Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática

# "Metodologias de Teste Funcional para Microprocessadores"

Relatório Técnico

Eduardo Augusto Bezerra
<eduardob@inf.ufrgs.br>

Porto Alegre, fevereiro de 1996

## SUMÁRIO

# 1 CONTEXTUALIZAÇÃO

Conforme colocado no relatório técnico "Teste de Sistemas Digitais", o teste de processadores em sistemas multiprocessados pode ser realizado a nível de processador ou a nível de sistema. Este segundo grupo verifica basicamente problemas que afetam a comunicação entre os componentes do sistema, mas não a funcionalidade de cada componente. No teste a nível de processador, são aplicados vetores de teste que sensibilizam elementos internos dos processadores, com o objetivo de verificar a ocorrência de falhas não detectadas no teste a nível de sistema. De acordo com essa divisão, na tabela 1, estão listados métodos de teste encontrados na literatura, classificados em dois grupos distintos correspondentes aos dois níveis abordados.

Tabela 1 - Metodologias de teste a nível de processador e a nível de sistema, para processadores convencionais e para o processador transputer.

| Grupos | Processadores Convencionais | Processador Transputer |
|---|---|---|
| **1. Teste a nível de processador** | [THA80] [VEL82] [SHE84] [ROB80] [BRA84] [SAL92] [ABR81] [FED84] [ANN82] [FRE84] | [BEZ95] |
| **2. Teste a nível de sistema** | [PRE67] [AVR87] [BLO90] [RUS75] [BAN86] [NAI92] [RUS75a] [SCH86] [BLO93] [HUA84] [YAN86] [SIT93] | [NIC88] [KUM93] [THO91] [KUK94] [CAS92] [TOR94] [CAS92a] [PAU95] |

Sendo o objetivo desse estudo a definição de um procedimento de teste a nível de processador, no presente relatório técnico será realizada uma breve descrição de alguns dos procedimentos de teste apresentados nos trabalhos do Grupo 1. Os métodos do Grupo 2, por serem procedimentos a nível de sistema, não se enquadram nos objetivos do presente trabalho e não serão analisados.

Os métodos de teste do Grupo 1, são baseados nos métodos propostos em [THA80] ou [ROB80], os quais utilizam o conjunto de instruções e registradores (organização interna) para realização do teste de processadores. Por se tratarem de trabalhos clássicos na área de testes de processadores, a descrição dos métodos propostos em [THA80] e [ROB80] será mais detalhada, com ênfase maior em [ROB80], uma vez que o procedimento de testes a ser proposto para o processador a ser escolhido, será desenvolvido de acordo com essa abordagem.

No final desse relatório técnico são realizados comentários a respeito da aplicação dos métodos ao processador selecionado, e sobre a escolha do método utilizado.

# 2 GERAÇÃO DE TESTES PARA MICROPROCESSADORES

O método de Thatte e Abraham [THA80] foi desenvolvido com o objetivo de ser genérico, ou seja, aplicável a qualquer microprocessador. No artigo é descrito um modelo para representação de microprocessadores, por intermédio de um grafo de sistema (system graph ou S-graph), no nível de transferência de registradores. Duas características principais do método são: modelagem do microprocessador utilizando seu conjunto de instruções e funções por elas realizadas; e utilização de um modelo de falhas no nível funcional, independente de detalhes de implementação física do microprocessador. São propostos procedimentos de geração de testes que utilizam a organização interna do microprocessador e seu conjunto de instruções como parâmetros, e geram testes para detectar todas as falhas pertencentes ao modelo de falhas utilizado.

## 2.1 Definições gerais

Na figura 1, é apresentado um exemplo de utilização do grafo de sistema para representação de um microprocessador hipotético que possui quatro registradores ($R_1$, $R_2$, $R_3$ e $R_4$) e nove instruções ($I_1$ .. $I_2$).

Os nodos de um grafo de sistema representam os registradores do microprocessador, e os arcos representam as instruções que causam fluxo de dados entre os nodos. O meio externo (memória e dispositivos de entrada/saída) é representado pelos nodos IN e OUT que são, respectivamente, origem e destino das informações processadas no grafo. Para identificar a seqüência do fluxo dos dados durante a execução de uma instrução Ij, são utilizados os índices p e q. Assim, para representação da ocorrência de $I_j^p$ antes de $I_j^q$, utiliza-se p < q.

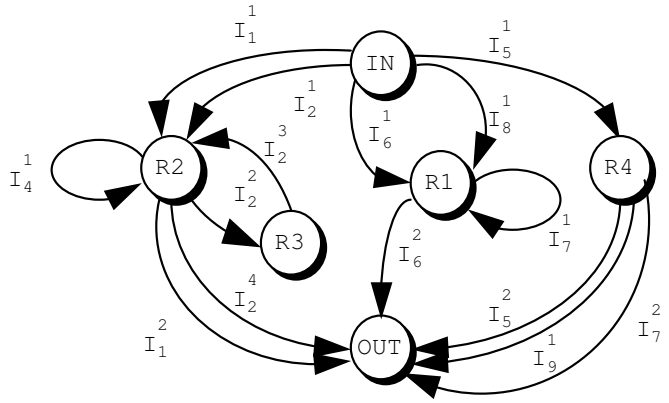


Figura 1 - Representação de um microprocessador hipotético utilizando o grafo de sistema.

Algumas definições são utilizadas na construção dos procedimentos de teste:

- S (Ij) é o conjunto composto pelos registradores que fornecem os operandos para a instrução Ij durante sua execução (registradores origem);
- D (Ij) é o conjunto composto pelos registradores que são alterados pela instrução Ij durante sua execução (registradores destino);
- E (Ij) é o conjunto composto pelos arcos que representam uma instrução Ij no grafo de sistema;
- WRITE (Ri) representa a seqüência mais curta de instruções de desvio ou transferência, necessária para alterar o conteúdo do registrador Ri (implícita ou explicitamente);
- READ (Ri) representa a seqüência mais curta de instruções de desvio ou transferência, necessária para ler o conteúdo do registrador Ri (implícita ou explicitamente).
  É importante ressaltar que a escrita em Ri é realizada com dados a partir da entrada do grafo (nodo IN), e a leitura do dado contido em Ri é realizada a partir da saída do grafo (nodo OUT).

## 2.2 Modelos de falhas

Os modelos de falhas utilizados para o microprocessador a ser testado são definidos em um alto nível de abstração (nível funcional), onde detalhes a respeito da implementação do hardware são abstraídos. A seguir, são listados os modelos de falhas definidos em [THA80] para um microprocessador genérico.

**Modelo de falhas para a função de endereçamento de registradores:**

A função de endereçamento de registradores é responsável pela tarefa de decodificação de endereços de registradores. O endereço de um registrador pode estar armazenado (como uma seqüência de bits) na instrução que envolve o registrador, ou ser gerado pela unidade de controle durante a execução de instruções.

No modelo de falhas para essa função, é possível descrever falhas no decodificador de endereços de registradores, e nos multiplexadores e demultiplexadores, responsáveis pela seleção dos registradores. As seguintes falhas são modeladas:

- ao ser executada, uma instrução Ij utiliza um registrador R1 no lugar do desejado R2;
- ao ser executada, uma instrução Ij utiliza um ou mais registradores adicionalmente ao registrador desejado R1;
- ao ser executada, uma instrução Ij que deveria utilizar um ou mais registradores, não utiliza nenhum.

