データパス分割に基づく高信頼プロセッサの提案とその評価

松坂，茂治
福岡大学大学院工学研究科電子情報工学専攻

井上，弘士
九州大学大学院システム情報科学研究院

出版情報：デザインガイア2004，電子情報通信学会研究報告VLD2004-49～60，pp.7-11，2004-12．電子情報通信学会VLD研究会
バージョン：
権利関係：

データパス分割に基づく高信頼プロセッサの提案とその予備評価

松坂 茂治† 井上 弘士‡

†福岡大学大学院工学研究科電子情報工学専攻 〒814-0180 福岡県福岡市城南区七隈 8-19-1
‡九州大学大学院システム情報科学研究院 〒816-858 福岡県春日市春日公園 6-1
E-mail: †matsusaka@v.tl.fukuoka-u.ac.jp, ‡inoue@c.csce.kyushu-u.ac.jp

あらまし コンピュータ・システムの高い信頼性を保つためには,障害の原因となる故障を検出する必要がある.故障を検出する一般的な手法として,時間的または空間的冗長性の利用が挙げられる.しかし,それらの冗長性の実現には追加ハードウェアや実行時間の増加といった問題が生じる.本稿では,ハードウェアの大幅な変更をすることなく空間的冗長性を実現するデータパス分割方式を提案する.また,演算時に必要となる最小のビット幅を考慮し,実行時間オーバヘッドを削減する方式を提案する.最善の場合を想定し実行時間を測定した結果,冗長度 2 の場合で平均 1.62 倍,4 の場合で平均 3.09 倍の実行時間増加となり,本手法は非常に有効であることが確認できた.

キーワード データパス分割,空間的冗長性,永久故障,一時故障

A Dependable Processor Architecture with Data-Path Partitioning

Shigeharu MATSUSAKA† Koji INOUE‡

†Dept. of Elec. and Computer Science, Fukuoka University 8-19-1 Nanakuma, Jonan-ku Fukuoka 814-0180 Japan

‡Dept. of Informatics, Kyushu University 6-1 Kasuga-Koen, Kasuga Fukuoka 816-8580 Japan
E-mail: †matsusaka@v.tl.fukuoka-u.ac.jp, ‡inoue@c.csce.kyushu-u.ac.jp

Abstract In order to maintain the high reliability of a computer system, it is necessary to detect the failure leading to an obstacle. In general, failure is detected by using time redundancy or spatial redundancy. However, in order to realize these redundancies, additional hardware and execution time increase. In this paper, the data-path partitioning which realizes spatial redundancy is proposed without making a large change of hardware. Moreover, the system which reduces an execution time overhead is proposed in consideration of the minimum bit width which is needed at the time of operation. In our evaluation, execution time became an average of 1.62 times (redundancy: two), and an average of 3.09 times (redundancy: four). Therefore, this technique is very effective.

Keyword data-path partitioning, spatial redundancy, permanent fault, temporary fault

1. はじめに

近年,コンピュータ・システムにおける信頼性の向上は最も重要な設計制約のひとつとなっている.特に,OLTP,人工衛星,医療,交通制御のような重要なアプリケーションでは,コンピュータ・システムの障害が重大な影響を及ぼすため,高い信頼性のシステムが必要不可欠である.そのため,コンピュータ・システムの核であるプロセッサの信頼性を向上させることが極めて重要となる.ここで障害とは,コンピュータ・システムがユーザの期待するサービスを提供しなくなることを意味し,障害の原因となる物理的な欠陥を故障と呼ぶ.故障は,長期間にわたって存在する永久故障(permanent fault)と,システムの稼動サイクルと比較して非常に短い時間にだけ存在する一時故障(temporary fault)に分けることが可能である.永久故障の発生要因としては,半導体接合の破壊や回路短絡,断線などシステムの内部的要因が考えられる.一方,一時故障は,温度の変化や振動,α粒子や宇宙線などシステムの外部的要因が主な原因となる.

