registradores, cuja finalidade é controlar o fluxo de dado entre eles ("gates", "snifters",...).

Funções eletrônicas usam o conteúdo de registradores, para produzir um novo dado que será transferido para um ou

mais registradores. Podem ser simples como um contador ou complexas como um bit-slice que implementa uma unidade de processamento. Uma função pode inclusive ser dotada de memória interna.

Unidades de memória são vistas, em nosso modelo, como funções eletrônicas com memória interna, que recebem um endereço e recuperam ou armazenam um valor de/em um outro registrador, controladas por um código de operação.

As definições acima permitem-nos uma abstração de qualquer função da máquina como um dispositivo, que usa recursos e possui ou não memória interna.

Um registrador físico pode ser dividido logicamente em partes (registrador lógico). E tanto um como outro serão tratados por registrador.

Algumas funções usam registradores bem determinados na entrada e na saída, outras usam o conteúdo proveniente de um registrador selecionado por uma outra função, e há ainda outras cujo resultado da ativação será armazenado em um ou mais registradores condicionados à seleção de uma outra função. Nos casos de indefinição a função usa como entrada/-saída uma via de dados que a liga com uma outra função.

Apenas nos casos de indefinição acima expostos, sente-se a necessidade de ver uma via como um recurso. A solução adotada consiste em criar pseudo-registradores que servirão de interface entre os dispositivos envolvidos, generalizando desta forma o modelo.

Um dispositivo entra em funcionamento sempre que acionado pelo relógio e os recursos necessários estejam disponíveis. Os recursos podem ser dados operacionais ou de controle (código de operação), sendo que os de controle são em geral oriundos de uma microinstrução.

Para controlar todos os dispositivos, uma microinstrução necessitaria prover um campo para cada um deles. Por questões técnicas e econômicas isto não é conveniente. O que se tem é um conjunto de microinstruções, onde cada tipo de microinstrução controla grupos de dispositivos. Cada tipo de microinstrução é reconhecido por uma configuração de bits.

3.2- Linguagem de Especificação de Micromáquinas

Na linguagem proposta uma máquina é descrita através de seus recursos, dispositivos e tipos de microinstrução.

A seguir apresentamos a linguagem através de exemplos.

3.2.1 - Descrição de Recursos

a) REGISTRADORES E SUAS PARTES - Um registrador é descrito por um nome, uma largura, um valor "default" e suas partes. As partes são descritas por um nome, sua localização dentro do registrador e um valor 'default'. Na descrição de partes é permitido tanto uma descrição hierárquica como descrições sobrepostas, como é visto no exemplo abaixo:

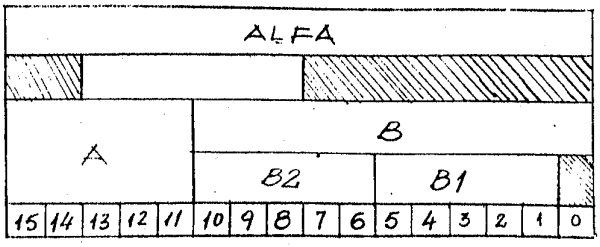
$P A,11,15

$P B,1,10

$P B1,1,5

$P B2,6,10

$P C,8,13



b) VETOR DE REGISTRADORES

Um vetor de registradores será descrito pela largura dos componentes e pela quantidade de elementos. Um elemento de um registrador não pode ser dividido em partes, e nem possui valor 'default'.

$VET MEMORIA,32,8192

descreve um vetor de nome memória, com 8192 elementos e de largura 32.

c) VIA DE DADOS

Vias de dados, no nosso sistema, são pseudo-registradores utilizados como interface entre dispositivos em cascata. São descritas por um nome e uma largura.

$VIA TEMP1,16

descreve uma via de nome TEMP1 com largura 16.

d) EQUIVALENCIA DE REGISTRADORES

Algumas vezes é necessário dar um nome particular a um elemento de um vetor. Por exemplo, em um vetor de registradores setimo poderia ter a função especial de [illegible] "Contador de Instruções". Isto é possivel através da entrada $EQV.

