FL 1077
NFI 87/154

KAPA - Uma Linguagem para a Descrição de Hardware do Nível de Transferência entre Registradores

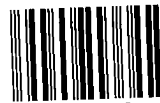
por

Flávio Rech Wagner

RP nº 68 ABRIL/1987

Nota técnica do projeto "Banco de Dados e Ferramentas para CAD de Sistemas Digitais

Trabalho realizado com o apoio do CNPq



UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Osvaldo Aranha, 99
90.210-Porto Alegre-RS-Brasil
Telex (051) 2680 Tel. (0512) 21.8499

Endereço para Correspondência:
UFRGS/CPGCC/Biblioteca
Caixa Postal 1501
90.001-Porto Alegre-RS-Brasil

UFRGS
BIBLIOTECA
CPD/PGCC

Comissão Editorial: José Palazzo Moreira de Oliveira
Carla Maria Dal Sasso Freitas

UFRGS

Reitor: Prof. FRANCISCO FERRAZ

Pró-Reitor de Pesquisa e Pós-Graduação: Prof. HÉLGIO TRINDADE

Coordenador do CPGCC: Prof. ROBERTO TOM PRICE

Comissão Coordenadora do CPGCC:

Prof. CLESIO SARAIVA DOS SANTOS
Prof. DALTRO JOSÉ NUNES
Prof. DANTE AUGUSTO COUTO BARONE
Prof. FLÁVIO RECH WAGNER
Prof. PAULO ALBERTO DE AZEREDO
Prof. ROBERTO TOM PRICE

Bibliotecária CPGCC/CPD: MARGARIDA BUCHMANN

U F R G S
BIBLIOTECA
CPD/PGCC

Linguagens: Descrição: Hardware
KAPA

U F R G S
CPD - PGCC
BIBLIOTECA

N.o CHAMADA: FL1077		N.o REG.: 31505 DATA: 11 / 05 / 87
ORIGEM: D	DATA: 15 / 4 / 87	PREÇO: CZ$ 150,00
FUNDO: CPD/PGCC	FORN.: PGCC	

RESUMO

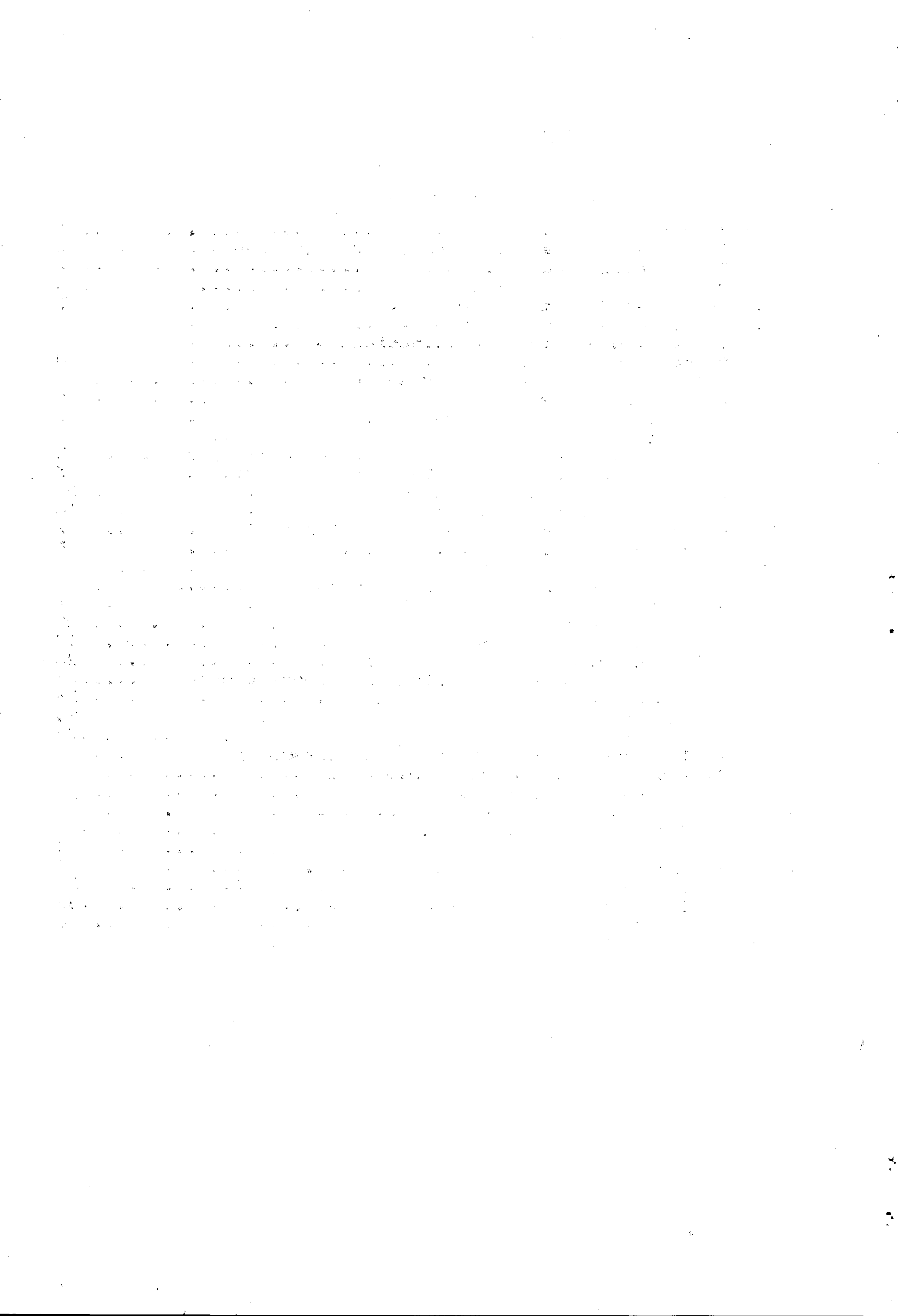
Este relatório apresenta a linguagem KAPA, destinada à descrição de sistemas digitais no nível de transferência entre registradores (RT). Esta linguagem tem formas textual e gráfica de representação que são inteiramente equivalentes entre si. As ferramentas a ela associadas (compilador, editor gráfico e simulador) fazem parte de AMPLO, um ambiente integrado de projeto de sistemas digitais auxiliado por computador.

PALAVRAS-CHAVE: linguagens de descrição de hardware, nível de transferência de registradores

ABSTRACT

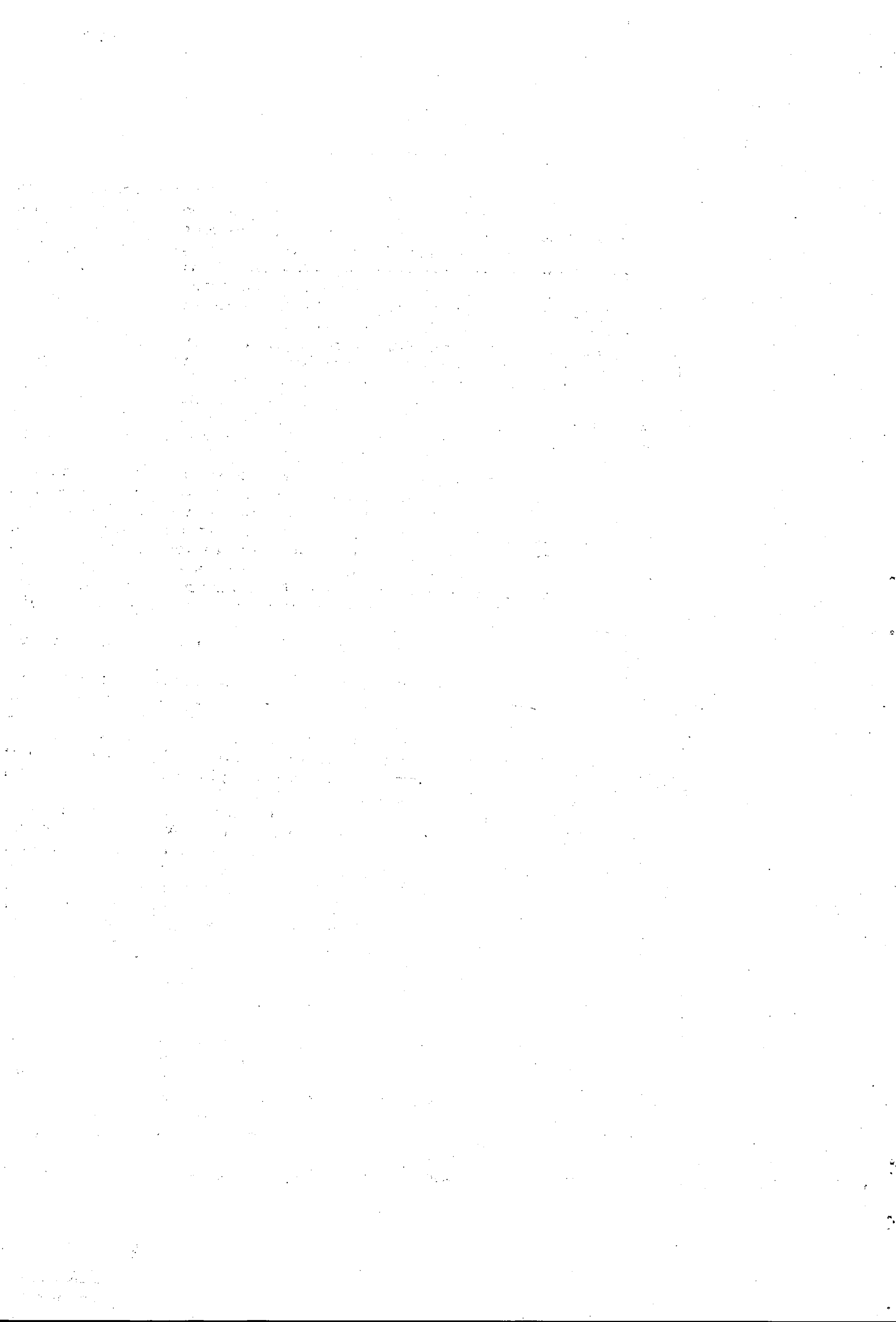
This report introduces the KAPA language, which allows the description of digital systems at the register transfer (RT) level. This language has textual and graphical forms that are completely equivalent. The associated design tools (compiler, graphical editor and simulator) are part of AMPLO, an integrated environment for the computer aided logical design of digital systems.

KEYWORDS: hardware description languages, register-transfer level



SUMARIO

1. Introdução 1
2. Organização geral da descrição de agências 2
   2.1 Designação da agência 2
   2.2 Descrição da interface 3
3. Descrição do comportamento 3
4. Declarações de portadores 4
5. Declaração de sinais combinacionais 7
6. Operadores 8
   6.1 Operadores de transformação 8
   6.2 Operador de concatenação 10
   6.3 Operadores compostos 11
   6.4 Operadores de comparação 12
   6.5 Operadores de codificação e decodificação 12
   6.6 Operadores de multiplexação e demultiplexação 13
   6.7 Operadores de condicionamento 15
   6.8 Operadores temporais 16
7. Atribuições e acessos parciais a terminais 17
8. Barramentos 19
9. Carga de registradores 20
   9.1 Registradores sensíveis à borda 20
   9.2 Registradores master-slave 21
   9.3 Registradores latch 22
   9.4 Controle combinado 23
   9.5 Condicionamento do controle 23
10. Sub-registradores e registradores concatenados 24
   10.1 Carga de sub-registradores 24
   10.2 Leitura de sub-registradores 26
   10.3 Carga de registradores concatenados 27
   10.4 Leitura de registradores concatenados 28
11. Arrays de registradores e memórias 29
   11.1 Carga de arrays 29
   11.2 Leitura de arrays 31
   11.3 Memórias 33
12. Constantes 34
13. Sinais de relógio 35
   13.1 Sinais de relógio primário 36
   13.2 Sinais de relógio secundário 36
14. Agências síncronas e assíncronas 38
Referências bibliográficas 39



1. INTRODUÇAO

O sistema AMPLO [WAG86c,WAG87c] é um ambiente integrado para o projeto lógico de sistemas digitais em desenvolvimento na UFRGS. Este ambiente possui ferramentas que permitem a descrição e validação de sistemas digitais em três níveis de projeto: sistema, transferência entre registradores (RT) e portas lógicas. Para cada um destes níveis existe uma linguagem de descrição de sistemas, respectivamente LASSO [BOR79], KAPA, que é descrita neste relatório, e NILO [WAG87b].

Sistemas são descritos de forma modular e hierarquizada como redes de agências [WAG84a,b], através da linguagem REDES [WAG87a]. Uma agência pode ser descrita de forma puramente estrutural, como uma composição de outras agências, utilizando-se a linguagem REDES, ou através de um comportamento especificado em uma das linguagens LASSO, KAPA ou NILO. Esta metodologia de projeto é apresentada em [WAG86a,c,WAG87c].