高信頼化を実現するためには,それぞれの構成要素において信頼性の高い部品を使用し,かつ故障を起こしにくい単純な構成とすることが前提となる.しかしながら,故障が起きた場合に,物理的な故障をシステム全体の故障としないような構成方法が必要となる.故障を検出する一般的な方法として,時間的または空間的冗長性の利用が挙げられる.時間的冗長性は,1つのプロセッサ上で同じプログラムを複数回実行し,各実行によって生成された結果を比較することにより故障を検出する.しかしながら,この場合には,命令の複数回実行による実行時間の増加という問題が生じる.また,同一プロセッサを使用するため,永久故障を検出することができない.これは,同一箇所で故障

表 1：従来型冗長性との比較

	従来型		提案手法	
	時間的冗長性	空間的冗長性	単純多重化方式	圧縮多重化方式
一時故障	✓	✓	✓	✓
永久故障		✓	✓	✓
実行時間	↑	→	↑	↗
コスト	→	↑	→	→

が発生するため，複数回実行した結果をそれぞれ比較しても故障として認識できないためである．一方，空間的冗長性は，複数のプロセッサ上で同一プログラムをそれぞれ実行し，各実行によって生成された結果を比較することで故障を検出する．この場合には，一時故障だけでなく，永久故障も検出することが可能である．しかしながら，複数個のプロセッサを使用するため，ハードウェア・コストが増加するといった問題が生じる．

そこで本稿では，ハードウェア・コストの大幅な増加を招くことなく，空間的冗長性を実現するデータパス分割方式を提案する．また，演算時に必要となる最小のビット幅（以下，有効演算ビット幅と呼ぶ）を考慮し，実行時間オーバヘッドを削減する方式を示す．さらに，全ての演算に関する有効演算ビット幅は既知であると仮定し，データパス分割適用後の実行時間オーバヘッドを評価する．

以下，第 2 節では提案するデータパス分割の詳細を説明する．次に，第 3 節ではデータパス分割に基づく２つの実現方式を示す．また，本手法における故障の検出範囲を明確にする．第 4 節では本手法により引き起こされる性能オーバヘッドを評価する．第 5 節では関連研究について説明し，最後に第 6 節で簡単にまとめる．

2. データパス分割に基づく空間的冗長性の実現

本節では，データパス分割により空間的冗長性を実現する単純多重化方式を提案する．また，実行時間オーバヘッドの低減を目的とした圧縮多重化方式を示す．ここで，説明を簡単にするため，以下のような前提条件を設ける．プロセッサが有するデータパスの幅は 32 ビットであり，各命令実行は１クロック・サイクル（以下 CC）で終了すると仮定する．また，実現する冗長度は 2 もしくは 4 とする．すなわち，実行プログラムの各命令を冗長度 2 の場合は 2 回，冗長度 4 の場合は 4 回実行し，それぞれの結果を比較する事で故障を検出する．

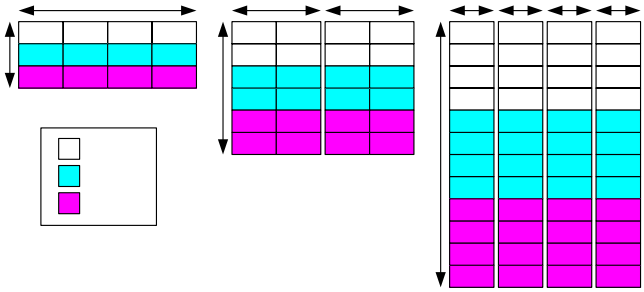

図 1：単純多重化方式

2.1. 単純多重化方式

提案方式では，例えば既存の 32 ビットデータパスを 16 ビットのデータパス 2 つに分割し，分割したデータパスそれぞれにおいて同一演算を実行する．この様子を図 1 に示す．ここでは，3 つの演算命令が連続して実行される場合を想定している．冗長度が 1 の場合，32 ビット幅である従来のデータパスをそのまま使用するため，3 命令の実行時間は 3CC となる．これに対し，空間的冗長度 2 を実現する場合，各命令の実行はそれぞれ 2 回の 16 ビット演算（上位と下位）で実現される．最初に，16 ビットに分割されたデータパスそれぞれにおいて下位 16 ビット演算を並列に実行する．次に，上位 16 ビット演算を同様に実行する．その後，分割されたデータパスでの実行結果を比較することにより故障を検出する．その結果，実行時間は 2 倍の 6CC となる．また空間的冗長度を 4 とする場合は，各命令の実行はそれぞれ 4 回の 8 ビット演算で実現されるため，実行時間は 4 倍の 12CC となる．このようなデータパス分割方式を単純多重化方式と呼ぶ．表 1 で示されるように，単純多重化方式は，従来の時間的冗長性を利用した場合と比較すると，性能（実行時間）とコストの面では同様の特徴を持つ．しかしながら，並列実行を行うため一時故障だけでなく永久故障も検出が可能となる．