$REG PC,16

$VET VETA,16,15

$EQV PC = VETA(7)

descreve um registrador PC, um vetor VETA e a equivalencia entre PC e o setimo elemento de VETA.

3.2.2 - Dispositivos

A descrição de um dispositivo, envolve a associação deste com uma rotina do usuário que o realizará, a relação dos recursos utilizados pelo dispositivo, e informações sobre tempo de ativação.

Exemplo - Seja um multiplexador(MUX1) implementado pela rotina MUTEX, o qual coloca em uma via de dados (VIA1) o conteúdo de um registrador A ou B, de acordo com o código de seleção oriundo do bit 10 do registrador de microinstruções(MICROIR). O dispositivo entra em funcionamento no primeiro subciclo, sua ordem de ativação é 2 e dispende um subciclo.

```
$REG VIA1,8

$REG A,8

$REG B,8

$REG MICROIR,64

$P SVIA1,10,10

$DISJ MUX1: MUTEX(SVIA1,A,A,VIA1),1,2,1
```

3.2.3- Microinstruções

Cada micromáquina pode possuir um ou mais tipos de microinstrução. Cada tipo de microinstrução é responsável pela ativação de um grupo de dispositivos. A cada tipo de microinstrução está associado um conjunto de bits do MICROIR que as identifica. exemplo

```
$REG MICROIR,16

$MINST PROCESS,1X011X0, GATE1,ALU,GATE2
```

Especifica a microinstrução PROCESS, reconhecida quando os bit bits 15,12 e 11 estiverem ligados e os bits 13 e 10 desligados. A microinstrução PROCESS é responsável pela ativação dos dispositivos GATE1,ALU e GATE2.

3.3- Linguagem de Controle da Execução

Para controlar o funcionamento da "máquina", será provido uma linguagem de comando, que permita ao usuário um completo domínio sobre a execução de um microprograma.

A versão inicial da linguagem permitirá:

a) seleção de microprogramas para carga,

b) seleção do ponto de partida para execução de um microprograma, com o controle de parada sendo feito pelo número de referencias ou pelo valor registrado em um recurso.

exemplo:

$EXEC 15 $REF micmem=10

inicia a execução de um microprograma a partir da posição 15 da micromemória e é interrompida quando o número de referencias ao recurso MICMEM atingir 10 vezes.

c) inspeção e modificação do conteúdo de um recurso, permitindo que antes ou após o término da execução de um grupo de instruções de microprograma, o usuário possa ter conhecimento e/ou modificar o conteúdo de um recurso.

exemplos:

i) $MOSTRA alfa,micmem(115..200)

exibe o conteúdo do registrador ALFA e dos elementos de endereço 115 a 200 do vetor de registradores MICMEM.

ii) $ALTERE alfa=beta

atribui ao recurso ALFA o conteúdo do registrador BETA.

Vale observar que a divisão de um recurso em partes, combinado com o comando de modificação, permite que sejam escritos microprogramas simbólicos durante uma sessão do simulador.

d) Monitoramento de recursos. Comandos através dos quais o usuário pode manisfestar o seu desejo de acompanhar as referencias a um dado recurso, assim como desativar o monitoramento quando lhe for conveniente. Compete ao sistema interceptar as referencias a esses recursos, e exibir a instrução que lhe provocou a referencia.

exemplo:

$ARASTRO mem01,beta,stack

ativa o rastreamento sobre os recursos MEM01, BETA e STACK.

4. Estágio Atual

O projeto do simulador encontra-se agora na modelagem física, tendo o início de sua implementação marcado para meados de novembro/82 e com previsão de encerramento para final de janeiro/83.

O sistema foi dividido basicamente em dois programas, conforme fluxo de dados do sistema (fig. 2). Estes programas são:

a) Processador de Descrição

Este programa é responsável pela análise sintática da especificação da micromáquina, geração das tabelas descritoras de recursos e demais componentes, e geração do "módulo de ativação de dispositivos" contido no programa "executor da micromáquina".

b) Executor da Micromáquina

Este programa é o responsável pela submissão de microprogramas à micromáquina especificada. Está organizado em cinco níveis lógicos (fig. 3), composto pelos seguintes módulos:

b.1) módulo principal responsável pelo controle da simulação.

b.2) módulo que obtem comandos de controle válidos, sendo portanto responsável pela obtenção e análise dos comando de controle da execução.