Todas as quatro linguagens suportadas pelo ambiente possuem formas gráfica e textual que são completamente equivalentes entre si [WAG86b]. Isto permite que a descrição de uma agência possa ser feita de forma textual, via editor de textos convencional e compilador, ou de forma gráfica, via um editor especializado para a linguagem. Tanto o compilador como o editor gráfico geram uma mesma estrutura de dados interna, que é independente da forma de entrada, exceto pelas informações geométricas manipuladas pelo editor. Esta representação interna é armazenada em uma base de dados unificada, de onde pode ser recuperada para a construção de modelos simuláveis.

A linguagem KAPA permite a descrição de sistemas como uma interconexão de blocos reais de hardware do nível RT. Ela pode ser classificada, portanto, como uma linguagem "estrutural", segundo a definição de Barbacci [BAR75], em oposição ás linguagens ditas "funcionais" ou "comportamentais", tais como ISPS [BAR81], que permitem descrições da função do sistema sem entrar nos detalhes de implementação.

KAPA é uma linguagem fortemente baseada em KARL [HAR77]. Várias construções elementares existentes em KARL não foram implementadas em KAPA por não corresponderem a blocos de hardware comumente utilizáveis. Além disto, muitas combinações de construções, que são possíveis em KARL, não o são em KAPA para que se tenha um mapeamento não ambíguo entre as formas gráfica e textual. Assim, a cada construção gráfica em KAPA corresponde uma e somente uma construção textual, e vice-versa. Também foi alterada a forma gráfica de algumas construções de KARL de forma a tornar mais clara a sua semântica, evitando-se uma possível ambiguidade na interpretação destas construções.

Uma diferença importante entre KAPA e KARL diz respeito à integração de KAPA no ambiente AMPLO, que impõe a necessidade de descrições modulares de sistemas como redes de agências.

Este relatório não irá apresentar KAPA a partir das restrições e modificações feitas em KARL. Ele é inteiramente auto-contido, de modo que o leitor nele encontrará a especificação de todas as construções disponíveis na linguagem, sem precisar fazer referências a outros textos.

Este relatório apresenta a sintaxe completa de todas as

UFRGS
BIBLIOTECA

construções da forma textual de KAPA. A forma gráfica da linguagem é apresentada de maneira mais informal. A especificação formal das operações gráficas necessárias à descrição de agências é objeto de outro relatório.

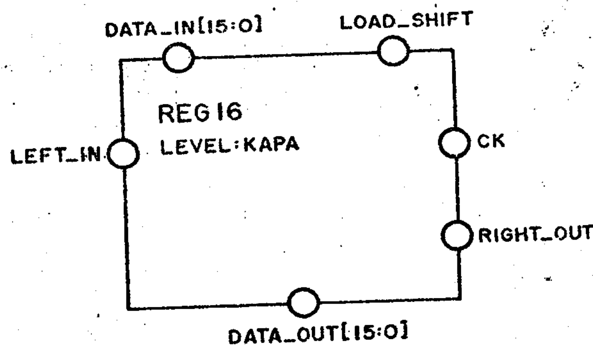
2. ORGANIZAÇAO GERAL DA DESCRIÇAO DE AGENCIAS

Uma agência é descrita conforme mostrado abaixo. A descrição contém quatro partes:

- identificação da agência;
- designação do nível de descrição;
- descrição da interface da agência, e
- descrição do comportamento da agência.

Esta última parte, introduzida pela palavra-chave behavior, é uma sequência de declarações e comandos que descrevem a agência como uma interconexão de primitivas comportamentais do nível RT. Estas primitivas são apresentadas a partir do capítulo 3. As 3 primeiras partes da descrição são comuns a todas as linguagens de AMPLO, seguindo em todas elas as mesmas sintaxe e semântica [WAG87a].

```
agency  REG16.1.1
level  =  KAPA
interface
     in  DATA_IN [15:Ø], LEFT_IN, LOAD_SHIFT : terminal;
         CK : clock;
     out  DATA_OUT [15:Ø], RIGHT_OUT : terminal;
behavior
     !
end;
```



2.1 Designação da agência

A descrição de uma agência é iniciada por

```
agency  <nome-agência> . <nro-alternativa> . <nro-versão>
level = KAPA
```

Esta descrição criará na base de dados de AMPLO uma nova versão para o tipo de agência designado por

<nome-agência> . <nro-alternativa>.

Se esta for a primeira versão para esta alternativa, a alternativa também será criada. Versões de projeto de uma mesma alternativa têm a mesma definição para a interface da agência. Tipos de agências correspondem portanto ás alternativas de projeto de agências definidas na base de dados. Ocorrências diversas destes tipos podem ser utilizadas na definição de novos tipos de agências através da construção use da linguagem REDES.

2.2 Descrição da interface

A palavra-chave interface introduz a definição dos sinais de interface da agência. Estes sinais podem ser de entrada (in), saida (out) ou bidirecionais (inout). Cada sinal è descrito por

<nome_sinal> <dimensão_sinal> : <tipo_sinal>.

<Dimensão_sinal> è a especificação da largura em bits do sinal, dada através de

[n1 : n2],

onde n1 è a designação do bit mais significativo do sinal, e n2 a do bit menos significativo. No caso de sinais com largura de apenas 1 bit, a dimensão não precisa ser especificada.

<Tipo_sinal> è a designação de um dos tipos de dados definidos na linguagem. A tabela a seguir mostra quais tipos de dados são aceitos para sinais de entrada, saida e bidirecionais.

sinais de entrada	terminal, bus, clock
sinais de saida	terminal, clock
sinais bidirecionais	bus

Um sinal de tipo "clock" sò è aceito como saida de uma agência assincrona (cf. capitulo 14).

3. DESCRIÇAO DO COMPORTAMENTO

Sistemas digitais descritos no nível RT têm elementos de dois tipos diferentes:

- elementos combinacionais, tais como somadores, multiplexadores, decodificadores, etc., e
- elementos com memòria, tais como registradores e memòrias, designados aqui genericamente como "portadores".

Saidas de elementos combinacionais e portadores são designados como "sinais" ao longo deste relatòrio.

A descrição do comportamento de uma agência em KAPA, introduzida por behavior, è composta por duas partes:

- declarações, e

UFRGS
BIBLIOTECA

- comandos.

A primeira parte serve à declaração dos portadores e dos sinais de saída dos elementos combinacionais. Os comandos são funções primitivas da linguagem, designadas como "operadores", ou combinações destas funções, e servem à especificação

- da lógica responsável pela geração de cada sinal combinacional,
- do tipo de carga dos portadores, e
- do conteúdo a ser carregado em cada portador.

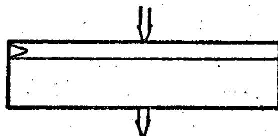
4. DECLARAÇÕES DE PORTADORES

Portadores são declarados através das seguintes palavras-chave:

<u>register</u>,
<u>subregister</u>,
<u>casregister</u>,
<u>array-register</u>,
<u>memory</u>,
<u>constant</u>, e
<u>array-constant</u>.

Um registrador é declarado como

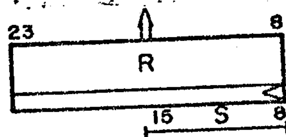
<u>register</u> <nome_registrador> <dimensão_registrador>;



A dimensão de um registrador é declarada da mesma forma que a dimensão de um sinal da interface da agência, tal como foi visto no item 2.1. A dimensão não precisa ser declarada no caso de registradores de 1 bit. Graficamente, um registrador é um retângulo, ao qual chega um vetor de dados de mesma dimensão que o registrador, e do qual sai um vetor também de mesma dimensão. Vetores de dados de dimensão maior do que um são representados por uma barra dupla, enquanto que vetores de largura igual a 1 são representados por uma barra simples. Um registrador mostra continuamente seu conteúdo no vetor de saída (o reflexo na saída de uma alteração no conteúdo do registrador pode no entanto ser retardado, no caso de registradores tipo "master-slave", como será visto mais adiante).

A declaração de sub-registradores tem por objetivo a designação, através de um sinônimo, de parte de um registrador já previamente declarado. No exemplo

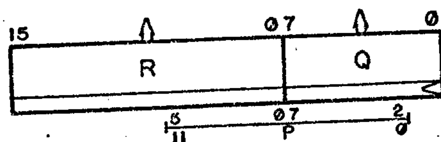
<u>register</u> R [23:8];
<u>subregister</u> R [**S**] = R [15:8];



os 8 bits menos significativos do registrador R são designados por S, e identificados implicitamente como sendo os bits 7 até Ø de S. Comandos da linguagem podem se referir indistintamente a S ou R [15:8], ou mesmo a quaisquer sub-vetores de S e R [15:8].

A declaração de registradores concatenados tem por objetivo a designação, através de um sinônimo, de uma justaposição de dois ou mais registradores. No exemplo

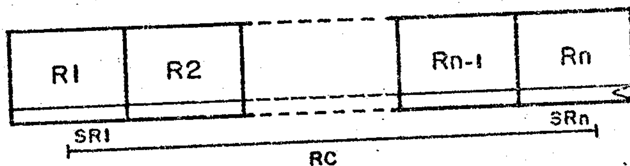
```
register  R [15:Ø], Q [7:Ø];
casregister  P [11:Ø] = R [5:Ø] : Q [7:2];
```



a concatenação dos 6 bits menos significativos de R com os 6 bits mais significativos de Q tem a designação alternativa P [11:Ø]. Comandos da linguagem podem se referir indistintamente a qualquer destas alternativas, tanto ao vetor de 12 bits como um todo como a um sub-vetor dele. As referências Q [7:6] e P [5:4] são portanto equivalentes, assim como R [5:3] e P [11:9].

A declaração de registradores concatenados, como se vê, pode envolver sub-registradores, tenham estes sido declarados explicitamente como tal ou não. O uso de sub-registradores para formar um registrador concatenado RC apresenta no entanto as seguintes restrições:

1) Sub-registradores podem aparecer apenas nas extremidades de RC. No exemplo abaixo, os registradores R2,...,Rn-1 utilizados para formar RC não podem ser sub-registradores.



2) O sub-registrador R1 [SR1] na extremidade mais significativa de RC deve necessariamente envolver os bits menos significativos de R1.

3) O sub-registrador Rn [SRn] na extremidade menos significativa de RC deve necessariamente envolver os bits mais significativos de Rn.

Exemplos inválidos de declarações de registradores concatenados:

register R1 [15:0], R2 [15:0], R3 [15:0];
casregister RX [37:0] = R1 : R2 [15:10] : R3;
* erro: o registrador do meio não pode ser um sub-registrador *
casregister RX [37:0] = R1 [15:10] : R2 : R3;
* erro: o sub-registrador na extremidade mais significativa de RX não alcança o LSB de R1 *
casregister RX [37:0] = R1 : R2 : R3 [5:0];
* erro: o sub-registrador na extremidade menos significativa de RX não alcança o MSB de R3 *

Um array de registradores è declarado como

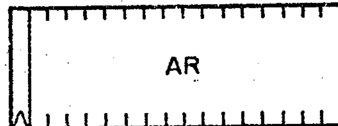
array-register <designação_array> <dimensão_array>;

A dimensão de um array é declarada por

[m1 : m2; n1 : n2],

onde m1 è o índice do primeiro registrador do array e m2 è o índice do último registrador. Todos os registradores do array têm a mesma largura em bits, sendo n1 a designação do bit mais significativo de cada registrador e n2 a designação do bit menos significativo. No exemplo

array-register AR [15:0;7:0],



declara-se um array de 16 registradores de 8 bits cada. Comandos posteriores podem fazer referência ao array como um todo ou a um registradór selecionado no array. A seleção de registradores, o tipo de carga e o acesso em leitura de um array de registradores serão vistos no capítulo 11.

Memórias são tipos especiais de arrays de registradores, sobre os quais faz-se uma série de restrições quanto á seleção de registradores, o tipo de carga e a leitura, tal como será visto posteriormente. Uma memória è declarada por

memory <nome_memória> <cimensão_memória>;

<Dimensão_memória> segue a mesma sintaxe que a declaração da dimensão de um array de registradores.

A declaração

constant <nome_constante> <dimensão_constante> = <valor>;

cria um vetor binário com a designação e largura especificadas e com um valor constante também especificado quando da declaração. Este vetor pode ser referenciado apenas em leitura em comandos posteriores. E impossível atribuir-se a ele um valor

dinamicamente. Valores podem ser especificados em

binário (valor precedido de '),

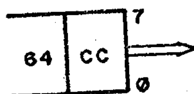
octal (valor precedido de "),

decimal, ou

hexadecimal (valor precedido de #).

O valor associado a uma constante deve ser consistente com a largura em bits para ela declarada. No exemplo

<u>constant</u> CC [7:Ø] = 'Ø1ØØØØØØ;



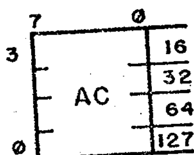
está sendo criado um vetor de 8 bits com valor constante igual a 64 decimal.

