社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2004－SLDM－117（23）
2004／12／2

パイプラインプロセッサ自己テストのための命令テンプレート生成

横山真也† 神戸和子† 井上美智子† 藤原秀雄†

†奈良先端科学技術大学院大学　情報科学研究科　〒630-0192 けいはんな学研都市

E-mail: †{shinya-y,kazuk-ka,kounoe,fujiwara}@is.naist.jp

あらまし　現在，プロセッサの構成方式としてパイプライン方式が主流となっている．そのような大規模かつ高速なプロセッサに対して，命令列の実行による自己テスト法が注目されている．この方式は，プロセッサがもつ命令列を実行してプロセッサのテストを行うので，ハードウェアオーバヘッドも遅延オーバヘッドもないという利点をもつ．本稿では，パイプラインプロセッサ自己テストのためのテストプログラムを生成する際に有効な命令テンプレートの生成手法を提案する．提案する手法では，パイプラインプロセッサ特有の構造や動作に着目し，命令テンプレートを生成する．テンプレートに基づきテストプログラムを生成することで，高品質なテストを生成できる．

キーワード　パイプラインプロセッサ，自己テスト，命令テンプレート，テストプログラム，階層テスト生成，

Efficient Generation of Instruction Templates for Pipeline Processor Self-Test

Shinya YOKOYAMA† Kazuko KAMBE† Michiko INOUE† Hideo FUJIWARA†

† Graduate School of Information Science, Nara Institute of Science and Technology

Keihanna Science City, 630-0192 Japan

E-mail: †{shinya-y,kazuk-ka,kounoe,fujiwara}@is.naist.jp

Abstract　Nowadays, pipeline processors are getting mainstream as processor structure. As testing methodology of such large-scale and high-speed processors, instruction-based self-test is getting much attention. As this methodology tests a processor by executing instructions of the processor, it has advantages of not causing hardware or delay overhead. In this paper, we propose an efficient generation methodology of instruction templates that are effective in test program generation for pipeline processor. Our method generates instruction templates paying attention to characteristic structure and behavior of pipeline processor. High-quality test is guaranteed by generating test program based on our instruction templates.

Keyword　Pipeline Processor, Self-Testing, Instruction Templates, Test Program, Hierarchical Test Generation,

1. まえがき

近年，VLSI技術の発展によりプロセッサの高集積化及び大規模化が可能になった．プロセッサの構成方式は、パイプライン方式が主流になり，処理速度もますます高速になってきている．

現在，プロセッサのテスト方式として完全スキャン方式が広く利用されている．完全スキャン方式では，プロセッサ内部のフリップフロップ(以下FF)を，外部ピンから直接制御/観測する機構を埋め込みテストする．そのため，付加回路によるハードウェアオーバヘッド，及び遅延オーバヘッド等の問題が生じる．また，プロセッサのように入出力ピン数が多い大規模なLSIを，実動作時と同様の高速テストを行うには，高価なテスタが必要になる．外部テスタを必要としない方式として，組み込み自己テスト(Built-In Self-Test)が提案されている．この方式は，テストパタン生成器と応答解析器を内部に組み込む．しかし，テストパタンを擬似ランダム的に発生させるため，一般的には高い故障検出率を保証できない．高い故障検出率を得るには，プロセッサの設計変更をする必要があり、完全スキャン方式と同様にハードウェアオーバヘッドや遅延オーバヘッドが大きくなってしまう．その他にも，ランダムテストパタンの適用が電力消費の増加に繋がるなど，多くの解決すべき問題が残されている．

こういった背景から，プロセッサのもつ命令を用いる自己テスト方式が重要となってきている[1][2][3]．この方式では，テストプログラムの実行によりプロセッサのテストを行うため，プロセッサの処理速度でテスト可能である．したがって，遅延オーバヘッドは生じない．また，完全スキャン，BISTのような付加回路を必要としないため，ハードウェアオーバヘッドもない。コスト面においても，高価な外部テスタを必要とせず，低コストであると言える．その上，通常動作と同様の電力消費でテストが可能であるため，消費電力が増加する問題は生じない．

筆者らは，この自己テスト方式に着目し，自己テストのためのテストプログラムを，階層テスト生成法に基づき生成する方法について研究を進めている．階層テスト生成では，プロセッサをモジュール分割し，ゲートレベルでモジュール単体に対してテスト生成を行う．テストパタンの正当化及び応答の観測は命令レベルで行う．