Modelo de falhas para a função de decodificação e controle de execução de instruções:

Essa função é responsável pelo tratamento de um dado obtido após a etapa de busca na memória. Nesse instante é necessário decodificar o dado obtido na busca, a fim de identificar a instrução a ser executada. Outra tarefa exercida por essa função é o controle da execução das instruções (seqüencialização das instruções).

As seguintes falhas são modeladas para a função de decodificação e controle de execução de instruções:

- ao invés da instrução desejada Ij ser executada, uma instrução Ik é executada no seu lugar;
- adicionalmente a instrução Ij, uma outra instrução Ik é executada;
- nenhuma instrução é executada.

**Modelo de falhas para a função de armazenamento de dados:**

Essa função é responsável pelo armazenamento de dados em registradores. O modelo de falhas assumido para essa função é o stuck-at (0 ou 1), sendo que essas falhas podem ocorrer em qualquer número de células de um registrador e em qualquer registrador.

**Modelo de falhas para a função de transferência de dados:**

Essa função é responsável pela transferência dos dados entre as diversas localizações de um microprocessador, utilizando os caminhos disponíveis. Para qualquer instrução Ij que realize uma transferência de dados, uma das seguintes falhas pode ocorrer:

- uma linha em um caminho de transferência pode estar stuck-at-0 ou stuck-at-1;
- duas linhas de um caminho de transferência podem estar acopladas (linhas em curto ou acoplamento capacitivo).

**Modelo de falhas para a função de manipulação de dados:**

A manipulação de dados é realizada pelas diversas unidades funcionais de um microprocessador, tais como: ULA; módulos responsáveis pelo incremento/decremento do apontador para a pilha (stack pointer); contador de programa (program counter) ou registrador de

índice (indice register); módulo responsável pelo cálculo do endereço de operandos nos vários modos de endereçamento (indexado, relativo, ...); etc.

Devido à grande variedade de unidades funcionais existentes, não foi proposto nenhum modelo de falhas específico para essa função, uma vez que um processador pode possuir certas unidades funcionais que outro não possui.

## 2.3 Procedimentos para geração de testes

A geração e execução de testes para as falhas previstas nos modelos de falhas são realizadas em três etapas principais:

1. escrita de vetores de teste nos registradores origem S(Ij), utilizando a seqüência mais curta de instruções de desvio ou transferência, necessária para alterar o conteúdo do registrador Ri (WRITE(Ri));
2. leitura dos valores obtidos nos registradores destino D(Ij), utilizando a seqüência mais curta de instruções de desvio ou transferência, necessária para ler o conteúdo do registrador Ri (READ(Ri)); e,
3. comparação do valor lido com o valor esperado, onde o valor esperado pode ser o mesmo que havia sido escrito, ou um determinado padrão definido previamente.

O método proposto utiliza um testador externo para monitorar todos os pinos do microprocessador. O testador externo armazena as seqüências de teste geradas de acordo com o modelo de falhas assumido; o microprocessador a ser testado executa as seqüências de teste, e o testador compara as respostas observadas com as esperadas. O teste é interrompido na ocorrência (detecção) de alguma saída diferente da saída esperada. O microprocessador é considerado sem falhas no caso de, após a execução de todos os procedimentos de teste, os valores lidos serem iguais aos valores esperados.

## 2.4 Teste Funcional de Microprocessadores

O método de teste apresentado em [BRA84] é uma atualização de [THA80], sendo que a principal diferença é a proposta de um novo modelo para o processo de execução de instruções (função de decodificação e controle de execução de instruções). Outros pontos a serem destacados nesse método são: proposta de um grafo de sistema reduzido para representação de microprocessadores; utilização de um microprocessador real (Motorola 68000) como exemplo de representação no grafo de sistema reduzido; e execução dos procedimentos de teste no modo de autoteste, dispensando assim a necessidade de um circuito testador externo.

O artigo apresenta um conjunto de testes funcionais completo para microprocessadores, gerado com base no novo modelo de falhas para o processo de execução de instruções, juntamente com os modelos de falhas para as funções de endereçamento de registradores, armazenamento de dados, transferência de dados e manipulação de dados defendidos pelo precursor.

O novo modelo para o processo de execução de instruções visa melhorar o método proposto em [THA80], no qual, para verificar o correto funcionamento da função de seqüencialização de instruções, é necessário executar cada instrução gerando testes para verificar se a instrução foi corretamente executada e nenhuma outra foi executada em seu lugar. Isso pode tornar o teste muito complexo e demorado, pois o conjunto de testes depende do conjunto de arcos da instrução sob teste e do conjunto de arcos da instrução falha que é executada adicionalmente à instrução correta. Além disso, cada instrução que possui um modo de endereçamento diferente, é representada por um conjunto de arcos diferente. Assim, um microprocessador com 48 instruções diferentes e 14 modos

de endereçamento diferentes, poderá necessitar de até 1536 arcos para sua representação no grafo de sistema.

Em [BRA84], para a representação das instruções, são utilizados os conceitos de microinstruções e microordens. Uma instrução é composta por uma seqüência de microinstruções e cada microinstrução é formada por um conjunto de microordens que são executadas em paralelo. Segundo os autores, o conjunto completo de microordens pode facilmente ser construído com o conhecimento do conjunto de instruções (obtido a partir do manual do usuário fornecido pelo fabricante do microprocessador). As microordens são divididas em três tipos: tipo 0, se utiliza apenas um registrador; tipo 1, se envolve uma transferência de dados entre registradores ou se realiza uma operação lógica; e tipo 2, se realiza uma operação aritmética. Com a utilização dos conceitos de microinstruções e microordens, é possível representar falhas referentes à execução parcial de instruções, o que não é possível no modelo original proposto em [THA80].

Dessa forma, o novo modelo de falhas para a função de decodificação e controle de execução de instruções compreende as seguintes falhas:

- em um determinado instante uma ou mais microordens podem estar inativas, logo a instrução pode não ser executada completamente;
- microordens que são normalmente inativas, tornam-se ativas; e
- um conjunto de microinstruções permanece ativo, adicionalmente às (ou ao invés das) microinstruções normais.

O grafo de sistema reduzido utilizado na representação do microprocessador é similar ao proposto em [THA80]. Da mesma forma, os nodos representam os registradores, e os arcos representam as instruções que causam fluxo de dados entre os registradores. A principal diferença é a utilização do conceito de registradores equivalentes. Dois registradores são equivalentes a um determinado conjunto de instruções I, se, e somente se, alguma instrução que utiliza um dos registradores (em um modo de endereçamento particular) pode utilizar no seu lugar o outro registrador como seu operando. Um conjunto de registradores que apresentam as características desses dois registradores constituem um conjunto de registradores equivalentes. Como exemplo de registradores equivalentes, pode-se citar os oito registradores de dados D0-D7, e os sete de endereço A0-A6 do 68000. O registrador de endereço A7 não pertence a mesma classe de equivalência de A0-A6, uma vez que A7 é utilizado implicitamente como um ponteiro para a pilha.