2.2. 圧縮多重化方式

第 2.1 節で述べたように，単純多重化方式の欠点は実行時間の増加である．そこで，この欠点を解決する手段として圧縮多重化方式を提案する．通常，多くのプログラムにおいて，演算命令実行時に必要とされる有効演算ビット幅は，データパス幅より小さいことが知られている．例えば，SPECint95 では，有効演算ビット幅が 16 ビット以下であった命令は全体の約 50%であるとの報告もある[1]．空間的冗長度を 2 とする場合，有効演算ビット幅が 16 ビット以下であれば，上位 16 ビット演算を実行する必要はない．同様に冗長度 4

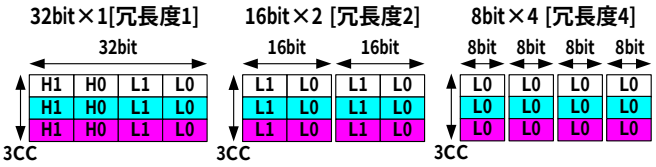

図 2：圧縮多重化方式

の場合も，有効演算ビット幅が 8 ビット以下である場合，下位 8 ビット演算のみを実行すればよい．すなわち，有効演算ビット幅が分割されたデータパス幅（16 ビットまたは 8 ビット）以下であれば， 1CC で命令実行を完了する．この条件が満たされる限り，実行時間の増加を伴うことなく分割されたデータパスで当該命令を実行できる．図 2 に本手法の概念を示す．図 2 では，3 つの演算命令すべての有効演算ビット幅が冗長度 2 の場合は 16 ビット以下，冗長度 4 の場合は 8 ビット以下での実行を示しており，3 つの演算命令がすべて 3CC で終了している．

3. データパス分割の実現

第 2 節で示したデータパス分割の実現方式として，静的データパス分割と動的データパス分割の概念を示す．また，故障の検出範囲について説明する．

3.1. 静的データパス分割

プログラムコンパイル時にデータパス分割を見かけ上実現できるように，仮想的にプロセッサのデータパスを分割する方式を静的データパス分割と呼ぶ．すなわち，専用コンパイラが冗長度（32bit×1, 16bit×2, 8bit×4）に応じたデータパス分割を実現するオブジェクト・コードを生成する．そして生成されたオブジェクト･コードを従来の 32 ビットプロセッサで実行することで空間的冗長性を実現する．図 3 に静的データパス分割の概念を示す．本概念は，プログラム実行以前に空間的冗長性を実現しているためハードウェアの大幅な変更を必要としない．つまり，既存の組込みプロセッサに本概念を適用できる可能性がある．しかしながら，プログラム実行以前に有効演算ビット幅を求める必要がある．この問題に関しては，文献[3][9]で提案された有効演算ビット幅解析技術を応用可能である．また，データパス分割を実現するコード生成技術に関しては，九州大学で開発された Valen-C コンパイラ[4]を用いる事で実現可能と考える．

3.2. 動的データパス分割

プログラム実行時に，専用ハードウェアによって演算を分割して実行する方式を動的データパス分割と呼ぶ．例えば，レジスタファイルから読み出したデータを分割するハードウェアを設けデータパスを分割する