本稿では，パイプラインプロセッサを対象とし，テストプログラムを生成する際に有効なテンプレートの生成手法を提案する．パイプラインプロセッサは，処理機能サイクルがパイプラインという複数ステージに分割され，ステージごとに独立処理できるハードウェア機構を装備している．提案手法は，パイプラインプロセッサ特有の構造や動作に着目し，テストモジュールが存在するステージに対して命令を選択する．また，テンプレートで正当化できるモジュールの入力空間を考慮することで、検出集合の異なる複数のテンプレートを生成する．

以下，第2章では関連研究について述べる．第3章では本研究で扱うパイプラインプロセッサモデルを示す．第4章で命令テンプレートに基づく自己テストについて簡単に述べる．第5章では命令テンプレート生成の概要について述べる．第6章で実験結果を示し，第7章でむすびとする．

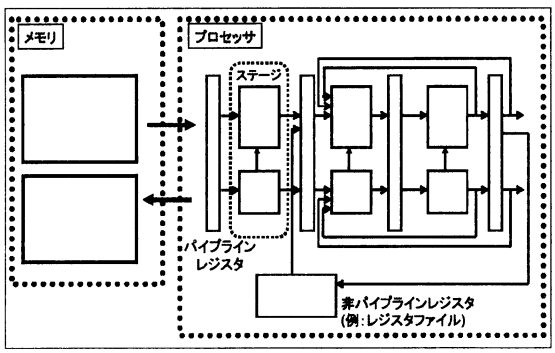


図 1: パイプラインプロセッサモデル

2. 関連研究

文献[4]では，プロセッサのRTL記述から，構成要素をfunctional, control, hiddenの3つのクラスに分割し，サイズの大きい構成要素からテストプログラムを生成している．テストプログラムは，各モジュールに対して，そのモジュールを活性化する演算を用いている．この方式はfunctionalクラスの，規則性のある構造をもつ機能モジュールのテストには有効であるが，control, hiddenクラスのモジュールについては詳しく述べられていない．また，この手法はパイプラインプロセッサを対象として特に意識しておらず，適応が難しい．

文献[5]では、モジュールにパターンを正当化し，応答を観測する命令列テンプレートを生成し，テンプレートで印加可能な値を制約としてテスト生成を行っている．しかし，制約を抽出するテンプレート数が限られているため，検出可能な故障に対してテスト生成しない可能性がある．また，テンプレートは人手で抽出している．

文献[6]は，レジスタ間のデータ転送について命令の実行順序に依存関係があることに着目し，メモリからモジュールの入力にテストパタンを伝播する命令系列を自動で抽出している．また、テンプレートで正当化できるモジュールの入力空間を考慮することで，検出故障集合の異なる複数のテンプレートを生成している．この手法は，非パイプラインプロセッサには有効であるがパイプラインプロセッサには対応していない．

3. パイプラインプロセッサモデル

図1に本研究で扱うパイプラインプロセッサモデルを示す．プロセッサ外部にメインメモリがあり，プロセッサが実行する命令の書き込みや，出力応答の観測をこのメインメモリで行う．プロセッサ内部は，データパス部と制御部で構成されており，制御部からデータパス部に制御信号が出される．また，処理機能サイクルがパイプラインという複数ステージに分割され，ステージごとに独立処理できるハードウェア機構を装備している．本稿では，この分割されたステージに着目した手法を提案する．

パイプライン処理においては，ハザードと呼ばれる，次のクロックサイクルで次の命令を実行できない事態が起こりうる．通常，データフローや制御フローは処理サイクルの順序通りに流れる．しかし，ハザードを検出したり解消したりするために，本来ならば後になるような，次ステージに進んでいる命令の演算結果や制御情報の受け渡しを，次ステージから先送りできるようなハードウェア機構を備えている．

プロセッサ内部には，ステージごとに対応するパイ

プラインレジスタが設けられている．パイプラインレジスタは、クロックごとに値が更新される．パイプラインレジスタに対して，非パイプラインレジスタが存在する．例えば，レジスタファイルである．これは，命令の実行によって陽に値を更新しようとしない限り，現在の値を保持している．また，各レジスタは機能的にサブレジスタに分割できる．