A geração dos testes é realizada de maneira similar a [THA80]. A principal diferença é que inicialmente são gerados (e executados) testes para detecção de falhas mais simples como, por exemplo, as que ocorrem nas instruções de leitura dos registradores. Uma vez verificado que as instruções de leitura estão livres de falhas, o correto funcionamento das instruções remanescentes é testado, carregando-se vetores de teste nos registradores (por intermédio das seqüências WRITE) e, a seguir, executando-se as instruções e lendo-se os registradores (por intermédio das seqüências READ). Os valores lidos a partir do nodo OUT são comparados com valores esperados por intermédio de instruções de comparação do próprio microprocessador, dispensando assim a necessidade de um testador externo.

## 2.5 Teste Funcional no Ambiente do Usuário

O método descrito em [FRE84] é um aprimoramento do método proposto em [THA80]. As principais diferenças são: utilização de um número reduzido de vetores de teste, obtidos por intermédio de uma seleção sistemática nos valores a serem utilizados; e, da mesma forma que o método proposto em [BRA84], utilização no ambiente do usuário, sem necessidade de um testador externo.

O microprocessador a ser testado é representado por um grafo de sistema semelhante ao definido em [THA80]. A principal diferença é a atribuição de um par ordenado (x, y) para cada nodo do grafo. Nesse par, o identificador "x" representa o número mínimo de instruções de transferência necessárias para carregar o nodo com um dado, e "y", o número mínimo de instruções de transferência necessárias para ler o conteúdo do nodo.

De maneira similar ao proposto em [BRA84], cada instrução é quebrada em um conjunto de microoperações com o objetivo de permitir a detecção de falhas na execução parcial de instruções.

Com relação ao modelo de falhas utilizado, a única diferença em relação ao modelo proposto em [THA80] é a utilização da função de decodificação e controle de execução de microoperações, no lugar da função de decodificação e controle de execução de instruções. Com isso é possível o tratamento da execução parcial de instruções.

# 3 TESTE FUNCIONAL DE MICROPROCESSADORES

O método proposto por Robach e Saucier [ROB80], assim como os métodos descritos nas seções anteriores, utiliza o conjunto de instruções e registradores para a verificação dos diversos blocos funcionais de um microprocessador. As principais características desse método são: teste realizado no nível funcional, porém utilizando algum conhecimento a respeito da estrutura de alguns módulos do microprocessador (ex: quantidade de células em registradores), podendo-se dizer que o método possui características funcionais e estruturais; e capacidade de diagnóstico a nível de software, apontando uma instrução ou conjunto de instruções que se encontrarem em um estado inconsistente, e hardware, apontando o módulo físico faltoso.

O método consiste basicamente na definição de grafos para representação das instruções do processador e posterior análise e classificação dos grafos, com o objetivo de obtenção de um conjunto mínimo de instruções que exercitem todo o processador. Os grafos representam as etapas de processamento das instruções, mostrando tudo o que é alterado no processador no momento da execução destas.

## 3.1 Representação do microprocessador

Quanto à forma de representação do microprocessador, a principal diferença com relação a proposta de [THA80] é a utilização de grafos de execução abstrata. Nos métodos descritos anteriormente todo o microprocessador é descrito por um único grafo (grafo de sistema), enquanto que no presente método, cada instrução é representada separadamente por um grafo (grafo de execução abstrata).

Detalhes a respeito de grafos genéricos podem ser encontrados em [CHE76]. Para construção do grafo de execução abstrata de uma instrução são utilizados os seguintes princípios: os vértices subdividem-se em vértices do primeiro tipo, aos quais estão associados os elementos de memória (registradores e memória) sendo representados graficamente por círculos, e vértices do segundo tipo aos quais estão associadas as microoperações realizadas durante a execução da instrução, sendo representados graficamente por retângulos (por questão unicamente de diferenciação). Se a microoperação processa o dado armazenado no elemento de memória, então existe um arco entre o elemento de memória e a microoperação. Existe um arco entre uma microoperação e um elemento de memória, se a microoperação devolve o resultado para um elemento de memória. Um grafo de execução abstrata é dito simples (figura 2a), se possui apenas um componente fortemente conectado; do contrário ele é dito múltiplo (figura 2b). Os grafos são unidirecionais.

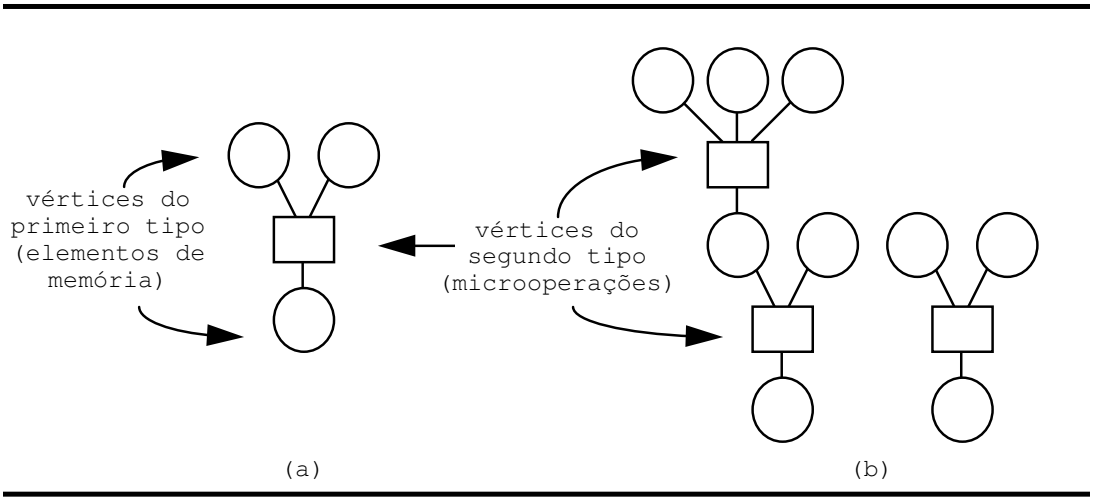


Figura 2 - Grafos de execução abstrata. (a) grafo simples, (b) grafo múltiplo.

## 3.2 Análise das instruções

Na análise dos grafos de execução abstrata das instruções de um processador, dois critérios considerados são: a complexidade, que é medida pela quantidade de hardware exercitado pela instrução (análise estrutural); e a acessibilidade, que é medida pela facilidade de acesso às informações de teste durante a execução da instrução (análise funcional).

Análise estrutural dos grafos de execução abstrata

O objetivo da análise estrutural é identificar a complexidade dos grafos. Essa análise é realizada sem considerar o significado dos vértices, ou seja, apenas a estrutura dos grafos é considerada.

Na análise estrutural, os grafos são classificados conforme sua relação de dominância estrutural. De acordo com a teoria dos grafos, uma instrução $I_1$ é estruturalmente dominada por uma instrução $I_2$, se o grafo de $I_1$ pode ser representado dentro do grafo de $I_2$, ou seja, se o grafo de $I_1$ for isomórfico em relação ao grafo da instrução $I_2$. Realizando-se uma análise estrutural dos grafos da figura 2, conclui-se que o grafo apresentado na figura 2a é estruturalmente dominado pelo grafo apresentado na figura 2b.

Os parâmetros relacionados à análise estrutural do grafo de uma instrução são: o número de arcos; o número de vértices; o número de camadas (profundidade do grafo); e o número de componentes fortemente acoplados (grau de multiplicidade do grafo).