A declaração

<u>array-constant</u> <nome_array> <dimensão_array> = <valor_array>;

cria um array de vetores binários constantes, todos com a mesma largura em bits, mas cada um podendo assumir um valor diferente. O valor do array é então uma lista de valores separados por vírgulas, um para cada posição do array, seguindo a notação já referida anteriormente. A dimensão do array de constantes é declarada da mesma forma que a dimensão de um array de registradores. No exemplo

<u>array-constant</u> AC [3:Ø;7:Ø] = 16,32,64,127;



está se declarando um array composto por quatro vetores constantes de 8 bits cada, contendo os valores 16 (constante de índice 3 no array), 32, 64 e 127 (constante de índice Ø no array).

A seleção de uma constante de um array será vista no capítulo 12.

5. DECLARAÇÃO DE SINAIS COMBINACIONAIS

Sinais combinacionais são declarados através das palavras-chave

<u>terminal</u>,
<u>subterminal</u>,
<u>bus</u>, e
<u>clock</u>.

BIBLIOTECA
CPD/PGCC

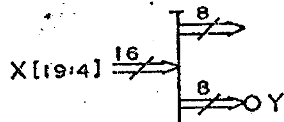
As declarações _clock_ e _bus_ serão vistas mais adiante neste relatório. A declaração

terminal <nome_sinal> <dimensão_sinal>;

identifica e dá a dimensão de um sinal gerado a partir de uma lógica combinacional. A atribuição de valor a um terminal é feita na parte de comandos da descrição da agência.

A declaração de sub-terminais tem por objetivo a identificação de parte de um terminal através de um sinônimo. No exemplo

terminal X [19:4];
subterminal X [Y] = X [11:4];



os 8 bits menos significativos do terminal X são identificados pelo nome alternativo Y. Implicitamente eles serão os bits 7 até Ø do vetor Y.

6. OPERADORES

Operadores combinam sinais entre si, gerando novos sinais. Eles podem ser classificados nos seguintes tipos:

- **operadores de transformação**, que geram um novo vetor com mesma dimensão que o(s) vetor(es) fonte para a operação;
- **operadores de comparação**, que geram um sinal de 1 bit a partir da comparação entre dois vetores de n bits;
- **operador de concatenação**, que gera um vetor de m+n bits de largura a partir da justaposição de um vetor de m bits a outro de n bits de largura;
- operadores de codificação e decodificação;
- operadores de multiplexação e demultiplexação;
- **operadores de condicionamento**, que condicionam a geração de um sinal ao valor de um sinal de controle;
- **operadores temporais**, que geram sinais defasados no tempo em relação aos sinais originais.

6.1 Operadores de transformação

Operadores binários de transformação geram um sinal de n bits a partir de dois sinais, que devem necessariamente ter sido declarados com a mesma dimensão de n bits. Graficamente, estes operadores são representados na forma de uma unidade aritmética e lógica. Os sinais de entrada para a operação podem ser de quaisquer tipos, sejam saídas de portadores ou sinais combinacionais. O sinal de saída do operador deve necessariamente ter sido declarado como tipo _terminal_ ou _sub-terminal_. Na forma

textual, a atribuição de valor à um terminal è identificada pela notação ".=". A tabela àbaixo mostra os operadores binàrios de transformação existentes.

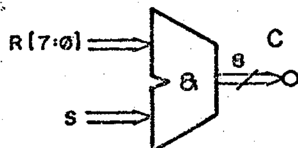
símbolo	operação
&	AND lògico
|	OR lògico
-&	NAND lògico
-|	NOR lògico
+	soma aritmètica
-	subtração aritmètica
xor	XOR lògico
nxor	NXOR lògico

Supondo as declarações

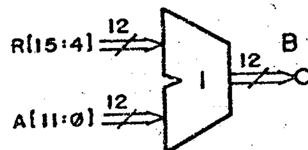
<u>register</u> R [15:Ø], S [7:Ø];
<u>terminal</u> A [15:Ø], B [11:Ø], C [7:Ø];

os exemplos abaixo mostram operações vàlidas.

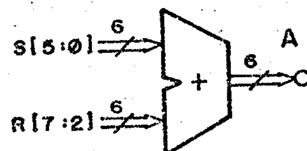
C .= R [7:Ø] & S;



B .= R [15:4] | A [11:Ø];



A [5:Ø] .= S [5:Ø] + R [7:2];



Operadores unàrios de transformação geram um sinal, necessariamente declarado como <u>terminal</u> ou <u>sub-terminal</u>, a partir de outro sinal de mesma dimensão. Graficamente, o operador unàrio è representado por um retângulo. A tabela abaixo mostra os operadores unàrios de transformação definidos na linguagem.

Nos operadores abaixo descritos, supos-se sempre deslocamentos e rotações de um bit para a esquerda ou a direita. Estes operadores têm uma forma genèrica

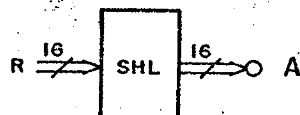
n op

que realiza deslocamentos e rotações de n bits numa ûnica operação. Da mesma forma, esta notação indica incremento ou decremento de um valor n, quando aplicada aos operadores **inc** e **dec**.

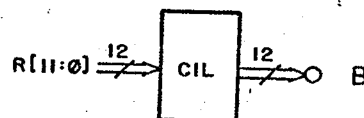
símbolo	operação
shl	deslocamento lógico para a esquerda (entra Ø no LSB)
shr	deslocamento lógico para a direita (entra Ø no MSB)
ashl	deslocamento aritmético para a esquerda (bit de sinal è conservado, entra Ø no LSB)
ashr	deslocamento aritmético para a direita (bit de sinal è conservado e duplicado no bit à direita)
cil	rotação para a esquerda
cir	rotação para a direita
not	complementação lógica
inc	incremento
dec	decremento
prir	prioridade à direita (filtra o bit = 1 mais à direita no vetor de entrada)
pril	prioridade à esquerda (filtra o bit = 1 mais à esquerda no vetor de entrada)

Supondo as declarações de portadores e terminais jà utilizadas anteriormente para os operadores binàrios, os exemplos a seguir mostram operações unàrias vàlidas.

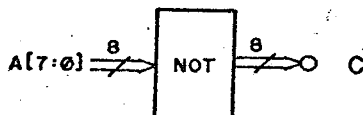
A .= shl R;



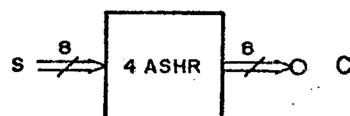
B .= cil R [11:Ø];



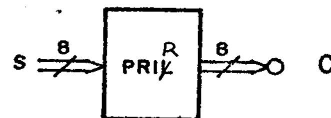
C .= not A [7:Ø];



C .= 4 ashr S;



C .= prir S;
(se S è p.ex. igual a 'ØØ1Ø11ØØ, C receberà o valor 'ØØØØØ1ØØ)



6.2 Operador de concatenação

O operador de concatenação, simbolizado por ":", gera um sinal combinacional de m + n bits de largura, a partir da justaposição de dois sinais quaisquer de larguras m e n respectivamente. Vàrias operações de concatenação podem ser realizadas simultaneamente num mesmo comando de atribuição. Graficamente, a concatenação è identificada por uma barra transversal, que tem de um lado o sinal de saìda e do outro os

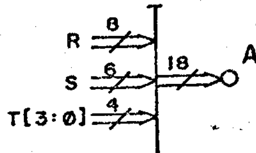
sinais fonte, colocados lado a lado na ordem em que serão concatenados. O bit mais significativo do vetor gerado è identificado por uma marca na extremidade correspondente da barra transversal.

Supondo as declarações

<u>register</u> R [7:Ø], S [5:Ø], T [11:Ø];
<u>terminal</u> A [17:Ø];

o comando

A .= R : S : T [3:Ø];



causa a concatenação dos sinais de saída dos registradores R, S e T [3:Ø], tal que o bit mais significativo de R será o bit mais significativo do vetor resultante, e o bit menos significativo de T será o bit menos significativo do vetor resultante.

6.3 Operadores compostos

Operações de transformação e de concatenação podem ser compostas livremente num único comando de atribuição de valor a um terminal. A ordem de avaliação dos operadores compostos è a seguinte:

- operadores unàrios são avaliados antes que os operadores binàrios;
- operadores binàrios são avaliados segundo a regra de prioridade estabelecida na tabela abaixo;
- operadores binàrios de mesma prioridade são avaliados da esquerda para a direita.

Operadores binàrios em ordem decrescente de prioridade na avaliação

1.	concatenação
2.	and, or, nand, nor, xor, nxor
3.	soma, subtração

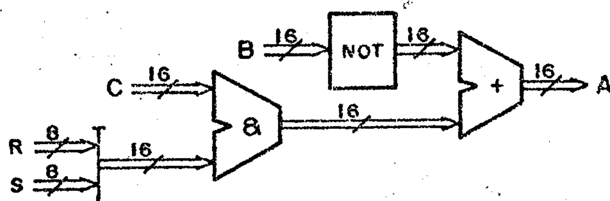
Esta ordem de avaliação pode ser modificada através do uso adequado de parênteses.

Supondo as declarações

<u>register</u> R [7:Ø], S [7:Ø];
<u>terminal</u> A [15:Ø], B [15:Ø], C [15:Ø];

o exemplo abaixo mostra uma operação composta vàlida.

```
A .= not B + R : S & C;
```



6.4 Operadores de comparação

Os operadores de comparação geram um sinal combinacional de 1 bit que é o valor booleano resultante da comparação entre dois sinais de n bits. Existem dois destes operadores, conforme a tabela abaixo.

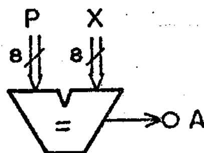
símbolo	operação
=	igualdade (resultado = 1 se os sinais têm mesmo valor)
-=	desigualdade (resultado = 1 se os sinais não têm mesmo valor

Como se trata de operadores binàrios, a forma gràfica utilizada è a da unidade aritmètica e lògica. Para diferenciar estes operadores mais claramente dos operadores de transformação, o sinal de saída deve estar sempre ligado a uma das arestas laterais do elemento gràfico. Supondo as declarações

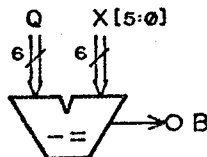
```
register  P [7:0], Q [5:0];
terminal  X [7:0], A, B;
```

os exemplos abaixo mostram operações vàlidas de comparação.

```
A .= P = X;
```




```
B .= Q -= X [5:0];
```



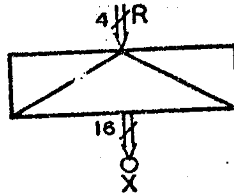
6.5 Operadores de codificação e decodificação

O operador de decodificação gera, a partir de um vetor de n bits cujo valor decimal è m, um sinal com 2**n bits, tal que apenas o (m+1)-èsimo bit deste sinal, a contar do seu bit menos significativo, tem valor igual a 1. Supondo as declarações

```
register  R [3:Ø];
terminal  X [15:Ø];
```

o comando abaixo gera um sinal X que corresponde á decodificação do sinal R:

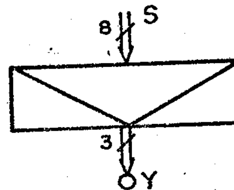
```
X .= decode R;
```



Se R tiver valor zero, X [Ø] terá valor 1.

O operador de codificação gera, a partir de um sinal de 2**n bits cujo (m+1)-ésimo bit, a contar do seu bit menos significativo, é igual a 1, um sinal de n bits com valor decimal igual a m. Se mais de um dos bits do sinal de entrada para este operador estiver ligado, a simulação acusará erro. Exemplo:

```
register  S [7:Ø];
terminal  Y [2:Ø];
Y .= encode S;
```



Se S tiver valor 'ØØØ1ØØØØ, então Y terá valor 4.

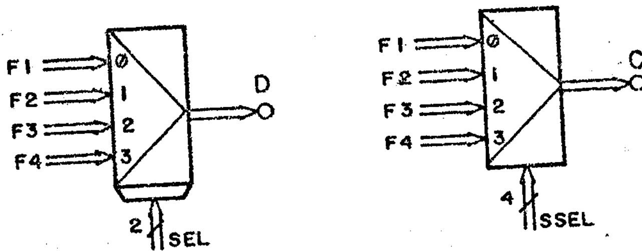
6.6 Operadores de multiplexação e demultiplexação

O operador de multiplexação permite controlar a conexão de um de 2**n sinais-fonte a um sinal-destino através de um sinal de seleção. Tanto o sinal-destino como todos os sinais-fonte devem ter a mesma dimensão.

O sinal de seleção pode ter largura de n bits, caso em que é considerado codificado, ou de 2**n bits, caso em que é considerado decodificado, devendo ser precedido pela palavra-chave sing (de vetor "singular", i.e., com apenas um bit igual a 1).