4. 命令テンプレートに基づく自己テスト

命令による制約下でモジュールのテストを行うには正確な制約を求めることが重要であるが，これは困難な問題である．そこで提案手法では命令テンプレートに基づいて制約を求める方法を採用する．命令テンプレートを利用することで，冗長故障に対するテスト生成及び命令列生成を回避できる．

図２に、本研究におけるテストプログラム生成の全体の流れを示す．**ステップ１**では，命令テンプレート生成において，テスト対象モジュールの入力にテストパタンを正当化，テストパタンを印加，及び出力応答を観測する命令などの選択の尺度である可制御性と可観測性について計算する．**ステップ２**では，選択したモジュールに対して命令テンプレートを生成し，命令テンプレートによる制約下でモジュールのテスト生成を行う．生成されたテストパタンから命令テンプレートのオペランドを決定し，テストプログラムを生成する．そのテストプログラムを実際にプロセッサに実行させ,全故障を故障シミュレーションし,故障検出率，故障検出効率を求める．上記をすべてのモジュールに対して行う．本稿では，**ステップ１**の可制御性と可観測性の計算法と，**ステップ２**のモジュールの入力空間を考慮した命令テンプレート生成手法を提案する．

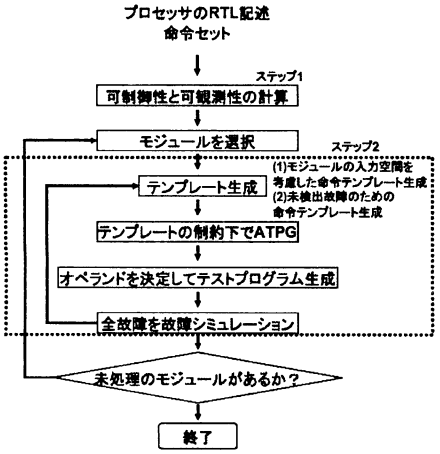


図 2: テストプログラム生成全体の流れ

5. 命令テンプレート生成

本章では，命令テンプレートを生成する際に命令選択の尺度となる可制御性と可観測性について述べ，命令テンプレート生成の概要を示す．

5.1. 命令テンプレートと隣接レジスタ

命令テンプレートはモジュールの入力に対するテストパタンの正当化，テストパタンの印加，応答を観測する命令列から成る．各命令のオペランドは未決定である．図３は，DLXs/pⅡプロセッサ[7]の ALU をテストする命令テンプレートの例である．5 行目の ADD 命令の加算処理に必要な値を，1 行目から 4 行目までの LHI 命令及び ADD.I 命令でレジスタファイル(以下 RF)に格納する．5 行目の ADD 命令で，テストパタンが ALU に印加され，演算結果は RF に格納される．6 行目と 7 行目の LHI 命令と ADD.I 命令でメモリのアドレス計算を行う．最後に 8 の SW 命令で演算結果が格納された RF の値を観測する．図 3 では，オペランドが記述されているが，これは決定したものではなく，レジスタ番号が等しい命令はテストプログラム生成時に同じレジスタを用いるという情報を示したものである．

モジュールの入力に組合せ回路だけで接続しているレジスタをそのモジュールの隣接レジスタと呼ぶ．モジュールの入力に対するテストパタンの正当化は，隣接レジスタの値を正当化することである．応答観測についても同様に考えることができ，誤りが伝播する隣接レジスタの値を観測するために隣接レジスタからメモリへの経路を探索する．

パイプラインプロセッサにおいて，テスト対象モジュールの隣接レジスタはパイプラインレジスタのサブレジスタとなる．パイプラインレジスタはクロックごとに値が更新されるので，同じパイプラインレジスタ内の各サブレジスタを違うタイミングで正当化できない．また，正当化のタイミングはパイプラインレジスタが対応するステージに関連している．したがって提案手法では,隣接レジスタが存在するステージに対し，テストパタンを印加する命令を選択する．

```
テンプレート
1:LHI    R1,x1
2:ADD.I R2,R1,x2
3:LHI    R3,x3
4:ADD.I R4,R3,x4
5:ADD   R5,R2,R4
6:LHI    R6,x5
7:ADD.I R7,R6,x6
8:SW     R7(R0),R5
```

図 3: 命令テンプレート例

5.2. 可制御性と可観測性

本研究では，パイプラインレジスタの値を正当化するために，テスト対象モジュールの隣接レジスタが存在するステージに対して命令を選択する．その際，選択可能な命令が複数ある場合に，尺度として可制御性と可観測性を利用する．