**Análise funcional dos grafos de execução abstrata**

Nessa análise, o significado dos vértices é considerado, permitindo assim o estudo da acessibilidade das instruções. A acessibilidade de uma instrução é dada pelo grau de controlabilidade1 dos elementos de memória origem dos dados (representada por t), e pelo grau de observabilidade2 dos elementos de memória destino (representada por t'). Logo, a acessibilidade de uma instrução é representada pelo par (t, t'), onde t é o maior valor de controlabilidade entre os diferentes elementos de memória origem, e t' é o maior valor de observabilidade dos destinos.

[1] um elemento de memória possui controlabilidade i, se a informação recebida por ele precisou passar antes por um elemento de memória com controlabilidade i-1.

[2] um elemento de memória possui observabilidade j, se para acessar a informação contida nele for necessário, previamente, transferi-la para um elemento de memória com observabilidade j-1.

A acessibilidade em uma instrução é utilizada na determinação da relação de dominância funcional do grafo que a representa. Sendo assim, uma instrução com acessibilidade $(t_1, t_1')$ é funcionalmente dominada por uma instrução com acessibilidade $(t_2, t_2')$, se $(t_1, t_1') < (t_2, t_2')$, com $(a, b) < (c, d) \leftrightarrow (a \leq c \text{ e } b \leq d)$ e $(a, b) \neq (c, d)$.

### 3.3 Estratégias de teste

No artigo são descritas duas estratégias de teste: *start-small* e *start-big*. A estratégia *start-small* é utilizada em situações onde são necessárias informações a respeito da localização da falha, como por exemplo em testes de fim de fabricação, para que o fabricante possa corrigir o defeito, que pode ter ocorrido durante o processo de fabricação ou no projeto. A utilização da estratégia *start-big* não possibilita a localização da falha, sendo utilizada em situações onde necessita-se apenas detectar sua ocorrência.

Na estratégia *start-small*, o teste inicia verificando uma pequena parte do hardware. Cada teste subseqüente adiciona uma pequena quantidade de hardware às partes previamente testadas. Se um teste detecta uma falha, essa falha pertence à parte adicionada pelo teste. Para utilização dessa estratégia, é necessário previamente realizar uma ordenação no conjunto de instruções utilizado no teste. O conjunto de instruções é ordenado: por intermédio da aplicação da relação de dominância estrutural, que realiza um particionamento estrutural no conjunto de instruções dividindo-o em classes, e por intermédio da aplicação da relação de dominância funcional, que realiza um particionamento funcional no conjunto de instruções dividindo-o em blocos. Após a classificação, é possível utilizar o conjunto de instruções, na ordem especificada, com o objetivo de localizar a falha.

Na abordagem *start-big*, o teste inicia verificando uma grande parte do hardware. Se nenhuma falha for detectada, o teste continua verificando as demais partes. Caso seja detectada uma falha, o teste é suspenso e a falha sinalizada, sem necessidade de informação da unidade em que ocorreu o erro (ideal para testes do tipo passa/falha). Essa abordagem consiste em uma rápida verificação no sistema, aproveitando períodos de ociosidade. Para execução do teste é preciso determinar o conjunto mínimo de instruções que exercita, pelo menos uma vez, cada elemento de memória e cada microoperação do processador. Essa escolha é realizada em duas etapas:

1. Escolha do conjunto dominante - dado o conjunto E de todas as instruções do processador, o conjunto dominante D é definido como o subconjunto de instruções que não são estruturalmente dominadas por qualquer outra instrução de E.
2. Cobertura das microoperações e elementos de memória - se o conjunto dominante não exercita cada microoperação e cada elemento de memória pelo menos uma vez, esse conjunto é completado por um subconjunto C de instruções (que pertencem a E - D), com o objetivo de atingir uma cobertura total. O conjunto de instruções para o teste é dessa maneira composto pelo conjunto dominante D e, se necessário, pelo subconjunto de cobertura C. Uma ordem para executar as instruções de $\{D \cup C\}$ é determinada da seguinte maneira: uma instrução $I_1$ é executada antes de uma instrução $I_2$, se $(t_1, t_1') < (t_2, t_2')$. Se os valores de acessibilidade (t, t') não forem comparáveis, a ordem de execução é indiferente.

### 3.4 Algoritmos de teste

O objetivo é detectar possíveis erros (ou mau funcionamento) do processador, pela análise do comportamento lógico dos grafos de execução abstrata. O teste de uma instrução é realizado em duas etapas:

1. Verificação da identidade do grafo de execução abstrata associado à instrução (teste de conformidade); e,
2. Verificação dos nodos dos grafos (testes dos elementos de memória e das microoperações).

Para qualquer instrução, define-se o conjunto de origens como Dv(S) e o conjunto de destinos como Dv(P). Para um conjunto de instruções é definido o domínio de conformidade Dc como a união dos conjuntos de origens Dv(S) de todas as instruções que pertencem a esse conjunto; e o domínio de observação Do como a união dos conjuntos de destinos Dv(P) de todas as instruções que pertencem a esse conjunto.

**Teste de Conformidade (verificação da identidade dos grafos de execução abstrata)**

Essa verificação tem como objetivo garantir que o grafo em execução é o grafo descrito. Isso é denominado teste de conformidade. Três tipos de falhas são consideradas:

- Falhas referentes à seleção da fonte - nenhuma fonte é selecionada; uma fonte errada é selecionada; e mais que uma fonte é selecionada.
- Falhas referentes à seleção do destino - nenhum destino é selecionado; um destino errado é selecionado; e mais que um destino é selecionado.
- Falhas referentes à ativação da microoperação - nenhuma operação é executada; uma operação errada é executada; e mais que uma operação é executada.

O teste de conformidade do grafo de execução abstrata verifica, pelo menos parcialmente, a seqüencialização do processador. Esse teste verifica se os comandos de ativação dos elementos de memória e microoperações são gerados corretamente. O procedimento de teste é o seguinte:

a. Considerar um grupo de instruções cujos domínios de conformidade e observação são respectivamente Dc e Do;
b. {O1, …, OP} é o conjunto de microoperações executadas pelas instruções do grupo;
c. Para cada instrução na qual S é a fonte, P o destino e Oi a microoperação ativada, o algoritmo de teste é o seguinte:

    1. inicialização: S = X; Dc - S = Y; Dv = Y, onde X e Y são vetores de teste, e considerando-se que $Oi(X) \neq Oj(X) \ \forall \ j \neq i$, e $Oi(X) \neq Y$.
    2. execução da instrução;
    3. observação de Dv(S) e Dv(P).

**Teste dos Elementos de Memória (verificação dos vértices de primeiro tipo dos grafos de execução abstrata)**

Nesse teste pode ser verificado um conjunto de registradores considerado como uma memória global, quando possível, ou registradores independentes. Em qualquer caso, as hipóteses de falhas geralmente consideradas são as seguintes: uma ou mais células estão stuck-at 1 ou 0; uma ou mais células falham ao realizar uma transição 1-0-1 ou 0-1-0; e, existem duas ou mais células acopladas.

**Teste das Microoperações (verificação dos vértices de segundo tipo dos grafos de execução abstrata)**

As unidades de computação de um processador (ULA, contadores, …) não são verificadas como elementos de hardware. É realizada uma verificação indireta por meio do correto funcionamento de suas ações, ou seja, é realizada uma verificação funcional das suas microoperações.