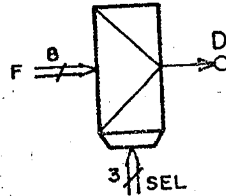
Na forma gráfica, o valor do sinal de seleção que corresponde a cada sinal-fonte é inscrito dentro do elemento gráfico "multiplexador" junto ao sinal-fonte. Na forma textual, os sinais-fonte são listados de acordo com os valores crescentes do sinal de seleção, a partir de Ø. Exemplos:

```
terminal  F1 [7:Ø], F2 [7:Ø], F3 [7:Ø], F4 [7:Ø],
          C [7:Ø], D [7:Ø], SEL [1:Ø], SSEL [3:Ø];
mux D .= case SEL of ( F1, F2, F3, F4 );
mux C .= case sing SSEL of ( F1, F2, F3, F4 );
```



É possível que, tendo os sinais-fonte e destino largura de 1 bit, os sinais-fonte sejam bits de um mesmo vetor:

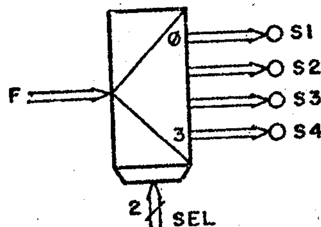
```
terminal  F [7:Ø], D, SEL [2:Ø];
mux D .= case SEL of F;
```



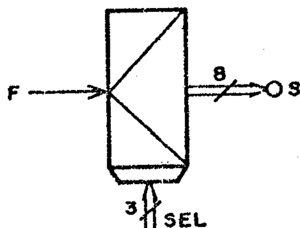
O operador de demultiplexação permite controlar a conexão de um sinal-fonte a um de 2**n sinais-destino, através de um sinal de seleção, que pode ser codificado e ter n bits de largura, ou ser decodificado, com 2**n bits de largura, sendo precedido por <u>sing</u>. Tanto o sinal-fonte como os sinais-destino devem ter a mesma dimensão. Como no caso do multiplexador, o valor do sinal de seleção que corresponde a cada sinal-destino é indicado dentro do elemento gráfico, junto ao sinal-destino. Na forma textual, os sinais-destino são listados na ordem crescente do valor correspondente do sinal de seleção, a partir de Ø. Também aqui é possível que os sinais-destino sejam bits de um mesmo vetor.

O operador de demultiplexação, assim como os operadores de condicionamento que serão vistos a seguir, introduzem a necessidade de lògica de 3 valores para a interpretação do comportamento do sistema. Apenas um sinal-destino, aquele selecionado pelo sinal de controle, recebe um valor Ø ou 1 em cada um de seus bits. Todos os demais sinais-destino têm um valor indeterminado. Dependendo da tecnologia na qual este sistema será construído, este valor indeterminado pode corresponder a um estado de alta impedância, ou mesmo a um valor Ø (no caso de utilização de portas lògicas tipo "open-collector" para a implementação do demultiplexador). Exemplos:

```
terminal  S1 [7:Ø], S2 [7:Ø], S3 [7:Ø], S4 [7:Ø],
          F [7:Ø], SEL [1:Ø];
demux case SEL of ( S1, S2, S3, S4 ) .= F;
```



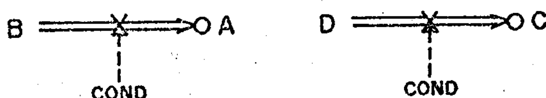
```
terminal  S [7:Ø], F, SEL [2:Ø];
demux case SEL of S .= F;
```



6.7 Operadores de condicionamento

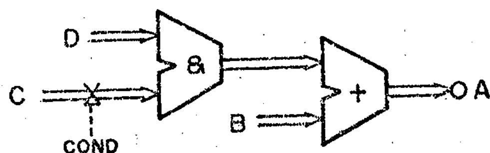
A atribuição de valor a um terminal pode ser condicionada, de tal modo que este terminal assuma o valor indeterminado enquanto um sinal de condicionamento não estiver ativo. A ativação deste sinal pode corresponder tanto ao valor lògico Ø como ao valor 1. Graficamente, o sinal de condicionamento deve ser representado através de uma linha tracejada. Exemplos:

```
terminal  A [7:Ø], B [7:Ø], C [5:Ø], D [5:Ø], COND;
A .= if COND then B fi;
C .= if not COND then D fi;
```



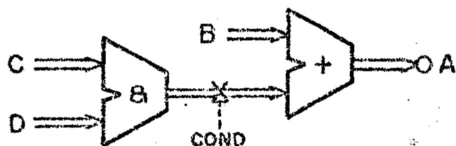
O condicionamento de terminais pode ser utilizado dentro de uma operação composta, como no exemplo abaixo:

```
terminal  A [3:Ø], B [3:Ø], C [3:Ø], D [3:Ø], COND;
* A .= B + if COND then C fi & D;
```



A ordem de avaliação da expressão fica clara pela terminação "fi", que identifica a qual sinal se aplica o condicionamento. Note-se que a expressão anterior é diferente de

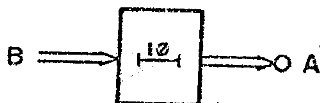
```
* A .= B + if COND then C & D fi;
```



6.8 Operadores temporais

O operador **delay** permite a geração de um sinal atrasado no tempo em relação a um sinal original. O atraso è especificado em unidades de tempo padrão. No exemplo

```
terminal  A [7:Ø], B [7:Ø];
A .= delay (1Ø) B;
```

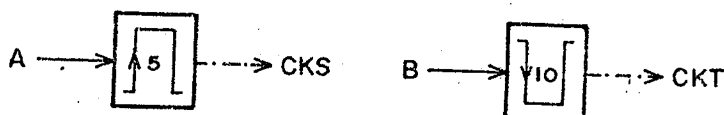


cada variação de valor num bit do sinal A serà atrasada de 1Ø unidades de tempo em relação à correspondente variação no bit do sinal B.

Note-se que a especificação de atraso è feita através de um operador independente dos demais operadores, embora nos sistemas reais o atraso esteja associado aos elementos tais como somadores, multiplexadores, registradores, etc.

O operador de diferenciação permite a geração de um sinal de relògio a partir da transição de um sinal de tipo qualquer. No exemplo

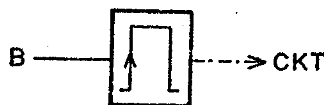
```
terminal  A,B;
clock  CKS,CKT;
CKS .= diff (5) A;
CKT .= ndiff (1Ø) B;
```



o sinal de relògio CKS recebe o valor 1 quando da transição positiva do sinal A, e retorna ao valor Ø apòs 5 unidades de tempo, enquanto o sinal CKT recebe o valor 1 quando da transição negativa do sinal B, retornando a Ø apòs 1Ø unidades de tempo. Sinais de relògio servem à carga de registradores, como serà visto mais adiante, e devem ser declarados explicitamente como sendo de tipo "clock". O sinal CKS gerado pelo comando do exemplo è um clock dito secundàrio, por ter sido gerado internamente a uma agência. O sinal a partir do qual è gerado o sinal de relògio deve ter largura de 1 bit.

No exemplo

```
terminal  B;
clock  CKT;
CKT .= diff B;
```



não foi especificada a duração do pulso de relògio. Neste caso, considera-se que o pulso tem duração zero. Operadores de diferenciação com pulsos de duração zero podem ser utilizados na descrição de agências ditas **sìncronas**.

Operadores "delay" e "diff" (com duração de pulso diferente de zero) caracterizam agências assíncronas.

A diferença entre estes dois tipos de agências será explicada no capítulo 14, e está relacionada ao algoritmo de simulação a ser utilizado para a interpretação do comportamento das agências.

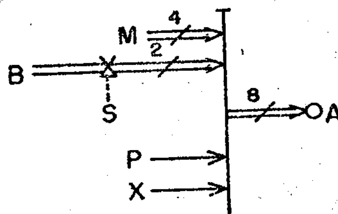
7. ATRIBUIÇOES E ACESSOS PARCIAIS A TERMINAIS

A um terminal com largura de bits maior do que 1 podem haver, ao longo de uma descrição textual, múltiplas atribuições de valor, desde que cada uma delas afete um bit diferente do sinal. Múltiplas atribuições de valor a um mesmo bit são proibidas.

Na descrição gráfica, estas múltiplas atribuições são modeladas com o auxílio do operador de concatenação, já introduzido.

Exemplo:

```
terminal  A [7:Ø], B [1:Ø], M [3:Ø], P, X, S;
A [3:2] .= if S then B fi;
A [Ø] .= X;
A [7:4] .= M;
A [1] .= P;
```

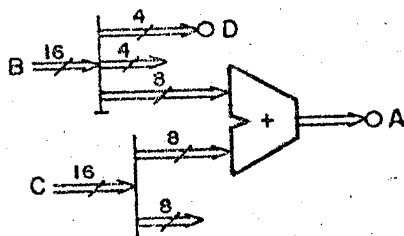


Note-se que a concatenação poderia ser declarada explicitamente também na descrição textual:

```
A .= M : if S then B fi : P : X;
```

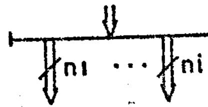
Assim como são possiveis atribuições individuais a bits isolados ou grupos de bits de um sinal declarado como terminal, é possivel o acesso de leitura a bits isolados ou grupos de bits. No exemplo

```
terminal  A [7:Ø], B [15:Ø], C [15:Ø], D [3:Ø];
A .= B [15:8] + C [7:Ø];
D .= B [3:Ø];
```



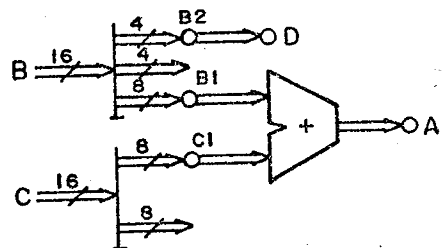
U F R G S
BIBLIOTECA
CPD/PGCC

são feitos acessos aos grupos de bits B [15:8], B [3:Ø] e C [7:Ø]. A forma gráfica



representa portanto a decomposição de um terminal de largura n em diversos sub-terminais com larguras n1,...,ni. Não é necessário que a decomposição seja uma partição completa dos bits do terminal.

Na forma textual, os acessos parciais aos sinais B e C não precisam utilizar a declaração explícita de sub-terminais, o que seria no entanto também possível caso se desejasse dar nomes alternativos aos sub-vetores acessados:

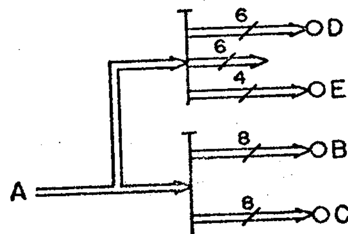



```
subterminal  B [B1] = B [15:8],
             B [B2] = B [3:Ø],
             C [C1] = C [7:Ø];
A .= B1 + C1;
D .= B2;
```

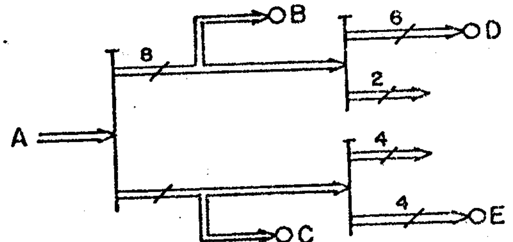
Na forma gráfica da operação de decomposição, sub-terminais serão declarados explicitamente se o usuário der nomes aos sub-vetores criados.

Através da construção gráfica de decomposição é impossível a definição implícita de sub-terminais que compartilhem bits de um terminal. Tal compartilhamento requer o uso de uma derivação de sinais, como mostrado abaixo:

```
terminal  A [15:Ø], B [7:Ø], C [7:Ø], D [5:Ø], E [3:Ø];
B .= A [15:8];
C .= A [7:Ø];
D .= A [15:1Ø];
E .= A [3:Ø];
```



A derivação introduzida na forma gráfica permite duplo acesso aos bits 15 até 1Ø e 3 até Ø do sinal A. Note-se que as operações de derivação e decomposição poderiam ser utilizadas de uma outra forma para se chegar ao mesmo resultado:



As duas formas gráficas acima corresponderiam ao mesmo texto, desde que sub-terminais não fossem declarados explicitamente.

8. BARRAMENTOS

Um barramento é um terminal de tipo especial, ao qual podem ser feitas múltiplas atribuições de valor, todas elas controladas através de sinais de seleção. O conjunto do barramento com a lógica de seleção funciona portanto como um multiplexador de caráter distribuído, no qual as fontes são sinais quaisquer, e o sinal de seleção é diferente para cada fonte.

Sinais-fonte devem ser ligados a um barramento através de operadores de condicionamento ou de demultiplexação. Barramentos podem ser utilizados livremente como sinais de entrada para outros operadores.