5.2.1. 局所可制御性

可制御性を考えるために，まず，各レジスタの局所可制御性を考える．局所可制御性とは，各命令を単独で実行した場合，その命令によってプロセッサ内部の各レジスタに，どのような値を設定できるかという尺度を，プロセッサ内部の非パイプラインレジスタ及び外部入力に基づいて評価したものである．命令セットから，その命令がレジスタ間にどのようなデータ転送を起こすかということを抽出することによって，非パイプラインレジスタの局所可制御性を求める．また，パイプラインレジスタの局所可制御性は，データ転送の情報を命令セットから抽出し，その命令の実行ではどのような制御情報が出されるかを RTL 記述から抽出することによって求める．

レジスタまたはサブレジスタに対し，各命令における以下の局所可制御性を考える．

- C_g: レジスタに任意の値を設定できる
(レジスタのビットすべてが C_g であれば，そのレジスタには完全可制御性があるという)
- C_c(const): 定数 const しか設定できない
- C_d(R): 非パイプラインレジスタ R に依存した値を設定できる
- －: その命令ではレジスタに値を設定できない

図 4 に，DLX/pⅡプロセッサのパイプラインレジスタと非パイプラインレジスタの局所可制御性の例を示す．図(a),図(b)がレジスタファイルの局所可制御性の例である．LHI 命令では，上位 16 ビットに可制御性があり，下位 16 ビットはすべて 0 という定数しか制御できないということを示している．ADD.I 命令では，上位 16 ビットがレジスタファイルの 0 ビット目から 15 ビット目に依存し，下位 16 ビットには可制御性がある．図(c)はパイプラインレジスタの局所可制御性の例である．ADD 命令では，0 ビット目から 31 ビット目，32 ビット目から 53 ビット目はどちらもレジスタファイルに依存しており，54 ビット目から 59 ビット目は可制御性がある．次の 1 ビットは制御できず，残りのビットは定数しか制御できないことを示している．このように，局所可制御性を，非パイプラインレジスタと外部入力に基づいて評価する．

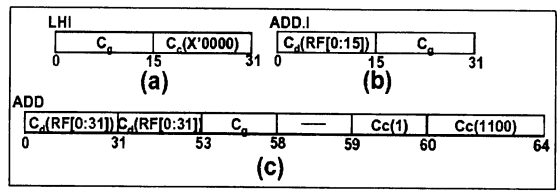


図 4: レジスタの局所可制御性の例

5.2.2. 可制御性

局所可制御性を用いて，外部入力だけに基づいた可制御性を求める．可制御性の定義は前述の C_g と C_c の 2 つである．

まず，非パイプラインレジスタ R の可制御性を，局所可制御性を組み合わせて以下のように求める．

1. R のすべてのビットを未解決とする
2. 未解決ビットの局所可制御性において，C_c が最小の命令のうち，C_g が最大である命令を選択(但し，未解決のビット数が減少しなければ，その命令を除き，選択をやり直す)
3. R のすべてのビットが C_g，C_c になるまで 2 を繰り返す
4. 選択した逆順の命令列を生成

例えば，DLX/pⅡプロセッサは，レジスタファイルが 32 ビットあり，未解決ビットは 32 ビットから始まる．C_c が最小の命令のうち，C_g が最大の命令を探索すると，ADD.I 命令が選択される．図 4 で示したように，ADD.I 命令は上位 16 ビットがレジスタファイルに依存しているので，未解決ビットは上位 16 ビットとなる．上位 16 ビットを探索空間として命令を探すと，LHI 命令が選択される．LHI 命令は上位 16 ビットに C_g を持つ．これで未解決ビットはなくなり，命令を選択した逆順に実行すれば，レジスタファイルのすべてのビットが C_g になり，DLX/pⅡプロセッサのレジスタファイルは完全可制御性を持つといえる．

非パイプラインレジスタ R に依存して評価されていた可制御性 C_d を，非パイプラインレジスタの可制御性に置き換えることによって，すべてのレジスタにおいて，外部だけに基づく可制御性が求まる．

本研究では，可制御性の評価時にパイプラインハザードを考慮しない．前述のように，パイプラインプロセッサにはハザードを検出したり解消したりするハードウェア機構が装備されている．これらは，ハザードが生じない場合と同じ効果のある実行を，時間のロスを最小限にするための仕組みである．したがって可制御性の評価時にパイプラインハザードは影響がないと考える．