As operações fornecem um resultado Z a partir de um ou diversos operandos X, Y, ... . O resultado e os operandos podem ser escritos como quantidades booleanas e qualquer operação pode ser representada por uma função booleana genérica de variáveis genéricas: Z = F(X, Y, ...). Segundo a álgebra booleana, os operandos de uma função booleana podem ser representados individualmente3, da seguinte maneira [KOH70]:

$$f(X) = \bar{x}1 \cdot f(0, x2, \ldots, xn) + x1 \cdot f(1, x2, \ldots, xn)$$

A aplicação sucessiva desse teorema permite que a função fi seja escrita da seguinte maneira:

$$f_i(X, Y, \ldots) = \sum_j \tilde{x}_i \cdot \tilde{y}_i \ \ldots \ C_i^j$$

Nessa função, o corpo é o produto das variáveis $\tilde{x}_i, \tilde{y}_i, \ldots$ (onde $\tilde{x}_i = \bar{x}_i$ ou $x_i$), e o contexto $C_i^j$ é uma função booleana das variáveis xk, yk, ..., onde $k \neq i$.

Testar as microoperações é equivalente a determinar um conjunto de valores para os operandos X, Y, … suficientes para garantir o correto funcionamento da operação. Isso é realizado verificando-se o sistema de equações que descreve a função. O procedimento proposto consiste em testar exaustivamente o corpo das funções utilizando todas as combinações possíveis dos i-ésimos bits dos operandos, e testar parcialmente o contexto C atribuindo o valor VERDADEIRO (1) e o valor FALSO (0) para cada contexto.

Para cada instrução I, três conjuntos de vetores de teste são definidos: VG que garante a identidade do grafo; VM que verifica os elementos de memória; e, VO que verifica as microoperações. O procedimento de teste consiste em três fases:

I. Inicialização das origens com um vetor de teste $T \in \{VG \cup VM \cup VO\}$;
II. Execução da instrução;
III. Observação do domínio de observação correspondente.

Em resumo, a construção de um procedimento de teste utilizando o método proposto em [ROB80] é realizada em quatro etapas:

1. construção dos grafos de execução abstrata para todas as instruções do processador;
2. análise (estrutural e funcional) dos grafos e classificação para obtenção do conjunto mínimo de instruções que exercitam todo o processador a ser testado;
3. geração do teste - utilizando uma das estratégias: start-big (quando não for necessário localizar a falha) e start-small (fornece a localização da falha);
4. construção dos algoritmos de teste, responsáveis pela verificação da identidade dos grafos (teste de conformidade) e pela verificação dos nodos dos grafos (teste dos elementos de memória e das microoperações).

[3] No caso de registradores, os elementos são suas células. Por exemplo, um registrador de n bits possui n elementos: $x_1$ .. $x_n$.

# 4 APLICABILIDADE DOS MÉTODOS EXISTENTES AO TRANSPUTER - ESTUDO DE CASO

Os métodos de teste descritos no presente relatório técnico foram desenvolvidos visando sua utilização em processadores convencionais, ou seja, processadores que seguem os preceitos básicos da arquitetura de Von Neumann. Como exemplo de processadores com essa arquitetura, pode-se citar os da família Intel (8085, 8086, 80x86), Motorola (6800, 680x0) e Zilog (Z-80, Z-8000). Esses processadores possuem uma organização interna e funcionamento bastante semelhantes no que diz respeito aos seus conjuntos de registradores, modos de endereçamento, enfim, na sua concepção.

No caso do transputer, que possui uma arquitetura significativamente diferente com relação aos processadores convencionais, os métodos de teste tradicionais não podem ser diretamente aplicados, sendo necessária a realização de alterações no método. As principais características do transputer T800 que o tornam diferente dos processadores convencionais, são a existência de uma FPU, uma memória RAM e canais de comunicação serial, integrados no mesmo chip, juntamente com a CPU e o escalonador implementado em microcódigo que possibilita a existência de processos concorrentes. Levando-se em consideração essas características, conclui-se que os métodos de teste para processadores podem ser aplicados apenas a dois módulos do transputer T800: a CPU e a FPU. Adicionalmente, para a CPU é necessário realizar alterações no método de teste escolhido, seja ele qual for, com o objetivo de adaptá-lo às características de concorrência existentes no transputer (ex. escalonador de processos) e não implementadas nos processadores convencionais citados anteriormente.

No caso da utilização do método proposto por Thatte e Abraham [THA80] para o teste do transputer, os registradores e o fluxo de instruções entre eles poderiam ser representados por intermédio do grafo de sistema. Porém, algumas situações ficariam complexas de se representar, devido ao fato do transputer possuir uma concepção muito diferente daquela para a qual o método foi proposto. Essa complexidade de representação parte do fato do transputer possuir apenas 6 registradores, e inúmeras instruções que causam transferência de dados entre eles e o mundo exterior, sendo que a maioria das instruções (senão todas) causam um fluxo de dados paralelo. Logo, uma única instrução terá que ser representada por vários arcos. Um exemplo é a instrução de carga de constante em um registrador: em um processador convencional, essa instrução seria representada por um único arco (ex. no 8086: mov AX,#4). No transputer (ex. no T800: ldc 4) são necessários vários arcos, pois a carga de um registrador causa um fluxo de dados entre os demais registradores (processo de empilhamento).

Adicionalmente, a utilização do grafo de sistema pode não ser eficiente o suficiente na representação de determinadas instruções do transputer, tais como a instrução startp (start process) utilizada para criar processos concorrentes. Ao criar um processo, a instrução startp adiciona para o fim da lista de escalonamento de processos do nível de prioridade do processo que executou a instrução, a área de trabalho do novo processo, habilitando assim a execução concorrente do novo processo com os processos existentes na lista. Utilizando as informações existentes nos manuais do transputer fica difícil de identificar os fluxos de dados que ocorrem durante a execução dessa instrução. De acordo com os manuais, para que essa instrução seja executada, é preciso que previamente o registrador Areg possua o endereço da área de trabalho do novo processo, e o registrador Breg contenha o valor do deslocamento do fim da instrução atual até a primeira instrução do novo processo. Uma possível representação para a instrução startp utilizando o grafo de sistema consta da figura 3. Nessa representação, existe um arco de Areg para o nodo OUT, um arco de Breg para Areg, um arco de Creg para Breg, e, um outro arco de Areg para o nodo OUT. Com isso, o endereço da área de trabalho do novo processo que está em Areg será colocado no final da fila de escalonamento, que se trata de uma memória representada pelo nodo OUT. Nesse instante ocorre um desempilhamento, e Areg recebe o conteúdo de Breg (deslocamento relativo ao início do código do novo processo), sendo essa operação representada pelo arco de Breg para Areg. E

finalmente, um novo arco de Areg para o nodo OUT é utilizado para representar o salvamento da informação que estava em Breg (agora está em Areg) na área de trabalho, para que posteriormente, quando o novo processo for escalonado, esse valor possa ser carregado em Iptr.

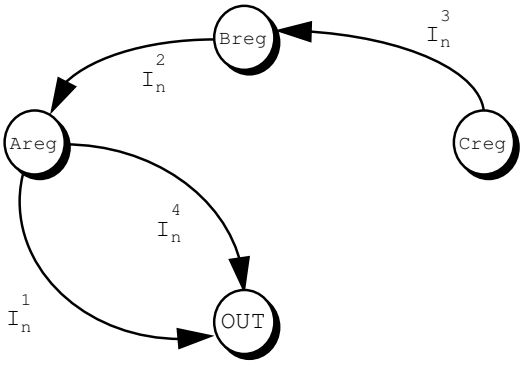


Figura 3 - Representação da instrução startp no grafo de sistema.