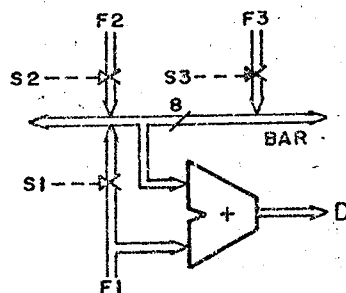
Um barramento deve ser declarado explicitamente por

```
bus  <nome_barramento>  <dimensão_barramento>;
```

Como o barramento é um tipo particular de terminal, adota-se para ele o mesmo símbolo de atribuição de valor adotado para os terminais, ou seja ".=".

Exemplo:

```
bus  BAR [7:0];
terminal  S1, S2, S3, F1 [7:0], F2 [7:0], F3 [7:0], D [7:0];
if S1 then BAR .= F1 fi;
if S2 then BAR .= F2 fi;
if not S3 then BAR .= F3 fi;
D .= BAR + F1;
```



Note-se que a construção textual que identifica a atribuição de valor a um barramento é

```
if S then BAR .= F fi
```

e não

```
BAR .= if S then F fi,
```

sendo esta segunda forma reservada para a operação de condicionamento aplicada a terminais comuns.

Uma vez que a lógica definida pelo projetista para o sistema esteja correta, apenas um dos sinais de seleção que condicionam as atribuições de valor a um barramento pode estar ativo a cada unidade de tempo. Isto não ocorrendo, haverá um conflito no barramento. Este conflito será identificado pelo simulador e comunicado ao usuário de dois modos:

- o usuário será notificado através de mensagem;
- a simulação continuará, sendo atribuído a cada bit do barramento o valor "indeterminado".

O valor a ser atribuído ao barramento em caso de conflito pode no entanto ser especificado pelo usuário da linguagem através da declaração da tecnologia com a qual o barramento será implementado. Esta declaração "estendida" do barramento substitui a declaração "bus" antes apresentada, de uma das seguintes formas:

- barramentos alimentados por portas tipo "open-emitter" são declarados por

downbus <nome_barramento> <dimensão_barramento>;

o que lembra o fato de que o barramento será implementado com um resistor "pull-down", e recebem o valor 1 em cada um de seus bits em caso de conflito;

- barramentos alimentados por portas tipo "open-collector" são declarados por

upbus <nome_barramento> <dimensão_barramento>;

o que lembra o fato de que o barramento será implementado com um resistor "pull-up", e recebem o valor 0 em cada um de seus bits em caso de conflito;

- barramentos alimentados por portas tipo "tri-state" são declarados por

tribus <nome_barramento> <dimensão_barramento>;

e recebem o valor "indeterminado" em cada um de seus bits em caso de conflito.

O usuário pode portanto optar por uma destas três alternativas ou pela declaração "bus", que não assume nenhuma tecnologia particular de implementação. Na forma gráfica, a designação do tipo de barramento será inscrita dentro do elemento gráfico correspondente.

9. CARGA DE REGISTRADORES

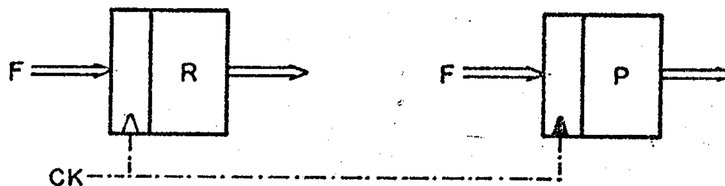
Registradores podem ser de tipo **sensível á borda**, **master-slave** ou **latch**. A diferenciação do tipo não se dá na declaração do registrador, que é idêntica para todos os tipos conforme visto no capítulo 4, mas sim quando do comando que especifica de que forma o registrador será carregado. A cada registrador pode corresponder um único comando de carga na descrição textual.

9.1 Registradores sensíveis á borda

Registradores sensíveis à borda são aqueles que copiam o valor do sinal de entrada quando da transição de um sinal de relógio, e exibem imediatamente em sua saída o novo valor

armazenado. Eles são especificados através de comandos "at", como nos exemplos abaixo:

```
register  R [7:0], P [7:0];
terminal  F [7:0];
clock  CK;
at CK do R := F ta;
at not CK do P := F ta;
```



O sinal de relògio è identificado graficamente pela forma "traço-e-ponto".

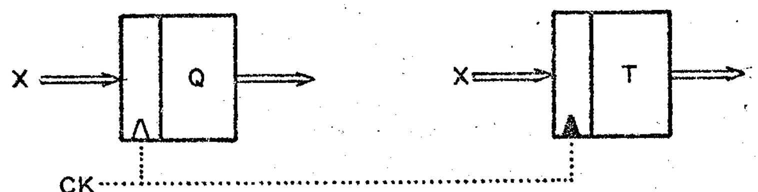
Como se trata de atribuição instantânea de valor a um portador, que tem capacidade de retenção deste valor, adota-se na forma textual o simbolo de atribuição ":=", para diferenciá-lo do adotado no caso de terminais. Este último corresponde na realidade à definição da saìda de uma lògica combinacional, que exibe constantemente um novo valor em função das entradas.

O sinal que controla a carga deve ter sido necessariamente declarado como de tipo "clock", seja ele um clock primàrio ou secundàrio (ver capìtulo 13). A carga pode ser controlada pela transição positiva ou negativa do sinal de relògio. Neste segundo caso, o operador "not" precede o nome do sinal de relògio no comando de carga. Na forma gràfica correspondente, a carga na transição negativa è identificada pelo sombreamento do pequeno triângulo colocado dentro do elemento gràfico "registrador", junto á entrada do sinal de relògio.

9.2 Registradores master-slave

Registradores master-slave são carregados numa transição de um sinal de relògio, mas sò mostram na saìda o novo valor armazenado apòs a transição de sentido inverso que se seguir neste sinal. O comando "on" identifica a carga de registradores deste tipo:

```
register  Q [15:0],
          T [15:0];
terminal  X [15:0];
clock  CK;
on CK do Q := X no;
on not CK
  do T := X no;
```



UFRGS
BIBLIOTECA
CPD/PGCC

Na descrição gràfica, o sinal de relògio è identificado pela forma "ponto-ponto". Como no caso dos registradores sensiveis à borda, o simbolo de atribuição è ":=".

A carga do valor de entrada para o registrador è feita na transição positiva do sinal de relògio, enquanto a exibição deste valor na saida ocorre apòs a transição negativa. A inversão da lògica de carga è feita precedendo-se o nome do sinal de relògio pelo operador "not", na forma textual, e sombreando-se o triângulo junto ao sinal de relògio, na forma gràfica.

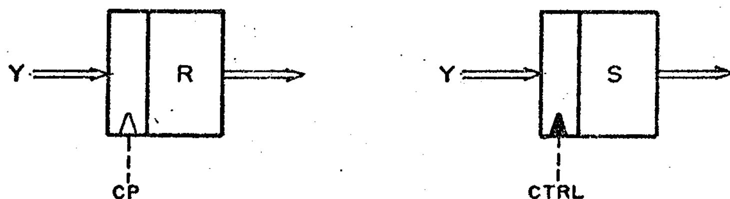
Como no caso anterior, o sinal de relògio deve ter sido declarado como de tipo "clock", podendo ser tanto um clock primàrio como secundàrio, este gerado por uma operação "diff". Neste caso, se nenhuma duração de pulso for associada à operação "diff", ambas as transições ocorrerão no mesmo instante de tempo, e o efeito do comando "on" serà o mesmo de um comando "at". Apenas no caso de ser especificada uma duração de pulso diferente de zero para o operador "diff" è que o novo conteùdo do registrador sò serà mostrado em sua saida num momento posterior à carga.

9.3 Registradores latch

Um registrador latch atualiza permanentemente sua saida enquanto o sinal de controle da carga estiver ativo. No momento em que este sinal assume o valor não ativo, o registrador fecha sua entrada, ignorando posteriores alterações que nela venham a ocorrer e mantendo na saida o ùltimo valor de entrada copiado. O comando "while" identifica a carga de um registrador latch. O sinal de controle pode estar ativo tanto em 1 como em Ø, sendo que neste caso seu identificador è precedido pelo operador "not". Ao contràrio dos outros dois tipos de registradores, um registrador latch pode também ser controlado por um sinal declarado como de tipo "terminal".

Exemplo:

```
register  R [11:Ø], S [11:Ø];
terminal  Y [11:Ø], CTRL;
clock  CP;
while CP keep R .= Y elihw;
while not CTRL keep S .= Y elihw;
```



Tendo em vista que, durante o tempo no qual o sinal de controle està ativo, o registrador se comporta como uma lògica combinacional, que reage imediatamente a qualquer alteração na entrada, o simbolo de atribuição adotado è ".=".

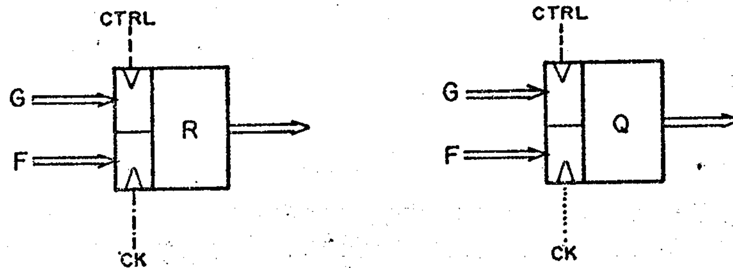
Na descrição gràfica, o sinal de controle è identificado

pela forma tracejada.

9.4 Controle combinado

É possível introduzir um controle assíncrono, como o dos registradores tipo "latch", em registradores tipo sensível à borda e master-slave. Exemplo:

```
register  R [7:Ø], Q [7:Ø];
terminal  F [7:Ø], G [7:Ø], CTRL;
clock  CK;
while CTRL keep R .= G otherwise at CK do R := F ta elihw;
while CTRL keep Q .= G otherwise on CK do Q := F no elihw;
```



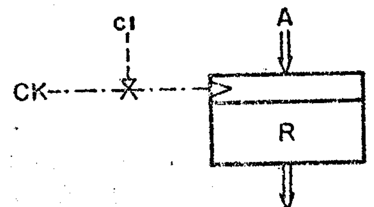
Nestes casos, o controle assíncrono tem prioridade sobre a carga síncrona. Enquanto o sinal CTRL estiver em 1, os registradores Q e R recebem o valor do sinal G. Quando CTRL não estiver em 1, os registradores Q e R poderão ser carregados em função das transições do sinal de relógio. Na descrição gráfica, o sinal de controle assíncrono e o sinal de relógio aparecem em lados opostos do registrador, sendo diferenciados por suas formas: o sinal de controle assíncrono é tracejado, enquanto o sinal de relógio, dependendo do caso, tem a forma "traço-e-ponto" ou "ponto-ponto".

9.5 Condicionamento do controle

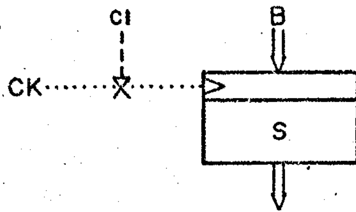
O controle de carga de um registrador pode ser condicionado através de um sinal qualquer, utilizando-se para isto a operação de condicionamento já introduzida no capítulo 6.

Exemplos de condicionamento de registradores com carga simples:

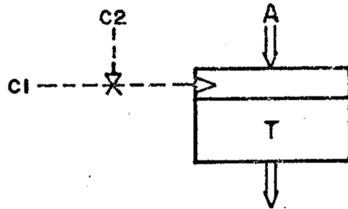
```
register  R [7:Ø], S [7:Ø], T [7:Ø];
terminal  A [7:Ø], B [7:Ø], C1, C2;
clock  CK;
if C1 then
  at CK do R := A
  ta fi;
```



```
if C1 then on CK do S := B no fi;
```

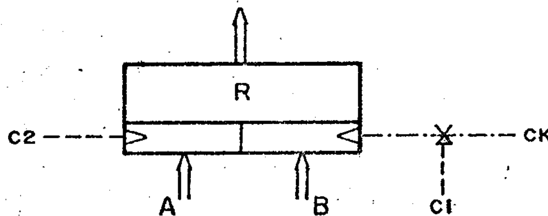



```
if C2 then while C1 keep T .= A elihw fi;
```



O condicionamento da carga de um registrador também pode ser feito no caso de controle combinado síncrono e assíncrono. Exemplo, usando as mesmas declarações anteriores:

```
while C2 keep R .= A otherwise if C1 then at CK do R := B ta
                                                         fi elihw;
```



A linguagem não permite no entanto o condicionamento simultâneo tanto do controle síncrono como assíncrono.

10. SUB-REGISTRADORES E REGISTRADORES CONCATENADOS

Sub-registradores e registradores concatenados podem ser declarados para que seja tornada mais fácil a descrição tanto de sua carga como de sua leitura. No entanto, a carga ou acesso parcial a um conjunto de bits de um registrador ou á concatenação de vários registradores dispensa a necessidade de declaração explícita de sub-registradores e registradores concatenados.