5.2.3. 可観測性

可観測性を考えるために，まず，局所可観測性を考える．局所可観測性とは，各命令の実行により，プロセッサ内部のレジスタの値をどの程度観測できるかという尺度をプロセッサ内部の非パイプラインレジスタと外部出力に基づいて評価したものである．局所可観測性を以下のように定義する．

- **(O_g，{制御が必要なレジスタの集合})**: そのレジスタを観測するために制御が必要なレジスタが制御できれば任意の値を観測できる
- **(O_d(R)，{制御が必要なレジスタの集合})**: そのレジスタを観測するために制御が必要なレジスタを制御でき，非パイプラインレジスタRが観測できれば観測できる
- -: その命令ではレジスタの値を観測できない

また，可観測性は直接可観測性と間接可観測性に分けられる．直接可観測性とは，値を直接観測できるという可観測性であり，間接可観測性とは，そのレジスタに関連する信号線やレジスタの値から間接的に観測できるという可観測性である．直接可観測性の場合は，O_g,O_d を DO_g,DO_d と表記し，間接可観測性の場合は IO_g,IO_d と表記する．

次に可観測性を求めるために，可観測性依存グラフ(Obsaervability Dependence graph:ODグラフ)を利用する．図5にODグラフの例を示す．ODグラフは，ノードがレジスタ，及び外部出力に対応する有向グラフである．レジスタRに O_g があれば，Rから外部出力に有向辺を持ち，レジスタRに O_d(R')があれば，RからR'に有向辺を持つ．また，各辺は可観測性に対応する命令の情報をラベルとして持つ．ODグラフに対し，外部出力までの経路を求め，外部出力に基づいて評価された可観測性を求める．

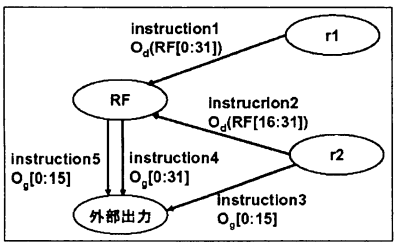
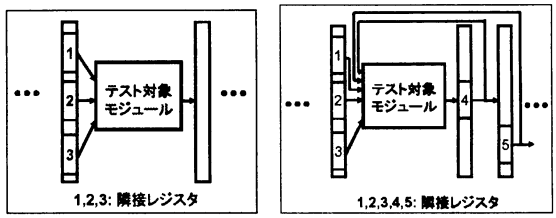


図 5: ODグラフの例

5.3. 命令テンプレート生成概要

命令テンプレートは，選択した各テスト対象モジュールに対して以下のように生成する．

1. 可制御性・可観測性は，テスト生成時に制約となる．そこで，ステージごとに同じ制約になる命令をグループ化する．また，可制御性・可観測性が包含関係にある命令もグループ化する．これにより，命令テンプレート生成時に入力空間の重複を防ぐことができる．
2. テスト対象としたモジュールにテストパタンを印加する命令を選択する．図6(a)のように，入力隣接レジスタが単一のステージに存在する場合，隣接レジスタへの可制御性 C_g のビット数の和について，命令を降順に並び替える．ランクの高い命令から順次選択し，各命令に対して命令テンプレートを生成する．また，図6(b)のように，入力隣接レジスタが複数ステージに存在する場合は，入力隣接レジスタが存在するステージそれぞれに対して1命令選択した命令列の組合せを全て抽出する．隣接レジスタへの可制御性 C_g のビット数の和について，抽出した命令列を降順に並び替える．そして，ランクの高い命令列から順次選択し，各命令列に対して命令テンプレートを生成する．



(a) (b)

図 6 :テスト対象モジュールと隣接レジスタ

3. 2において選択した命令または命令列に，非パイプラインレジスタ及び外部とのインターフェース命令列を付加する．インターフェース命令列とは，テスト対象モジュールに印加するテストパタンを非パイプラインレジスタにロードする命令列、及びテスト対象モジュールから出た出力応答を外部まで伝播させる命令列である．例として，図3において，1から4までの命令が，5のADD命令で必要なテストパタンを非パイプラインレジスタにロードしている．また，ADD命令で非パイプラインレジスタに書き戻した値を出力応答として，6と7の命令でメモリのアドレス計算を行い，8の命令で外部まで伝播させている．これらの命令列は，5.2.2や5.2.3で示したように，非パイプラインレジスタの可制御性・可観測性を求める際に生成しておくことができる．