Por intermédio desse exemplo, pode-se verificar a ineficiência do grafo de sistema na representação de determinadas instruções complexas. No caso da instrução de criação de processos, diversos fluxos de dados devem percorrer o processador. Como exemplo, pode-se citar a identificação do nível de prioridade do processo corrente para que o novo processo possa ser colocado na fila correta. Para descobrir qual é esse nível deve existir algum fluxo de dados não representado no grafo de sistema. Infelizmente, pela documentação fornecida pelo fabricante do processador, não é possível identificar esse fluxo, ficando dessa maneira sua representação prejudicada.

Os problemas existentes para aplicação ao transputer de qualquer um dos métodos listados na tabela 1 (teste de processadores convencionais a nível de processador), são semelhantes. Porém, o método proposto por Robach e Saucier [ROB80], comparando-se com os demais, possui complexidade menor na modelagem do processador, e maior facilidade para adaptar o modelo de falhas aos blocos funcionais CPU e FPU do transputer. A complexidade menor na modelagem do processador pode ser explicada pelo fato de cada instrução ser representada em separado por intermédio dos grafos de execução abstrata. Porém, o problema com relação à representação de instruções complexas, como *startp*, citada anteriormente, continua. A maior facilidade para adaptar o modelo de falhas ao transputer está diretamente relacionada à menor complexidade na modelagem do processador.

# BIBLIOGRAFIA


[ABA83] ABADIR, M.S.; REGHBATI, H.K. Functional Testing of Semiconductor Random Access Memories. **Computing Surveys**, New York, v.15, n.3, p. 175-198, Sept. 1983.

[ABR81] ABRAHAM, J. A.; PARKER, K. Practical Microprocessor Testing: Open and Closed Loop Approaches. **IEEE COMPCON**, [S.l.], p. 308-311, Spring 1981.

[ANC86] ANCEAU, F. **The Architecture of Microprocessors.** Wokingham: Addison-Wesley, 1986. 252 p.

[AND71] ANDERSON, D.A. **Design of Self-Checking Digital Networks Using Coding Techniques.** Urbana, Illinois: Coordinated Science Laboratory, Sept. 1971.

[ANN82] ANNARATONE, M.A.; SAMI, M.G. An Approach to Functional Testing of Microprocessors. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 12., June 1982, Santa Monica. **Proceedings...** [S.l.]: IEEE, 1982. p.158-164.

[AVR87] AVRESKY, D. R. et al. An Approach to Fault Diagnosis in Multimicrocomputer Systems: Algorithm and Simulation. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 17., July 1987, Pittsburgh. **Proceedings...** New York: IEEE, 1987. p.305-310.

[BAN86] BANERJEE, P.; ABRAHAM, J.A. Concurrent Fault Diagnosis in Multiple Processor Systems. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 16., July 1986, Viena. **Proceedings...** New York: IEEE, 1986. p.298-303.

[BAN86a] BANERJEE, P.; ABRAHAM, J.A. Bounds on Algorithm-Based Fault Tolerance in Multiple Processor Systems. **IEEE Transactions on Computers**, New York, v.C-35, n.4, p.296-306, Apr. 1986.

[BER88] BERNSTEIN, P. A. Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing. **Computer**, New York, p.37-45, Feb. 1988.

[BEZ94] BEZERRA, E.A. **Definição de Grafos de Execução Abstrata para Realização de Teste Funcional no Transputer.** Porto Alegre: CPGCC da UFRGS, 1994. 76p. (Trabalho Individual, 386).

[BEZ95] BEZERRA, E.A.; JANSCH-PÔRTO, I. Procedimento de Teste para Detecção de Falhas no Transputer. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 6., 1995, Canela. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p.201-219.

[BEZ95a] BEZERRA, E.A.; JANSCH-PÔRTO, I. A Minimal Test Set for the Transputer Processor. In: CONGRESS OF THE BRAZILIAN MICROELECTRONICS SOCIETY, 10., 1995, Canela. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p.722.

[BLO90] BLOUGH, D.M.; MASSON, G.M. Performance Analysis of a Generalized Concurrent Error Detection Procedure. **IEEE Transactions on Computers**, New York, v.39, n.1, p.475-485, Jan. 1990.

[BLO92] BLOUGH, D.M.; SULLIVAN, G. F.; MASSON, G. M. Efficient Diagnosis of Multiprocessor Systems under Probabilistic Models. **IEEE Transactions on Computers**, New York, v.41, n.9, p.1126-1136, Sept. 1992.

[BLO93] BLOUGH, D.M.; PELC, A. Diagnosis and Repair in Multiprocessor Systems. **IEEE Transactions on Computers**, New York, v.42, n.2, p.205-217, Feb. 1993.

[BRA84] BRAHME, D.; ABRAHAM, J.A. Functional Testing of Microprocessors. **IEEE Transactions on Computers**, New York, v.C-33, n.6, p.475-485, June 1984.

[BRE76] BREUER, M.A.; FRIEDMAN, A.D. **Diagnosis & Reliable Design of Digital Systems.** Woodland Hills: Computer Science Press, 1976. 308p.

[BUT88] BUTZERIN, T.; SAMAD, A.; ARCHAMBEAU, E. Asic Testing - with High Fault Coverage. **VLSI Systems Design**, Palo Alto, CA, p.50-57, Sept. 1988.

[CAS92] CASTRO, H.S.; GOUGH, M.P. Mars94: A Fault-Tolerant Multi-Transputer Array for Space Applications. In: TRANSPUTERS'92: ADVANCED RESEARCH AND INDUSTRIAL APPLICATIONS, 1992, Amsterdan. **Proceedings...** Amsterdan: IOS Press, 1992. p.284-292.

[CAS92a] CASTRO, H.S. **Fault Tolerance Through Reconfigurability:** Applications in Space Instrumentation. Brighton: University of Sussex, School of Engineering, June 1992. 173p. Tese de Doutorado.

[CAV95] CASTRO ALVES, V. Functional Test of Single and Multi-Port SRAMs. In: CONGRESS OF THE BRAZILIAN MICROELECTRONICS SOCIETY, 10., 1995, Canela. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p.169-188.

[CHE76] CHEN, W.K. **Applied Graph Theory - Graphs and Electrical Networks.** Amsterdam: North-Holland, 1976. 542p.

[CRI91] CRISTIAN, F. Understanding Fault-Tolerant Distributed Systems. **Communications of the ACM**, New York, v.34, n.2, p.57-78, Feb. 1991.

[DIJ75] DIJKSTRA, E.W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. **Communications of the ACM**, New York, v.18, n.8, p.453-457, Aug. 1975.

[FED84] FEDI, X.; DAVID, R. Experimental Results from Random Testing of Microprocessors. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 14., June 1984, Kissimmee, Florida. **Proceedings...** New York: IEEE, 1984. p.225-230.

[FLY66] FLYNN, M.J. Very High-Speed Computing Systems. **Proceedings of the IEEE**, New York, v.54, p.1901-1909, Dec. 1966.

[FRE84] FRENZEL, J.F.; MARINOS, P.N. Functional Testing of Microprocessors in a User Enviroment. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 14., June 1984, Kissimmee, Florida. **Proceedings...** New York: IEEE, 1984. p.219-224.