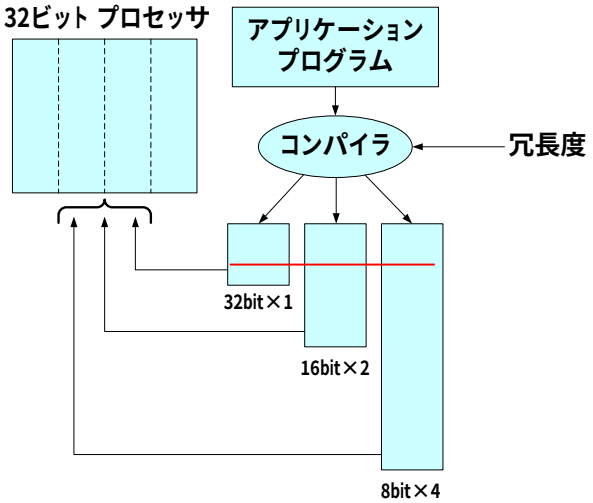

図 3：静的データパス分割

ことで空間的冗長性を実現する．圧縮多重化方式を実現する場合，有効演算ビット幅を動的に解析する回路が必要となるためハードウェア・コストの増加という欠点がある．しかしながら，静的データパス分割とは異なり，プログラム実行時に有効演算ビット幅を解析することが可能である．この問題に関しては，文献[1]で提案された動的解析の技術を応用可能である.また，ユーザの要求（性能・エネルギー・信頼性など）や実行環境に応じて冗長度を変更できる利点がある．

3.3. 故障の検出範囲

本節では，第 2.1 節および第 2.2 節で説明したデータパス分割方式（単純多重化方式および圧縮多重化方式）における故障の検出範囲について，単純な 5 段パイプライン・プロセッサ（図 4）を例に説明する．提案する方式は，演算命令を分割したデータパスで並列に実行し，結果を比較することで故障を検出する．つまり，レジスタファイルからデータを読み出し，演算を行った結果を比較するため故障の検出範囲は，図 4 の斜線部に示すデータパス部分となる．この図は，静的データパス分割で単純多重化方式および圧縮多重化方式を実現した場合の故障の検出範囲である．プログラム実行前に分割は終了しているため，レジスタファイルも故障の検出の対象となる．しかしながら，動的データパス分割を考慮した場合，プログラム実行中にレジスタファイルからデータを読み出した後，専用ハードウェアによってデータパスを分割するため，レジスタファイルは故障の検出範囲とはならない．また，本手法では，パイプライン制御のような制御回路部分の故障には対応していない．ここで，比較器についてだが，回路の三重化などを行い故障が発生しないことを前提にしなければならない．また一般的に，現在のマイクロプロセッサでは，命令キャッシュ，データキ