10.1 Carga de sub-registradores

A cada bit de um registrador pode haver no máximo uma atribuição de valor ao longo de uma descrição. Múltiplas atribuições de valor devem ser implementadas necessariamente através de um multiplexador colocado á frente da entrada de dados

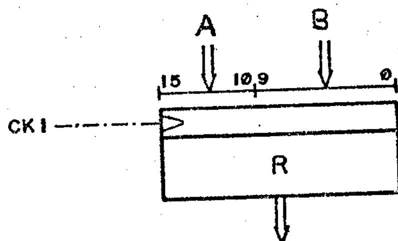
do registrador.

É possível no entanto que cada bit ou sub-conjunto de bits de um registrador seja carregado com um valor proveniente de um sinal-fonte diverso. Todas as cargas devem ser no entanto do mesmo tipo (sensível à borda, master-slave, latch ou uma combinação válida destas) e devem ser controladas pelo mesmo sinal de relógio ou controle. Supondo as declarações

```
register  R [15:Ø];
terminal  A [5:Ø], B [9:Ø];
clock  CK1;
```

os comandos

```
at CK1 do R [15:1Ø] := A ta;
at CK1 do R [9:Ø] := B ta;
```

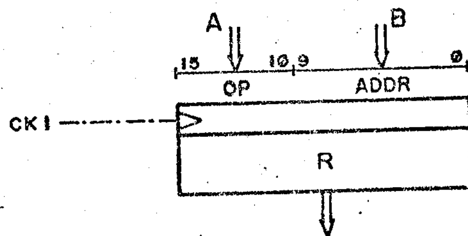


especificam que o registrador é do tipo sensível à borda, e que é carregado pelo sinal de relógio CK1. Seus 6 bits mais significativos são carregados com o valor do sinal A, enquanto os 1Ø bits menos significativos são carregados com o valor do sinal B.

Se o usuário deseja especificar cargas de tipos diferentes, ou realizadas com sinais de relógio ou de controle diferentes, é necessário declarar cada sub-conjunto de bits como um registrador diferente.

Os sub-registradores podem ser declarados explicitamente como tal. No exemplo anterior, isto resultaria em:

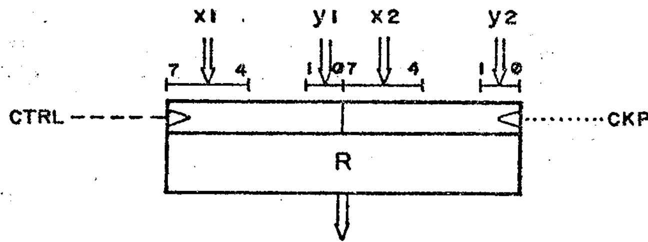
```
register  R [15:Ø];
subregister  R [OP] = R [15:1Ø],  R [ADDR]  = R [9:Ø];
terminal  A [5:Ø], B [9:Ø];
clock CK1;
at CK1 do OP := A ta;
at CK1 do ADDR := B ta;
```



O conjunto das atribuições aos sub-registradores de um certo registrador R não precisa cobrir todos os bits de R, como no exemplo abaixo:

```
register  R [7:Ø];
terminal  X1 [3:Ø], X2 [3:Ø], Y1 [1:Ø], Y2 [1:Ø], CTRL;
clock  CKP;
```

```
while CTRL keep R [7:4] .= X1 otherwise on CKP do R [7:4] :=
                                          X2 no elihw;
while CTRL keep R [1:Ø] .= Y1 otherwise on CKP do R [1:Ø] :=
                                          Y2 no elihw;
```



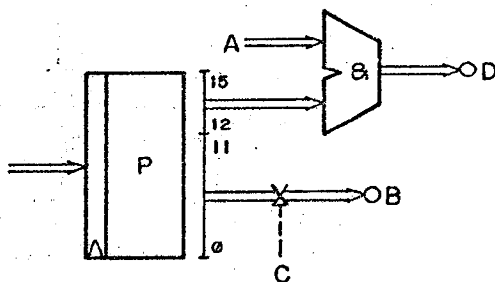
Nota-se que nenhuma atribuição è feita aos bits 3 e 2 de R. Embora gramaticalmente correta, esta descrição resultarà em uma indeterminação nos valores destes bits durante a simulação.

O exemplo ilustra a restrição de que os sub-registradores de um mesmo registrador devem ter o mesmo tipo de carga, no caso combinada assíncrona (controlada por CTRL) e síncrona (tipo master-slave, controlada por CKP). No exemplo fica também claro que num comando "while...keep...otherwise..." o registrador ou sub-registrador especificado na clàusula "keep" deve ser o mesmo utilizado na clàusula "otherwise".

1Ø.2 Leitura de sub-registradores

Em todos os comandos que utilizam o sinal de saída de um registrador como fonte para alguma operação, è possível especificar um sub-registrador, tenha ele sido declarado explicitamente como tal ou criado implicitamente pela utilização de um sub-conjunto de bits do registrador. Exemplo:

```
register  P [15:Ø];
terminal  A [3:Ø], B [11:Ø], C, D [3:Ø];
D .= A & P [15:12];
B .= if C then P [11:Ø] fi;
```



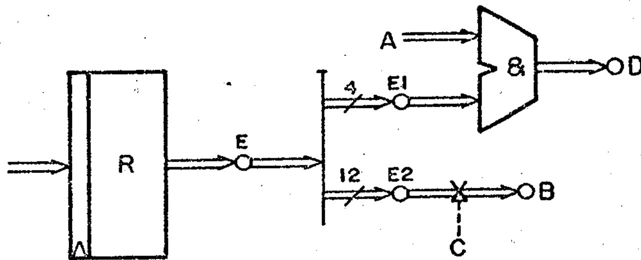
O mesmo resultado poderia ser obtido através da declaração explícita de sub-registradores:

```
sub-register  R [R1] = R [15:12],  R [R2] = R [11:Ø];
D .= A & R1;
B .= if C then R2 fi;
```

A forma gráfica seria a mesma, exceto pela especificação do nome dos sub-registradores.

Um resultado semelhante poderia ser obtido ainda de uma outra forma, utilizando-se o conceito de sub-terminal e o operador de decomposição, já introduzidos previamente:

```
register   R [15:Ø];
terminal   A [3:Ø], B [11:Ø], C, D [3:Ø], E [15:Ø];
sub-terminal   E [E1] = E [15:12],   E [E2] = E [11:Ø];
E .= R;
D .= A & E1;
B .= if C then E2 fi;
```

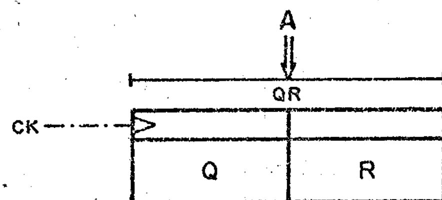


A diferença desta solução para as anteriores fica bem clara na forma gráfica. Um terminal adicional E foi criado, correspondendo ao sinal de saída do registrador R. Este terminal é então desmembrado em dois sub-terminais E1 e E2, que são utilizados nas operações subsequentes.

1Ø.3 Carga de registradores concatenados

Pode-se realizar a carga de um registrador formado pela concatenação de dois ou mais registradores. Este registrador concatenado deve ter sido declarado explicitamente como tal. No exemplo

```
register   Q [7:Ø], R [7:Ø];
casregister   QR = Q : R;
terminal   A [15:Ø];
clock CK;
at CK do QR := A ta;
```



o registrador formado pela concatenação de Q e R recebe o valor do terminal A, sendo o registrador concatenado assim formado do tipo sensível à borda e controlado pelo sinal de relógio CK.

Graficamente, a declaração de um registrador concatenado causa a justaposição dos registradores a serem concatenados, e o sinal de controle aparece uma única vez, aplicado sobre todo o conjunto.

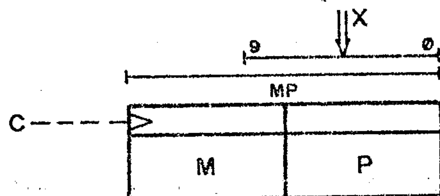
Não é possível a carga de um registrador concatenado formado implicitamente no próprio comando de atribuição. A expressão

```
at CK do Q : R := A ta;
```

é portanto invàlida.

É também possível a carga de um sub-registrador de um registrador concatenado, tenha este sub-registrador sido declarado explicitamente como tal ou não. Exemplo:

```
register   M [5:0], P [7:0];
casregister   MP = M : P;
terminal   X [9:0], C;
while C keep MP [9:0] .= X elihw;
```



Aplicam-se à carga de um registrador concatenado e seus sub-registradores as mesmas restrições feitas à carga de registradores simples e sub-registradores:

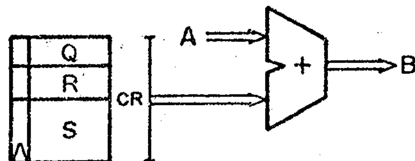
- apenas uma atribuição a cada bit do registrador concatenado pode existir ao longo de uma descrição;
- todas as atribuições a sub-conjuntos de bits de um registrador concatenado devem ser de mesmo tipo (sensível à borda, master-slave ou latch).

Como restrição adicional, não é possível carga combinada síncrona e assíncrona de um registrador concatenado.

10.4 Leitura de registradores concatenados

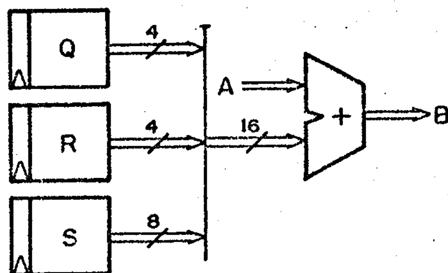
Em todo comando onde registradores possam ser utilizados como fonte de sinais, pode ser especificada a concatenação de registradores. A justaposição destes registradores não precisa ter sido declarada explicitamente como um registrador concatenado.
Exemplo:

```
register   Q [3:0], R [3:0], S [7:0];
casregister   CR = Q : R : S;
terminal   A [15:0], B [15:0];
B .= A + CR;
```



O mesmo resultado pode ser obtido se a justaposição dos sinais de saída dos registradores não for declarada explicitamente como sendo um registrador concatenado:

```
B .= A + Q : R : S;
```



Note-se que neste caso não existe um registrador concatenado, o que fica claro na descrição gràfica. Fez-se apenas a concatenação dos sinais de saìda dos registradores Q, R e S, o que è graficamente obtido a partir do operador de concatenação introduzido no capìtulo 6. Este operador na realidade cria um terminal obtido por concatenação, que neste caso não foi declarado explicitamente como um novo terminal, o que poderia ter sido feito da seguinte maneira:

```
terminal  X [15:0];
X .= Q : R.: S;
B .= A + X;
```

Graficamente, a ùnica diferença para a forma anterior reside na especificação do nome do terminal formado pela concatenação.

É também possìvel a leitura de um sub-registrador de um registrador concatenado, desde que este tenha sido declarado explicitamente como tal.

11. ARRAYS DE REGISTRADORES E MEMORIAS

Arrays de registradores podem ser acessados em leitura ou carga em uma de duas formas: por seleção estàtica de um dos registradores do array, caso o endereçamento seja feito atravès de uma constante, ou por seleção dinâmica de um registrador, no caso de endereçamento feito atravès de um sinal de valor variàvel.

Memòrias são casos particulares de arrays de registradores, sobre os quais se faz restrições quanto à forma de endereçamento e quanto a suas ligações com os outros elementos.

11.1 Carga de arrays

A carga de um array pode ser feita atravès de

- seleção estàtica do registrador (endereçamento por constante), ou
- seleção dinâmica do registrador (endereçamento por sinal de valor variàvel).

Em nenhum caso è permitida a carga de um sub-registrador de um registrador que faça parte de um array, nem à carga de um registrador concatenado do qual faça parte um registrador de um

array.

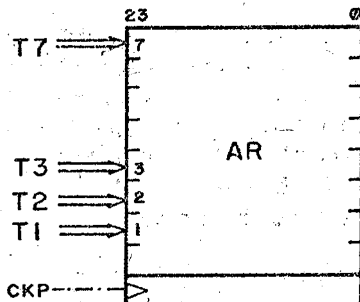
Todos os registradores de um array têm o mesmo tipo de carga (sensível à borda, master-slave, latch ou combinação válida destes), sendo que a carga combinada síncrona e assíncrona não é permitida para arrays com seleção estática de registradores. Todos os registradores de um array devem ser carregados e/ou controlados pelo(s) mesmo(s) sinal(is).