[FRI80] FRIEDMAN, A.D.; SIMONCINI, L. System-Level Fault Diagnosis. **Computer**, New York, p.47-53, Mar. 1980.

[FUJ85] FUJIWARA, H. **Logic Testing and Design for Testability.** Cambridge: MIT Press, 1985. 284p.

[GOR80] GORSLINE, G.W. **Computer Organization:** Hardware/Software. Englewood Cliffs: Prentice-Hall, 1980. 309 p.

[HAY78] HAYES, J.P. **Computer Architecture and Organization.** New York: MacGraw-Hill, 1978. 498 p.

[HAY85] HAYES, J.P. Fault Modeling. **IEEE Design and Test of Computers**, New York, v.2, n.2, p.88-95, Apr. 1985.

[HOA78] HOARE, C.A.R. Communicating Sequential Processes. **Communications of the ACM**, New York, v.21, n.8, p.666-677, Aug. 1978.

[HUA84] HUANG, K.H.; ABRAHAM, J.A. Algorithm-Based Fault Tolerance for Matrix Operations. **IEEE Transactions on Computers**, New York, v. C-33, n.6, p.518-528, June 1984.

[INM84] INMOS. **IMS T414.** Bristol: INMOS Limited, 1984. 31p. (Preliminary data).

[INM87] INMOS. **The Transputer Instruction Set:** A Compiler Writers' Guide. Bristol: INMOS Limited, 1987. 161p.

[INM88] INMOS. **The Transputer Databook.** Bath: Bath, 1988. 477p.

[INM88a] INMOS. **Transputer Development System.** New York: Prentice Hall, 1988. 491p.

[INM88b] INMOS. **Occam2:** Reference Manual. New York: Prentice Hall,1988. 133p.

[INM89] INMOS. **The Transputer Applications Notebook:** Architecture and Software. Bristol: INMOS Limited,1989. 234p.

[INM89a] INMOS. **3L Parallel C IMS D711 Delivery Manual.** Bristol: INMOS Limited, 1989. 271p.

[INM90] INMOS. **IMS B008:** User Guide and Reference Manual. Bristol: INMOS Limited, 1990. 104p.

[INM90a] INMOS. **S708 User Guide.** Bath: Bath, 1990. 84p.

[INM91] INMOS. **The T9000 Transputer Products Overview Manual.** Phoenix: INMOS Limited, 1991. 194p.

[INT85] INTEL. **Microcontroler Handbook.** Santa Clara: Intel Corporation, 1985.

[INT87] INTEL. **Microprocessor and Peripheral Handbook:** Vol. I - Microprocessor. Santa Clara: Intel Corporation, 1987.

[JOH84] JOHNSON, D. The Intel 432: a VLSI Architecture for Fault-Tolerant Computer Systems. **Computer**, New York, v.17, n.8, p.40-48, Aug. 1984.

[KEB92] KEBICHI, O.; NICOLAIDIS, M. A Tool for Automatic Generation of BISTed and Transparent BISTed RAMs. In: EUROPEAN DESIGN AUTOMATION CONFERENCE, Feb. 1992. **Proceedings...** Paris: IEEE, 1992. p.570-575.

[KIT94] KITAJIMA, J.P. **Modèles Quantitatifs d'Algorithmes Parallèles.** Grenoble: Institut National Polytechnique de Grenoble, 1994. 182p. Tese de Doutorado.

[KIT95] KITAJIMA, J.P. **Ferramenta ANDES.** Porto Alegre, Brasília, 3 set. 1995. (Informação por Correio Eletrônico. InterNet Username: kita@guarany.cpd.unb.br).

[KOH70] KOHAVI, Z. **Switching and Finite Automata Theory.** Bombay: Tata McGraw-Hill, 1992. 232p.

[KUH86] KUHL, J.G.; REDDY, S.M. Fault-Tolerance Considerations in Large, Multiple-Processor Systems. **Computer**, New York, p.56-67, Mar. 1986.

[KUK94] KUMAR, K.V.; CHANDRA, V. Transputer-Based Fault-Tolerant and Fail-Safe Node for Dual Ring Distributed Railway Signalling Systems. **Microprocessors and Microsystems**, Trowbridge, v.18, n.3, p.141-150, Apr.1994.

[KUM93] KUMAR, R.K.; SINHA, S.K.; PATNAIK, L.M. A Fault-Tolerant Multi-Transputer Architecture. **Microprocessors and Microsystems**, Trowbridge, v.17, n.2, p.75-81, Mar.1993.

[KUM94] KUMAR, R.K. **Transputer Test.** Porto Alegre, Bangalore, 26 out. 1994. (Informação por Correio Eletrônico. Rede InterNet. Username: vidyut!rkkum@vigyan.iisc.ernet.in).

[KUO90] KUO, N.H.; GOUGH, M.P. 3D Parity Checking Models for errors in RAM Memories Used in Space On-board Computers. **Microprocessors and Microsystems**, Trowbridge, v.14, n.9, p.599-605, Nov.1990.

[LAI83] LAI, K.W.; SIEWIOREK, D.P. Functional Testing of Digital Systems. In: DESIGN AUTOMATION CONFERENCE, 20., June 1983, Miami Beach. **Proceedings...** Miami:

IEEE, 1983. p. 207-213.

[LAP85] LAPRIE, J.C. Dependable computing and fault-tolerance: concepts and terminology. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 15., 1985, New York. **Proceedings...** New York: IEEE, 1985. p.2-11.

[LOM95] LOMBARDI, F. Interconnect Diagnosis. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 6., 1995, Canela. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p.3.

[MAE93] MARTINS, E. Validação Experimental da Tolerância a Falhas: A Técnica de Injeção de Falhas. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 5., 1993, São José dos Campos. **Anais...** São José dos Campos: INPE, 1993. p.56-70.

[MAR82] MARINESCU, M. Simple and Efficient Algorithms for Functional RAM Testing. In: IEEE INTERNATIONAL TEST CONFERENCE, Nov. 1982. **Proceedings...** [S.l.]: IEEE, 1982.

[MOT88] MOTOROLA. **M68000 Family Reference.** USA: Motorola, 1988. 415p.

[MUK94] MUKHERJEE, A.M.; TYRRELL, A.M. Investigating the Effects of Induced Faults in Transputer Systems. **Microprocessors and Microsystems**, Trowbridge, v.18, n.3, p.151-163, Apr.1994.

[NAI92] NAIR, V.S.S.; HOSKOTE, Y.V.; ABRAHAM, J.A. Probabilistic Evaluation of On-Line Checks in Fault-Tolerant Multiprocessor Systems. **IEEE Transactions on Computers**, New York: IEEE, v.41, n.5, p.532-541, May 1992.

[NIC88] NICOLE, D.A. **Reconfigurable Transputer Processor Architecture.** Southampton: Southamptom Transputer Support Centre, 1988. 18p. (Esprit Project 1085, Technical Report, n.2).

[NIC94] NICOLE, D.A. et al. **Southampton's Portable Occam Compiler (SPOC).** Southampton: Southamptom Transputer Support Centre, 1994. 30p. (http://www.hensa.ac.uk/parallel/occam/compilers/spoc/).

[NIM92] NICOLAIDIS, M. **Transparent BIST for RAMs.** Grenoble: TIMA/INPG, Jan. 1992. (Technical Report).