6. 実験結果

本稿では，DLX/pⅡプロセッサのALUに対して命令テンプレート生成を行い，故障シミュレーションを行った．DLX/pⅡプロセッサは，5段のパイプラインステージを持つ．3つのアドレスモードがあり，52個の命令が実装されている．本稿では，ALU内部の単一縮退故障が対象である．テスト生成ツールはTestGen(synopsys)を用い，RTL-VHDLの論理合成にはDesignCompiler(synopsys)を利用した．ALUの入力隣接レジスタが存在する単一ステージに対し命令を選択し，命令テンプレートを生成し，その制約下でテスト生成を行った．

次にテストパタンからオペランドを決定し，プロセッサ全体で故障シミュレーションを行った．ここで，ALUのための命令テンプレートには割り込みに関連する命令や，JUMP命令が含まれているものがあるが，それらの命令テンプレートで故障を検出する方法は考慮中である．よって，それらの命令テンプレートを除き，故障シミュレーションを行った．全命令を用いた場合とそれらの命令を除いた場合のALU単体でのテスト生成結果を表1に，ALUに対するテストプログラムの故障シミュレーション結果を表2に示す．高い故障検出率，高い故障検出効率を達成していることがわかる．ALU単体では、全命令を，テストパタンを印加する命令として選択しテスト生成を行ったので、未検出であった130の故障は冗長であると判定できる。

7. むすび

本稿では，パイプラインプロセッサ自己テストのためのテストプログラム生成法を提案した．提案手法では，レジスタの可制御性，可観測性，非パイプラインレジスタに対する制御，観測のための命令列を利用して，各モジュールに対し命令列テンプレートを生成する．テンプレートから求めた制約の下でモジュール単体に対しテストパタンを求め，命令列に変換する．ALUに対する実験結果では，ALU単体に対して得られたテスト生成結果と同様，命令列による故障シミュレーションでも高い故障検出率，高い故障検出効率が得られることを示した．今後の課題は，本稿では除いた命令を用いてのテスト，パイプラインプロセッサ特有のフォワーディングユニットなどALU以外のモジュールのテストである．

謝辞 本研究に際し，多くの貴重な意見を頂いた本学の大竹哲史助手，米田助手，岩垣研究員はじめコンピュータ設計学講座の諸氏に深く感謝します．

文　献

[1] Wei-Cheng Lai, Angela Krstic, Kwang-Ting Cheng, "On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set," Proc. 18th VLSI Test Symp. , pp. 15－20. May. 2000.

[2] V.Singh and M.Inoue and K.Saluja and H.Fujiwara, "Instruction-Based Delay Fault Self-testing of Processor Cores",Proc.17thInternational Conf.on VLSI Design, pp.933-938, 2004.

[3] Jian Shen, Jacob A. Abraham, "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation," Proc. Int. Test Conf., pp.990-999, Oct.1998.

[4] N.Kranitis, G.Xenoulis, D.Gizopoulos, A.Paschalis, Y. Zorian, "Low-Cost Software-Based Self of RISC Processor Cores", IEEE Design, Automation & Test in Europe, pp. 1530-1591, 2003.

[5] Li Chen, Sujit Dey, "Software-Based Self-Testing Methodology for Processor Cores," IEEE Trans.Computer-Aided Design, Vol. 20, No. 3, pp369-380, Mar. 2001.

[6] K.Kambe, M.Inoue and H.Fujiwara, "Efficient template generation for instruction-based self-test of processor cores," IEEE 13th Asian Test Symposium (ATS'04) (to appear). 2004.

[7] D.A.Patterson and J.L.Hennessy, Computer Architecture -A Quantitative Approach (second edition), Morgan Kaufmann, 1996.

表 1: ALU単体でのテスト生成結果

命令	テスト パターン数	検出 故障数	未検出 故障数	全故障数	故障 検出率	故障 検出効率
全命令	244	8410	130	8540	98.48%	100%
割り込み，JUMP命令以外	234	7895	645	8540	92.45%	93.97%

表 2: ALUに対するテストプログラムの故障シミュレーション結果

	テスト プログラム数	検出 故障数	未検出 故障数	全故障数	故障 検出率	故障 検出効率
ALU	1832	7832	708	8540	91.71%	93.23%
プロセッサ全体	1832	23707	62327	86034	27.56%	27.71%