Suponda a declaração

```
array-register  AR [7:0;23:0];
```

que cria um array de 8 registradores de 24 bits cada, o exemplo abaixo mostra a carga deste array através de seleção estática de registradores:

```
terminal  T1 [23:0], T2 [23:0], T3 [23:0], T7 [23:0];
clock  CKP;
at CKP do AR [1;  ] := T1 ta;
at CKP do AR [2;  ] := T2 ta;
at CKP do AR [3;  ] := T3 ta;
at CKP do AR [7;  ] := T7 ta;
```

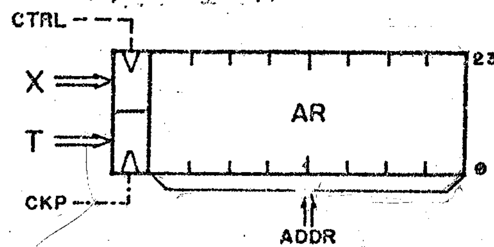


A forma AR [n;] indica o acesso a todos os bits do registrador de índice n dentro do array. Os comandos acima atribuem portanto os valores dos terminais T1, T2, T3 e T7 aos registradores de índice 1, 2, 3 e 7 do array, respectivamente. Note-se que não é necessária a especificação de atribuições a todos os registradores do array. Embora gramaticalmente correta, tal situação criará valores indeterminados como conteúdo de registradores.

A existência de pelo menos uma atribuição onde um registrador do array seja selecionado estaticamente exclui a possibilidade de fazer-se carga com seleção dinâmica no mesmo array.

O exemplo abaixo mostra o mesmo array anterior sendo carregado através de seleção dinâmica, usando-se um terminal ADDR para endereçamento. Desta feita, o array possui também um sinal de controle assíncrono.

```
terminal  T [23:0], ADDR [2:0], X [23:0], CTRL;
clock CKP;
while CTRL keep AR [ADDR;  ] .= X
     otherwise at CKP do AR [ADDR;  ] := T ta elihw;
```



Esta descrição diz que, quando CTRL for igual a zero e houver a transição positiva do sinal de relógio, o registrador cujo índice no array for igual ao valor decimal do sinal ADDR receberá o valor de T.

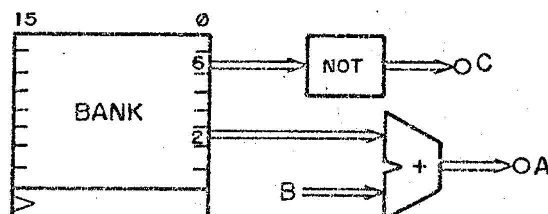
Note-se que o sinal ADDR deve ter a largura adequada para o endereçamento de todo o array, no caso 3 bits.

11.2 Leitura de arrays

Também na leitura de arrays pode-se fazer tanto seleção estática como dinâmica de registradores. Um tipo de seleção exclui no entanto a aplicação do outro ao mesmo array. Não é possível a leitura de um sub-registrador de um registrador de um array, nem a concatenação de um registrador de um array com outro registrador qualquer para fins de leitura.

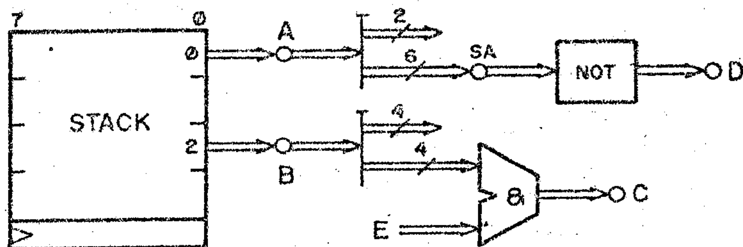
Exemplo de leitura com seleção estática:

```
array-register  BANK [7:Ø;15:Ø];
terminal  A [15:Ø], B [15:Ø], C [15:Ø];
A .= BANK [2;  ] + B;
C .= not BANK [6;  ];
```



Note-se que é possível a leitura de parte dos bits de um registrador de um array com o auxílio da operação de desmembramento de terminais ou com o auxílio de sub-terminais declarados explicitamente. Exemplo:

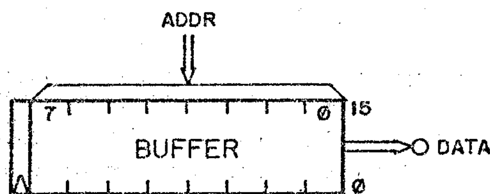
```
array-register  STACK [3:Ø;7:Ø];
terminal  A [7:Ø], B [7:Ø], C [3:Ø], D [5:Ø], E [3:Ø];
subterminal  A [SA] = A [5:Ø];
A .= STACK [Ø;  ];
D .= not SA;
B .= STACK [2;  ];
C .= E & B [3:Ø];
```



Da mesma forma que para a leitura de sub-registradores, terminais auxiliares podem ser utilizados para a concatenação do conteúdo de registradores de um array com outros sinais.

Exemplo de leitura com seleção dinâmica:

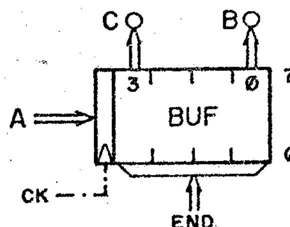
```
array-register  BUFFER [7:Ø;15:Ø];
terminal  DATA [15:Ø]; ADDR [2:Ø];
DATA .= BUFFER [ADDR; ];
```



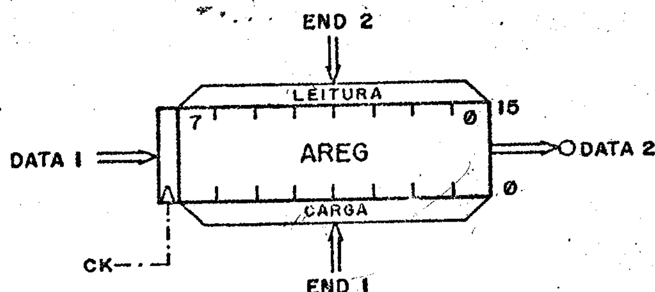
Também nestes caso terminais auxiliares podem ser utilizados para a leitura de sub-registradores ou para a concatenação do conteúdo de um registrador do array.

Ao se combinar a leitura e a carga de um array de registradores, pode-se adotar tipos de seleção diferentes para cada uma. No caso de se utilizar seleção dinâmica tanto para a leitura como para a carga, pode-se empregar sinais de endereçamento diferentes. Exemplos:

```
array-register  BUF [3:Ø;7:Ø];
terminal  A [7:Ø], B [7:Ø], C [7:Ø], END [1:Ø];
clock  CK;
at CK do BUF [END; ] := A ta;
B .= BUF [Ø; ];
C .= BUF [3; ];
```




```
array-register  AREG [7:Ø;15:Ø];
terminal  DATA1 [15:Ø], DATA2 [15:Ø],
                END1 [2:Ø], END2 [2:Ø];
clock CK;
at CK do AREG [END1; ] := DATA1 ta;
DATA2 .= AREG [END2; ];
```



11.3 Memòrias

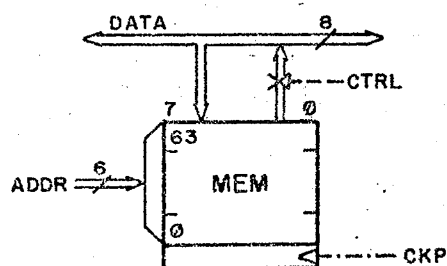
Memòrias são arrays de registradores sobre os quais se faz as seguintes restrições:

- sò è possível a carga e a leitura de registradores selecionados dinamicamente;
- o sinal de endereçamento è o mesmo tanto para a carga como para a leitura;
- se a memòria è acessada tanto em escrita como em leitura, o sinal de dados usado como fonte para a carga coincide com o sinal que recebe o valor lido numa leitura. Este sinal deve ser bidirecional, e portanto declarado como um barramento. A leitura deve ser condicionada nó entanto por um sinal de controle.

Valem as mesmas restrições aplicadas sobre os arrays de registradores quanto ao acesso a sub-registradores ou registradores concatenados dos quais façam parte registradores do array.

Exemplo de memòria acessada tanto em leitura como em escrita:

```
memory MEM [ ADDR ] = MEM [63:Ø;7:Ø];
bus  DATA [7:Ø];
terminal ADDR [5:Ø], CTRL;
clock  CKP;
at CKP do MEM := DATA ta;
if CTRL then DATA .= MEM fi;
```



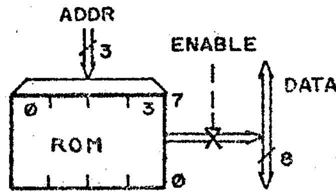
A declaração da memòria, neste caso com 64 palavras de 8 bits cada, já contèm também a designação do sinal utilizado para endereçamento. Nos comandos que descrevem a carga e leitura desta memòria, este sinal não precisa ser novamente especificado.

Como para os arrays de registradores, qualquer um dos 3 tipos de carga simples pode ser adotado para as memòrias. Cargas combinadas não são possíveis.

Uma memòria ROM pode ser modelada através da construção "memory", o que ocorre caso esta seja acessada apenas em leitura, como no exemplo abaixo:

UFRGS
BIBLIOTECA

```
memory  ROM [ ADDR ] = ROM [3:0;7:0];
terminal  DATA [7:0], ENABLE, ADDR [2:0];
DATA .= if ENABLE then ROM [ADDR; ] fi;
```

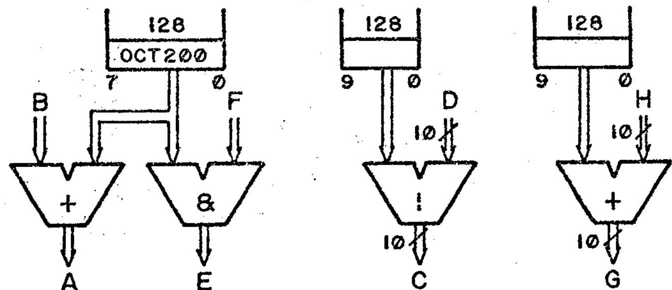


O conteúdo desta memòria ROM serà fixado quando de uma sessão de simulação, através do diàlogo que se estabelece entre o simulador e o usuàrio para inicialização do valor dos sinais das agências do modelo.

12. CONSTANTES

Constantes podem aparecer como fonte em quaisquer expressões envolvendo sinais, e não necessitam ser declaradas explicitamente. Exemplos:

```
constant  OCT200 [7:0] = 128;
terminal  A [7:0], B [7:0], C [9:0], D [9:0],
          E [7:0], F [7:0], G [9:0], H [9:0];
A .= B + OCT200;
E .= F & OCT200;
C .= D ! 128;
G .= H + 128;
```



A constante OCT200 foi declarada explicitamente como sendo um vetor de 8 bits. Ela pode ser utilizada agora em diferentes comandos na descrição. Tratar-se-à sempre da utilização do mesmo sinal como fonte, o que fica claro na representação gràfica, onde hà apenas um elemento OCT200, a partir do qual partem vàrias derivações.

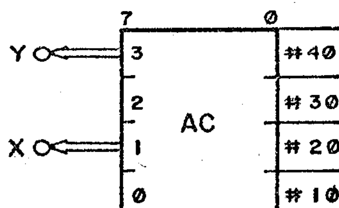
No caso do uso de uma constante não declarada explicitamente, o vetor criado terà a largura necessària para manter a compatibilidade com a largura dos demais sinais envolvidos na operação. No exemplo anterior, a constante 128 utilizada nos dois últimos comandos serà um vetor de 10 bits, em função dos sinais C, D, G e H. Cada nova utilização de um certo valor, mesmo que exigindo a mesma largura para o sinal, corresponderà à criação de um novo elemento, o que fica claro na descrição gràfica.

Graficamente, a constante declarada explicitamente

distingue-se das demais pela designação de seu nome. A partir de tal vetor è possivel a declaração de sub-terminais e terminais concatenados. Tal não è possivel na falta da declaração explicita, pois embora a situação fosse representàvel graficamente, ela não o è textualmente.

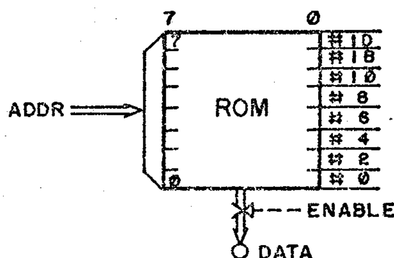
Um array de constantes pode ser compreendido como um array de registradores ao qual sò è permitido acesso em leitura, estando a carga interditada. Todas as considerações e representações adotadas no caso de leitura de arrays de registradores são também vàlidas para os arrays de constantes. Exemplo com seleção estàtica de constantes:

```
array-constant  AC [3:Ø;7:Ø] = #1Ø,#2Ø,#3Ø,#4Ø;
terminal  X [7:Ø], Y [7:Ø];
X .= AC [1; ];
Y .= AC [3; ];
```



Exemplo com seleção dinâmica de constantes:

```
array-constant  ROM [7:Ø;7:Ø] = #Ø,#2,#4,#6,#8,#1Ø,#1B,#1D;
terminal  DATA [7:Ø], ADDR [2:Ø], ENABLE;
DATA .= if ENABLE then ROM [ADDR; ] fi;
```



Como se vê, um array de constantes pode ser utilizado para modelar uma memòria ROM, cujo conteùdo è fixado dentro da pròpria descrição, e não pode portanto ser alterado de uma simulação para outra.