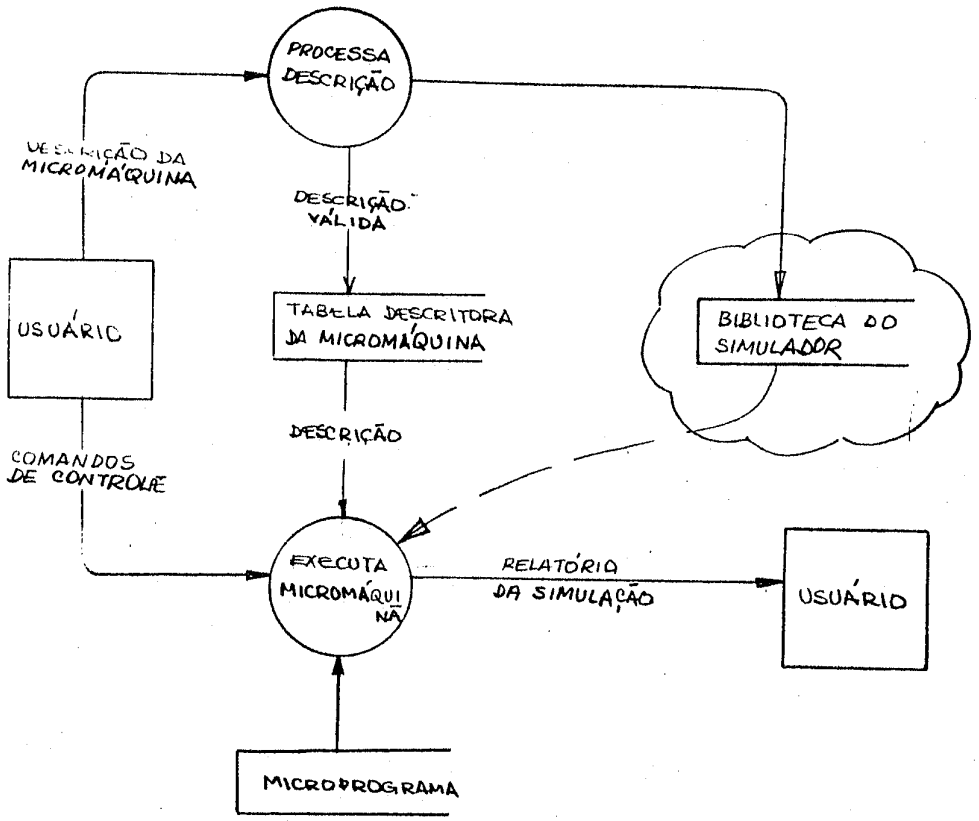


Fig. 2 - Fluxo de Dados do Simulador

b.3) módulos responsáveis pela execução dos comandos do usuário, sendo que o módulo que realiza o comando de execução funciona como um segundo supervisor, controlando a execução de instruções e avaliando as condições de parada.

c) rotinas do usuário, responsáveis pela implementação de "funções" da máquina a simular.

d) rotinas do sistema que manipulam o tipo abstrato "recursos", responsável pela obtenção e modificação de atributos de recursos da máquina