[NUN93] NUNES, R. C.; NAVAUX P.O.; JANSCH-PÔRTO I. Algoritmo de Reconfiguração na Máquina T-NODE em Caso de Falhas. SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES - PROCESSAMENTO DE ALTO DESEMPENHO, 5., 1993, Florianópolis. **Anais...** Florianópolis: SBC, 1993. p. 344-357.

[NUN93a] NUNES, R.C. **Reconfiguração no T-NODE em Caso de Falhas.** Porto Alegre: CPGCC da UFRGS, 1993. 117p. Dissertação de mestrado.

[NUN93b] NUNES, R.C.; JANSCH-PÔRTO, I. e NAVAUX, P.O. Estratégias de Reconfiguração sob Falhas para Multiprocessadores: Aplicação à Máquina T-NODE. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 5., 1993, São José dos Campos. **Anais...** São José dos Campos: INPE, 1993. p.80-95.

[OGD78] OGDIN, C.A. **Microcomputer Design.** Englewood Cliffs: Prentice-Hall, 1978. 190p.

[PAU95] PAULA, A.R. Aspectos de Tolerância a Defeitos do Sistema de Computação do Primeiro Microssatélite de Aplicações Científicas do INPE. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 6., 1995, Canela. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p.63-75.

[PAR86] PARKER, K.P. Testability: Barriers to Acceptance. **IEEE Design & Test of Computers**, Ney York, v. 3, n. 5, p.11-15, Oct. 1986.

[PRE67] PREPARATA, F.P.; METZE, G.; CHIEN, R.T. On The Connection Assignment Problem of

Diagnosable Systems. **IEEE Transactions on Electronic Computers**, New York, v. EC-16, p. 848-854, Dec. 1967.

[RAB95] RABAGLIATI, A. **Reliable Transputer RAM Test - MTEST.** Porto Alegre, Colorado Springs, 17 Apr. 1995. (Informação por Correio Eletrônico. Rede InterNet. Username: andyr@wizzy.com).

[RAI90] RAI, S.; AGRAWAL, D.P. **Advances in Distributed System Reliability.** Los Alamitos: IEEE Computer Society Press, 1990. 333p.

[ROB80] ROBACH, C.; SAUCIER, G. Microprocessor Functional Testing. In: IEEE TEST CONFERENCE, Nov. 1980, Cherry Hill, **Proceedings...** [S.l.:s.n.], 1980. p.433-443.

[RUG89] RUSSEL, G.; SAYERS, I.L. **Advanced Simulation and Test Methodologies for VLSI Design.** London: Van Nostrand Reinhold, 1989. 378p.

[RUS75] RUSSEL, J.D.; KIME, C.R. System Fault Diagnosis: Masking, Exposure and Diagnosability without Repair. **IEEE Transactions on Computers**, New York, v.C-24, p.1155-1161, Dec. 1975.

[RUS75a] RUSSEL, J.D.; KIME, C.R. System Fault Diagnosis: Closure and Diagnosability with Repair. **IEEE Transactions on Computers**, New York, v.C-24, p.1078-1088, Nov. 1975.

[SAL92] SALINAS, J.; LOMBARDI, F. A Data Path Approach for Testing Microprocessors with a Fault Bound: the MC68000 Case. **Microprocessors and Microsystems**, Oxford, v.16, n.10, p.529-539, 1992.

[SCH86] SCHUETTE, M. A. et al. Experimental Evaluation of Two Concurrent Error Detection Schemes. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 16., July 1986, Viena. **Proceedings...** New York: IEEE, 1986. p.138-143.

[SHD90] SHEPHERD, D. Verified Microcode Design. **Microprocessors and Microsystems**, Oxford, v.14, n.10, p.623-630, Dec. 1990.

[SHE84] SHEN, L.; SU, S.Y.H. A Functional Testing Method for Microprocessors. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 14., June 1984, Kissimmee, Florida. **Proceedings...** New York: IEEE, 1984. p.212-218.

[SIE82] SIEWIOREK, D.P.; SWARZ, R.S. **The Theory and Practice of Reliable System Design.** Bedford: Digital Press, 1982. 772 p.

[SIT93] SITARAMAN, R.K.; JHA, N.K. Optimal Design of Checks for Error Detection and Location in Fault-Tolerant Multiprocessor Systems. **IEEE Transactions on Computers**, New York, v.42, n.7, p.780-793, July 1993.

[TEL91] TELMAT INFORMATIQUE. **T-NODE Hardware Manual.** Soultz: Telmat Informatique, 1991.

[THA80] THATTE, S.M.; ABRAHAM, J.A. Test Generation for Microprocessors. **IEEE Transactions on Computers**, New York, v.C-29, n.6, p.429-441, June 1980.

[THO91] THOMPSON, H.A. Transputer-Based Fault Tolerance in Safety-Critical Systems. **Microprocessors and Microsystems**, Oxford, v.15, n.5, p.243-248, June1991.

[TOR94] TORII, T.; CHELIAN, M.S. A New Fault-Tolerant Multi-Transputer Configuration for Avionics Two-Lane Systems. **Microprocessors and Microsystems**, Trowbridge, v.18, n.7, p.371-376, Sept.1994.

[VEL82] VELAZCO, R. **Test Comportemental de Microprocesseurs.** Grenoble: Institut National Polytechnique de Grenoble, 1982. 245p. Tese de Doutorado.

[VIS87] VISWANADHAM, N.; SARMA, V.V.S.; SINGH, M.G. **Reliability of Computer and Control Systems.** Amsterdam: North-Holland, 1987. 466p. (North-Holland Systems and

Control Series, v.8).

[WAG88] WAGNER, F.R. et al. **Métodos de Validação de Sistemas Digitais.** Campinas: UNICAMP, 1988. 199p.

[WAK78] WAKERLY, J.F. **Error Detecting Codes, Self-Checking Circuits and Applications.** Netherlands: Elsevier Science, 1978. 231p.

[WEB87] WEBER, R.F. Teste de Sistemas Digitais no Nível de Comportamento. CONGRESSO NACIONAL DE INFORMÁTICA, 20., 1987, São Paulo. **Anais...** São Paulo: SUCESU, set. 1987. p.1234-1239.

[WET90] WEBER, T.S. et al. Fundamentos de Tolerância a Falhas. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 10.; JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 9., 22-27 jul. 1990, Vitória. **Apostila...** [S.l.]:SBC, 1990. 75p.

[WEY92] WEYERER, M.; GOLDEMUND, G. **Testability of Electronic Circuits.** Englewood Cliffs: Prentice Hall, 1992. 232p.

[WIL82] WILLIAMS, T.W.; KENNETH, P.P. Design for Testability - A Survey. **IEEE Transactions on Computers**, New York, v.C-31, n.1, p.2-15, Jan. 1982.

[WOO95] WOOD, D. et al. **Kent Retargetable Occam Compiler (KROC).** Canterbury: University of Kent at Canterbury, 1995. 12p. ("Occam For All" project, http://www.hensa.ac.uk/parallel/occam/projects/occam-for-all/kroc/).

[YAN86] YANG, C.L.; MASSON, G.M. An Efficient Algorithm for Multiprocessor Fault Diagnosis Using the Comparison Approach. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 16., July 1986. **Proceedings...** Viena: IEEE, 1986. p.238-243.

[ZIS78] ZISSOS, D. **System Design with Microprocessors.** London: Academic Press, 1978. 202p.