13. SINAIS DE RELÓGIO

Sinais de relògio são utilizados em uma descrição para especificar a condição pela qual um registrador ou uma memòria è carregado. Eles podem ser de dois tipos: primàrios, gerados pelo pròprio simulador, ou secundàrios, gerados dentro de agências a partir de outros sinais.

13.1 Sinais de relògio primàrio

Sinais de relògio primàrio aparecem na descrição de uma agência declarados como sinais de entrada na sua interface. Eles não podem receber nenhuma atribuição de valor dentro da agência, sendo usados sempre apenas como fonte em operações. Sinais de relògio primàrio podem ser conectados diretamente a uma saìda da agência. Esta facilidade de conectar uma entrada diretamente a uma saìda è especialmente ùtil no desenho de circuitos integrados.

Sinais de relògio primàrio podem ser multifàsicos, não havendo limite quanto ao nùmero de fases que pode ser declarado. Um relògio multifàsico è declarado por exemplo como

```
clock CK [1:3];
```

e suas fases são referidas como CK [1], CK [2] e CK [3], enquanto um relògio monofàsico è declarado

```
clock CKP;
```

Em todos os comandos de carga de registradores jà vistos anteriormente, pode-se utilizar tanto um clock monofàsico como uma fase de um clock multifàsico.

Relògios primàrios são declarados unicamente na interface da agência, e utilizados apòs a palavra-chave "behavior":

```
agency AY.1.1
level = KAPA
interface
        in CKP : clock;
           !
behavior
           !
        at CKP do ... ta;
           !
end
```

O perìodo e a largura dos pulsos de relògio são declarados no inìcio de uma sessão de simulação, apòs a construção do modelo simulàvel.

13.2 Sinais de relògio secundàrio

Um sinal de relògio secundàrio è criado atravès de uma de três operações:

- operação "diff" aplicada a um sinal de tipo "terminal", como visto no capìtulo 6.8;
- operação de condicionamento aplicada a um outro sinal de tipo "clock", corresponda ele a um relògio primàrio ou secundàrio;
- operação de còpia de um sinal de tipo "terminal".

Os sinais de relògio secundàrio podem ser gerados tanto

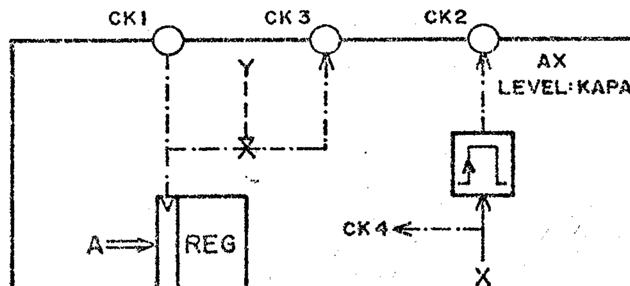
externamente à agência, caso em que devem ser declarados na sua interface, como no interior da agência, caso em que devem ser declarados após a palavra-chave "behavior". Relógios secundários devem ser necessariamente monofásicos.

Um sinal de relógio secundário pode ser conectado a uma saída da agência no interior da qual ele é gerado, de modo a ser utilizado por outras agências. Neste caso ele deverá ser declarado tanto na interface da agência como no seu interior, após a palavra-chave "behavior".

Um sinal de relógio de uma fase só, declarado na interface da agência, pode corresponder tanto a um relógio primário como a um secundário. A distinção só pode ser feita pelo contexto da agência. Se este sinal não for conectado a nenhum sinal de saída de outra agência, ele é um sinal de relógio primário, que será fornecido automaticamente pelo simulador. Se, ao contrário, ele estiver conectado à saída de uma outra agência, trata-se necessariamente de um sinal de relógio secundário. Para a agência que recebe este sinal a distinção é irrelevante, já que de qualquer forma os pulsos de relógio são gerados externamente a ela.

Exemplo:

```
agency  AX.1.1
level = KAPA
interface
     in  CK1 : clock;
         |
     out  CK2, CK3 : clock;
         |
behavior
     clock  CK2, CK3, CK4;
     register  REG [7:0];
     terminal  A [7:0], X, Y;
          |
     at CK1 do REG := A ta;
     CK2 .= diff X;
     CK3 .= if Y then CK1 fi;
     CK4 .= X;
          |
end;
```



14. AGÊNCIAS SÍNCRONAS E ASSÍNCRONAS

Agências síncronas diferenciam-se das assíncronas por não conterem:

- operadores "delay";
- especificação da duração dos pulsos gerados por operadores "diff".

Em função destas restrições, o comportamento das agências síncronas é uma abstração do sistema real, na qual se considera que todas as alterações de valor nos sinais de saída de portadores e elementos combinacionais ocorrem apenas quando de transições nos sinais de relógio primários, e são todas simultâneas entre si. A passagem de tempo se reduz à sequência de pulsos do relógio primário, sendo irrelevantes o período do relógio e a duração dos seus pulsos.

As agências assíncronas, por outro lado, modelam com mais precisão o comportamento temporal dos sistemas reais, pois é possível especificar atrasos de propagação na transição de valor de sinais combinacionais e relacionar estes atrasos ao "timing" real do relógio, especificado no início da sessão de simulação.

Uma explicação mais detalhada da interpretação dada ao comportamento de agências síncronas, nas quais co-existam registradores carregados por sinais de relógio primário e outros por sinais de relógio secundário, foge ao escopo deste relatório. Ela será fornecida em relatório posterior que tratará do simulador KAPA, que será implementado em duas versões, uma síncrona e outra assíncrona.

AGRADECIMENTOS

Agradeço a Carla M. Dal Sasso-Freitas e Flávio M. de Oliveira, responsáveis pela especificação do editor gráfico KAPA, e a João C. Netto e Luigi Carro, responsáveis por um estudo sobre o simulador KAPA, pela interação havida durante a fase final de especificação da linguagem KAPA.

REFERENCIAS BIBLIOGRAFICAS

[BAR75] BARBACCI, M.R. "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems". IEEE Transactions on Computers, Vol. C-24, No. 2, Feb. 1975. pp 137-150

[BAR81] BARBACCI, M.R. "Instruction Set Processor Specifications (ISPS): The Notation and its Applications". IEEE Transactions on Computers, Vol. C-30, No. 1, Jan. 1981. pp 24-40

[BOR79] BORRIONE,D. e J.F.GRABOWIECKI. "Informal Introduction to LASSO: a Language for Asynchronous Systems Specification and Simulation". In: P.A.Samet (ed), EURO-IFIP 79. North-Holland Publ. Co., Amsterdam, 1979. pp 419-426

[HAR77] HARTENSTEIN,R. Fundamentals of Structured Hardware Design. North-Holland Publ. Co., Amsterdam, 1977

[WAG84a] WAGNER,F.R. "Modelamento de Processos Digitais com Redes de Instâncias". Porto Alegre, CPGCC-UFRGS, Mar. 1984. 20p. (Relatòrio Tècnico 006)

[WAG84b] WAGNER,F.R. "Modelamento e Simulação de Processos Digitais Usando Redes de Instâncias". In: Seminàrio Integrado de Software e Hardware, XI. Viçosa, MG, 21-28 julho 1984. Anais, UFV, 1984. pp 309-318

[WAG86a] WAGNER,F.R. "A Multi-level Digital Systems Simulator based on Nets of Agencies:. In: JSST Conference on Recent Advances in Simulation of Complex Systems. Toquio, Japão, 15-17 julho 1986. Proceedings, 1986. pp 125-130

[WAG86b] WAGNER,F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "Equivalência de Descrições Textuais e Gràficas de Sistemas Digitais num Ambiente de CAD". In: Seminàrio Integrado de Software e Hardware, XIII. Recife, PE, 19-25 julho 1986. Anais, UFPe, 1986. pp 486-493

[WAG86c] WAGNER,F.R. et al. "Ambiente Integrado para Projeto de Sistemas Digitais Auxiliado por Computador". In: Congresso Nacional de Informàtica, XIX. Rio de Janeiro, RJ, 18-25 agosto 1986. Anais, Vol. 2, SUCESU, 1986. pp 111-116

[WAG87a] WAGNER,F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "Linguagens de Descrição de Hardware para Suporte à Integração do Processo de Projeto em AMPLO". Porto Alegre, CPGCC-UFRGS, Março 1987. 18p (Relatòrio de Pesquisa 065)

[WAG87b] WAGNER,F.R. e C.M.D.S.-FREITAS. "NILO - Uma Linguagem para Descrição de Hardware no Nível de Portas Lògicas". Porto Alegre, CPGCC-UFRGS, Março 1987. 18p (Relatòrio de Pesquisa 066)

[WAG87c] WAGNER, F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "A Digital Systems Design Methodology based on Nets of Agencies". Aceito para publicação em: 8th International Symposium on Computer Hardware Description Languages and their Applications. Amsterdam, Holanda, 27-29 abril 1987

Relatórios de Pesquisa

RP-67 ROCHA COSTA, A. C., "Seminário de Epistemologia da Inteligência Artificial, 1986: Máquina e Inteligência", março 87.

RP-66 WAGNER, F. R. & DAL SASSO-FREITAS, C. M., "NILO - uma linguagem para a descrição de hardware no nivel de portas lógicas", março 87.

RP-65 WAGNER, F. R., DAL SASSO-FREITAS, C. M. & GOLENDZINER, L. G., "Linguagens de descrição de hardware para suporte à integração do processo de projeto em AMPLO", março 87.

RP-64 ROCHA COSTA, A. C., "Prolog como linguagem de implementação de sistemas: oito programas ilustrativos", fev 87.

RP-63 POLANCZYK, C. A., CLAUDIO, D. M. & LOPEZ, J. D., "Software elementar para intervalos", fev 87.

RP-62 WESTPHALL, C. B., "Sinópse da especificação MAP", dez 86.

RP-61 ROCHA COSTA, A. C., "Sobre os fundamentos da inteligência artificial", nov 86.

RP-060 WALTER, C., "A method for the specification of manufacturing systems and their controllers", out 86.

RP-059 PALAZZO OLIVEIRA, J.P. & RUIZ, D.D.A., "Formulários Eletrônicos: definição e manipulação automática da interface de usuário", set 86.

RP-058 TOSCANI, L.V. & SZWARCFITER, J.L., "Algoritmos aproximativos", set 86.

RP-057 JANSCH-PORTO, I. & COURTOIS, B., "Design and assembling of SDC checkers based on analytical fault hypotesis", set 86.

RP-056 TAZZA, M., "Fundamentals of a Net Theory Based Performance Evaluation Model", ago 86.

RP-055 FREITAS, C.M.D.S & OLIVEIRA, F.M, "Editor gráfico para sistemas digitais descritos no nivel lógico", ago 86.

RP-054 ROCHA COSTA, A.C., "Para uma revisão epistemológica da inteligência artificial", jul 86.

RP-053 TAZZA, M., "Quantitative analysis of a resource allocation problem: a net theory based proposal", jul 86.

UFRGS
BIBLIOTECA
CPD/PGCC

RP-052 LONGHI, M.T., "Representação de objetos sólidos por octrees", julho 86.

RP-051 NASCIMENTO, F.R. do, Kit MCP-68000: Descrição do projeto de hardware., julho 86.

RP-050 MURR, A.L.D. & CASTILHO, J.M.V. de, Tradução de sentenças em lògica para sentenças em forma clausal: uma implementação em PROLOG.", julho 86.

RP-049: TOSCANI L.V. & VELOSO P.A.S., Especificação formal e anàlise da complexidade da programação dinâmica., mai 86.

RP-048: HURTADO, J.O.H.; GOMES, R.F. & SEELING, M., Documentação da concepção e testes de um circuito integrado NMOS. jun/86.

RP-047: PALAZZO OLIVEIRA, J., Electronic forms: a local area microcomputer system - Project description. (também disponivel em portuguès.), mai/86.

RP-046: TOSCANI, L.V. Guia de estudo da complexidade de algoritmos de procura. nov/85.

RP-045: NASCIMENTO, F.R. & OLIVEIRA, R.S., Programa monitor para um microcomputador educacional implementado com o MC68000. dez/85.

RP-044: WAGNER, F.R., Simulação de sistemas modelados como redes de agencias. nov/85.

RP-043: FREITAS, C.M.D.S. & OLIVEIRA, F.M., Editor gráfico para sistemas modelados como redes de agencias. nov/85.

RP-042: TAZZA, M., Sistemas-C/E como ferramenta de modelagem. out/85.

RP-041: WAGNER, F. R.; FREITAS, C. M. D. S. & GOLENDZINER, L. G., O processo de projeto de sistemas digitais num ambiente integrado de CAD. out/85.

RP-040: TAZZA, M., Performance evaluation using a net theory based model. set/85.

RP-039: WAGNER, F. R., On the properties of event oriented logic simulation according to significant timing models. set/85.

RP-038: WAGNER, F. R., Algoritmos de simulação de hardware no nivel RT. set/85.

RP-037: COSTA, A. C. R., Processando linguagens naturais em PROLOG - Parte 1: Formalismo gramatical básico. jul/85.