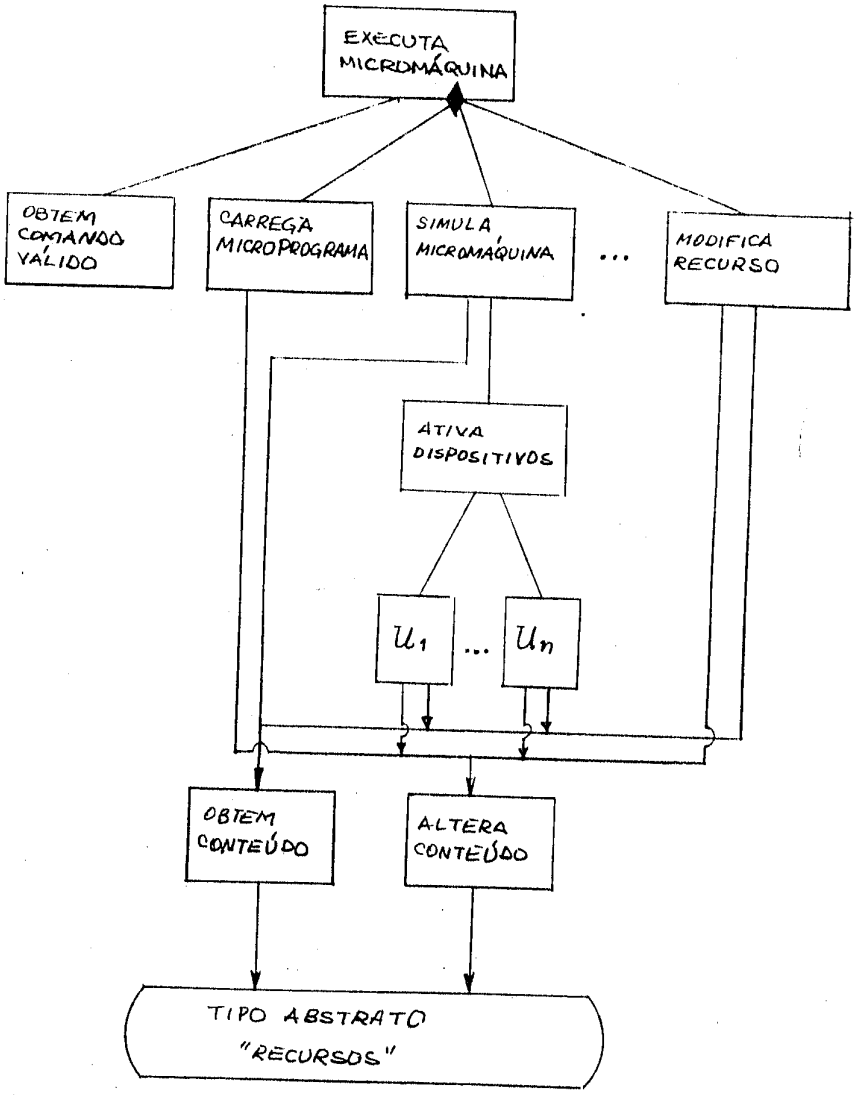


Fig. 3 - Diagrama de Módulos do "executor"

A manipulação dos recursos da máquina foi encapsulada em um tipo abstrato de dados de dois níveis lógicos denominado "recursos". O primeiro nível é visível apenas aos módulos do "simulador" propriamente dito, composto por operações que permitem instalar e consultar atributos de um recurso. O segundo nível, acessível tanto ao simulador quanto às rotinas

do usuário é composto por operações que permitem consultar e modificar o conteúdo de um "recurso".

A implementação física de "recursos" será feita através de um esquema de paginação que possibilitará que só seja dispendida a quantidade de memória necessária e suficiente para uma dada micromáquina. Cada página é composta de um número fixo de blocos de TAMPAL bits, blocos estes responsáveis pela representação de registradores, vias de dados e vetores de registrador, sendo estes últimos encarregados de representar as várias memórias endereçaveis da microdmáquina.

Com o objetivo de obter melhor performance, cada registrador será associado a um bloco completo, a menos que sua largura seja maior que TAMPAL bits, quando então se estenderá por blocos consecutivos.

Um dos principais critérios de qualidade que norteiam a implementação de um software desta classe, é sem duvida alguma a portatilidade. Evidentemente que portatilidade, em geral, é uma qualidade conflitante com eficiencia. É de conhecimento público entretanto que eficiência pode ser ajustada após uma primeira implementação, principalmente quando esta está relacionada à potência da linguagem de programação escolhida.

Com base nessas argumentações a linguagem Pascal foi escolhida para implementação do simulador, embora saber-se que não é a maneira mais adequada de manipular bits. Isto poderá ser corrigido a posteriori através da implementação de "recursos" em linguagem assembler da máquina onde se instalará o software.

5. Conclusão

Neste artigo apresentamos a motivação para o projeto e construção de um sistema Simulador de Micromáquinas.

Abordamos também a especificação funcional, o projeto lógico e o projeto físico de um software flexível para tal aplicação.

Na especificação funcional tratamos da modelagem de uma micromáquina, de uma linguagem para descrever micromáquinas e uma outra para "operar" o simulador. As duas linguagens, apesar de poderosas, podem parecer pouco elegantes. Justifica-se a escolha destas através da simplicidade, a qual facilitará tanto a implementação como futuras expansões do sistema. Esta facilidade é embasada na sintaxe das linguagens que usam um caracter de escape, e pela divisão modular(funcional) dada ao sistema.

O projeto lógico do sistema foi esboçado, e considerações foram feitas em torno da implementação física de "recursos" e da escolha da linguagem de programação a adotar.

Para que se tenha uma visão completa da linguagem de descrição, apresentamos em apêndice a descrição completa de uma máquina simples.

6. Bibliografia

1. Anastas, M.S.; Vaughn, R.F. - Direct Architectural implementation of a Requeriments Oriented Computing Structure, SIGMICRO Newsletter, vol.10, No.4, dez/79

2. Cavouras, J.C.; Davist, A.H. - Simulation Tools in Computer System Design Mthodologies; The Computer Journal, vol.24, No.1, 1981

3. Drongowski, P.J.; Rose, C.W. - Application of Hardware Description Languages to Microprogramming: Method, Pratice, and Limitations; SIGMICRO Newsletters, vol.10, No.4, dez/79

4. Iliffe, J.K. - Advanced Computer Design, USA, Prentice Hall International Inc., 1982

5. Katzan, H.Jr - Microprogramming Primer, USA, McGraw Hill Company, 1977

6. Langdon, G.G.Jr; Fregni, E. - Projeto de Computadores Digitais, BRASIL, Edgard Blucher Ltda, 1974

7. Mezzalona, M.; Prinetto, P. - Design and Implementation of a Flexible and Interactive Microprogram Simulator, SIGMICRO Newsletter, vol.10, No.4, dez/79

8. Myers, G.J. - Advances in Computers Architecture, USA, J.Wiley, 1978

9. Myers, G.J. - Digital System Design With Bit-Slice Logic, USA, J.Wiley, 19[illegible]

10. Persson, M. - Design of a Microprogram Generator for the VARIAN V73; SIGMICRO Newsletter, vol.8, No.4, dez/77

11. Schleimer, S.; Myers, W.J. - Experience with a High Level Micromachine Simulator; SIGMICRO Newsletter, vol.10, No.4, dez/79

12. Skordalakis, E. - LGCAS as a Microprogram Development Tool, SIGMICRO Newsletter, vol.11, No.1, mar/80

13. Sommerville, J.F. - Towards Machine-independent Microprogramming; Euromicro Journal, vol.5, No.4, dez/79

14. Sutphen, S.F. - Teaching and Research experiences with an Emulation Laboratory, AFIPS, Conference Proceedings, NCC, vol.48, 197 1979

15. Tamura, E.; Tokoro, M. - Hierarchical Microprogram Generating System, SIGMICRO Newsletter, vol.10, No.4, dez/79

16. Tannenbaum, A.S. - Structured Computer Organization, USA, Prentice Hall, 1976

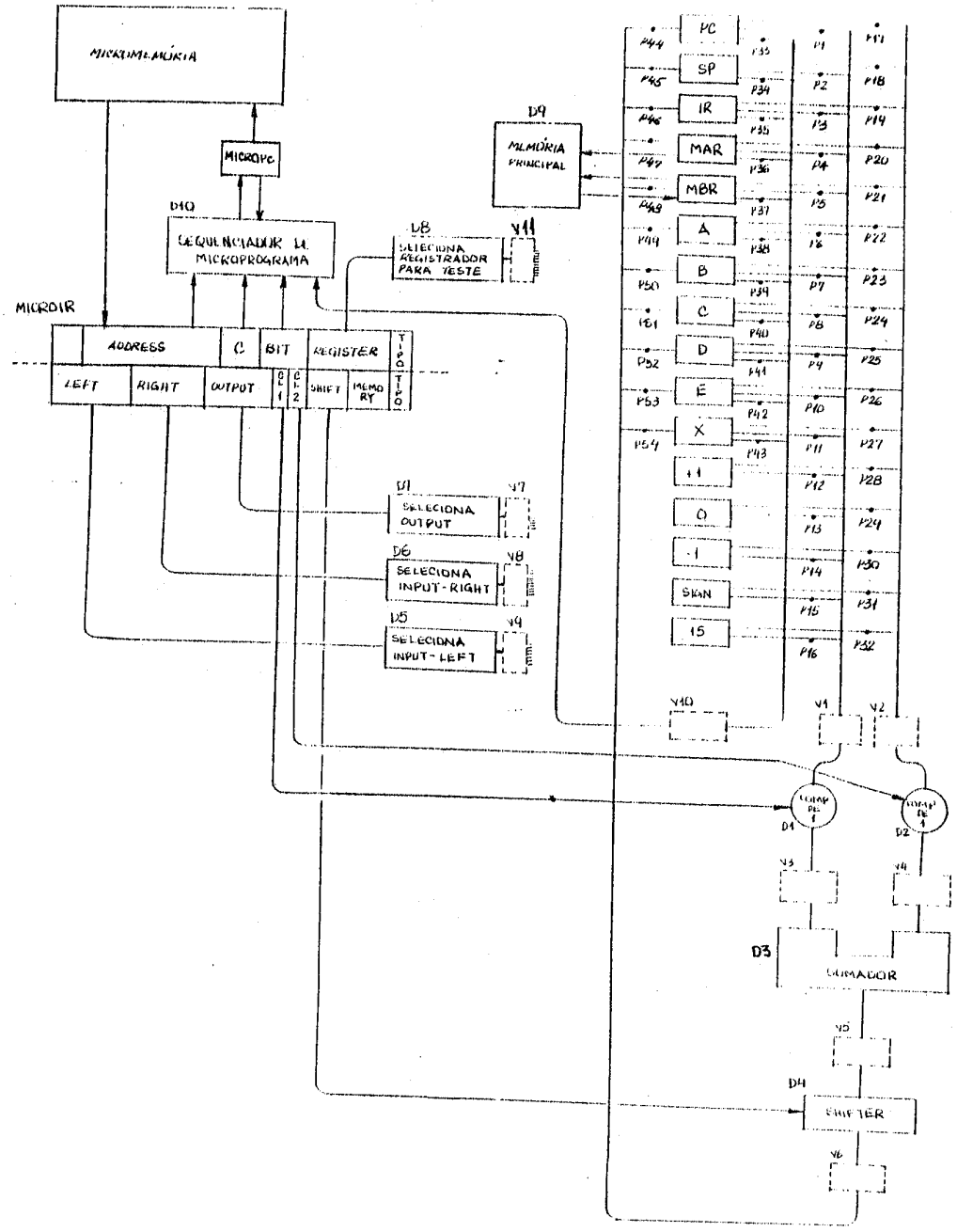
17. Tracz, W.J. - Microprogramable Computer Architecture a design Considerations; SIGMICRO Newsletter, vol.11, No.4, mar/80

18. Wilkes, M.V. - Ten Years and More of Microprogramming; SIGMICRO Newsletter, vol.8, No.4, dez/77

APÊNDICE

A seguir apresentamos o diagrama de uma micromáquina e a sua especificação na linguagem de descrição de micromáquinas. O exemplo apresentado tem como base a micromáquina apresentada em /10/, que apesar de simples permite um exercício completo da linguagem de especificação.

Após a especificação dos recursos e dispositivos apresenta-se também as rotinas 'GATE' e 'SOMADOR', as quais realizam alguns dispositivos da máquina especificada.


MICROMEMÓRIA
MICROPC
D10
SEQUENCIADOR DE MICROPROGRAMA
D8
SELECIONA REGISTRADOR PARA TESTE
V11
D9
MEMÓRIA PRINCIPAL
MICRORIR
ADDRESS
C
BIT
REGISTER
TIPO
LEFT
RIGHT
OUTPUT
C1
C2
SHIFT
MEMORY
TIPO
D1
SELECIONA OUTPUT
V7
D6
SELECIONA INPUT-RIGHT
V8
D5
SELECIONA INPUT-LEFT
V9
PC
SP
IR
MAR
MBR
A
B
C
D
E
X
+1
0
-1
SKIN
15
V10
V1
V2
D4
D2
V3
V4
D3
SOMADOR
V5
D4
SHIFTER
V6

```
$COM
$COM        Descriçao de Registradores
$COM
$REG PC,13
$REG SP,13
$REG IR,16
$REG MAR,13
$REG MBR,16
$REG A,16
$REG B,16
$REG C,16
$REG D,16
$REG E,16
$REG MENOSUM,16, X'FFFF'
$REG MAISUM,16, X'0001'
$REG QUINZE,16 X'0001'
$REG SIGN,16, X'FFFF'
$REG $MICROPC,8
$REG $MICROIR,40
  $P LEFT,18,15
  $P RIGHT,14,11
  $P OUTPUT,10,7
  $P COMPL1,9,6
  $P COMPL2,5,5
  $P SHIFT,4,3
  $P MEMORY,2,1
  $P TIPO, 0,0
  $P ADDRESS,17,10
  $P C,9,9
  $P BIT.8.5
```

```
   $P  REGISTER,4,1
 $COM
 $COM          Descrição De Memórias
 $COM
 $VET $MICMEM,40,256
 $VET MAINMEM,16,8192
 $COM
 $COM          Descrição De Vias de Dados
 $COM
 $VIA V1,16
 $VIA V2,16
 $VIA V3,16
 $VIA V4,16
 $VIA V5,16
 $VIA V6,16
 $VIA V7,11
   $P SP44,0,0
   $P SP45,1,1
          .
          .
          .
   $P SP54,10,10
 $VIA V9,16
   $P SP1,0,0
   $P SP2,1,1
          .
          .
          .
   $P SP16,15,15
 $VIA V8,16
```

```
   $P SP17,0,0
   $P SP18,1,1
          .
          .
          .
   $P SP32,15,15
$VIA V11,11
   $P SP33,1,1
   $P SP34,1,1
          .
          .
          .
   $P SP43,10,10
$COM
$COM        Descrição de Dispositivos
$COM
$DISP D1:COMPLMT(V1,V3, COMPL1),1,1,3
$DISP D2:COMPLMT(V2,V4, COMPL2),1,1,3
$DISP D3:SOMADOR(V3,V4,V5),2,1,1
$DISP D4:SHIFTER(V5,V6,SHIFT),3,1,1
$DISP D5:DECOD  (LEFT,V9),1,1,1
$DISP D6:DECOD  (RIGHT,V8),1,1,1
$DISP D7:DECOD  (OUTPUT,V7),3,1,1
$DISP D8:DECOD  (REGISTER,V11),1,1,1
$DISP D9:MEMORIA(MAR,MBR,MEMORY,MAINMEM),4,1,X
$DISP D10:SEQUENCIADOR(ADDRESS,C,BIT,V10,MICROPC),3,1,2
$DISP P1:GATE(PC,V1,SP1),1,1,2
```

```
$DISP P32:GATE(QUINZE,V2,SP32),1,1,2
$DISP P33:GATE(PC,V10,SP33),3,1,1
                .
                .
                .
$DISP P43:GATE(X,V10,SP43),3,1,1
$COM
$COM        Descriçao de Microinstruçoes
$COM
$MINST PROCESSA: D1,D2,D3,D4,D5,D6,D7,D9,D10
$MINST TESTE   : D8,D10

$END


PROCEDURE GATE(R1,R2,PORTA)
BEGIN
   IF GET(PORTA) = TRUE
      THEN
         TRANSFER(R1,R2)
END


PROCEDURE SOMADOR(OP1,OP2,RESULT)
VAR  R3: Registrador;
BEGIN
     R1:= GET(OP1);
     R2:= GET(OP2);
     R3:= ADD(R1,R2);
     PUT(RESULT,R3
END
```