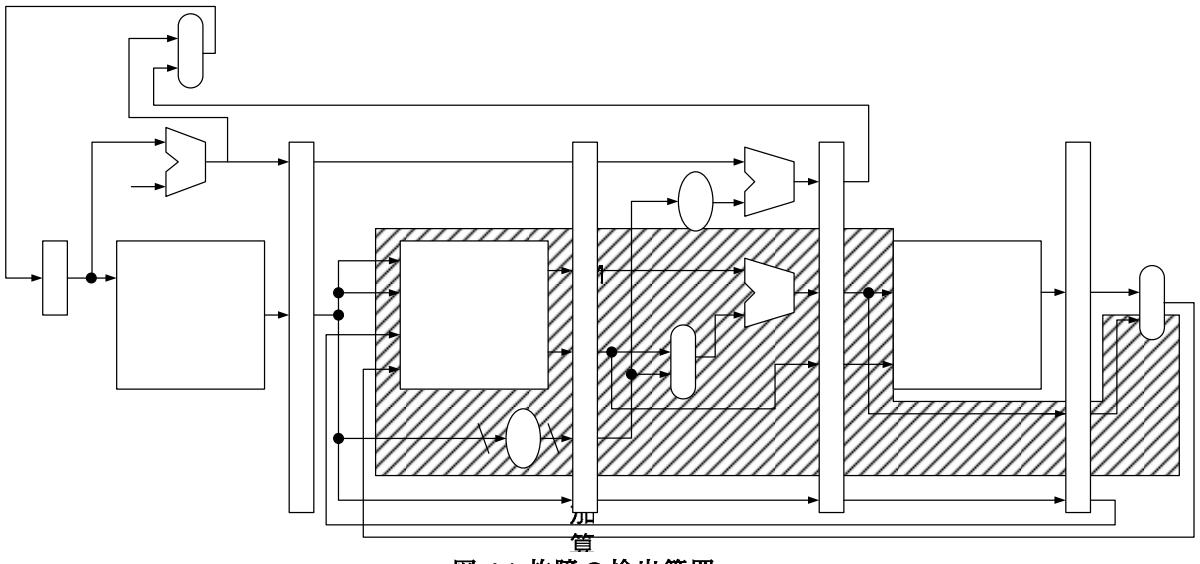

図 4：故障の検出範囲

ャッシュ，レジスタファイルは，パリティや誤り訂正符号（EEC：error-correcting code）等を用いて保護されている[8].

4. 予備評価

本節では，命令レベル・シミュレーションを行い，提案したデータパス分割手法における有効性の予備評価を行う．前提条件として，全ての有効演算ビット幅は既知であると仮定し，32 ビットのデータパス幅で冗長度 2 および冗長度 4 とした場合での圧縮多重化方式における実行時間増加を測定した．なお本稿では，故障を検出することを目的としているため，故障からの回復については考慮していない．

4.1. 実験環境

実行時間(*Execution Time*)は，次の式で与えられる．

$$Execution\ Time = IC * CPI * CCT$$

IC：実行命令数

CPI：命令当たりの平均所要クロック・サイクル数

CCT：クロック・サイクル時間

ここで，*CPI* を 1 と仮定し，*CCT* を固定する．したがって，実行時間は，実行命令数 *IC* にのみ依存する．また，圧縮多重化方式を用いて冗長度 2 ならびに 4 を実現した場合，実行命令数は次の式から求められる．

$$IC_{SR2} = IC_{org} + IC_{gt16b} * 1 \quad (1)$$

$$IC_{SR4} = IC_{org} + IC_{gt8b} * 3 \quad (2)$$

IC_{SR2}, IC_{SR4}：冗長度 2 または 4 の場合の実行命令数

IC_{org}：従来型 32 ビットデータパスでの実行命令数

IC_{gt16b}：「有効演算ビット幅≧16」の命令数

IC_{gt8b}：「有効演算ビット幅≧8」の命令数

本手法の有効性を評価するため SimpleScalar ツール・セット(ver.3.0d)[5]を改良して，各プログラム実行における IC_{org} と IC_{gt16b} と IC_{gt8b} を測定した．式(1)，(2)から分かるように，第 2 項（IC_{gt16b} *1 ならびに IC_{gt8b} *3）が実行時間オーバヘッドとなる．ベンチマーク・プログラムとしては SPEC2000[9]から 7 つの整数プログラムを用いた．また，入力データとしては SPEC から提供される small input を使用した．

4.2. 実験結果

第 2.1 節で述べたように，単純多重化方式を用いた場合の実行時間は，従来型プロセッサでの実行時間と比較して，冗長度 2 の場合で約 2 倍，冗長度 4 の場合で約 4 倍となる．これは単純多重化方式では，有効演算ビット幅を考慮しないため，IC_{gt16b} および IC_{gt8b} は IC_{org} と同一値となるためである．一方，圧縮多重化方式を用いたときの各ベンチマークにおける実行時間増加を図 5 に示す．結果は，冗長度が 1 である従来型プロセッサでの実行時間で正規化している．圧縮多重化方式を用いた場合の実行時間増加は，冗長度 2 の場合で平均 1.62 倍，冗長度 4 の場合で平均 3.09 倍となる．しかしながら，実行時間が冗長度の倍数となる単純多重化方式と比較すると，圧縮多重化方式での実行時間は減少する．

実行時間の増加をより詳細に解析するために，実行された命令の内訳（分岐命令・load/store 命令・ALU 命令・その他の命令）を調査し，それらに対する圧縮可能な命令の割合（圧縮可能命令率）を測定した．その結果を表 2 に示す．この結果は全ベンチマークの平均である．例えば，実行された全命令の 41.59%は ALU 命令であり，その内の 52.62%圧縮可能（つまり，有

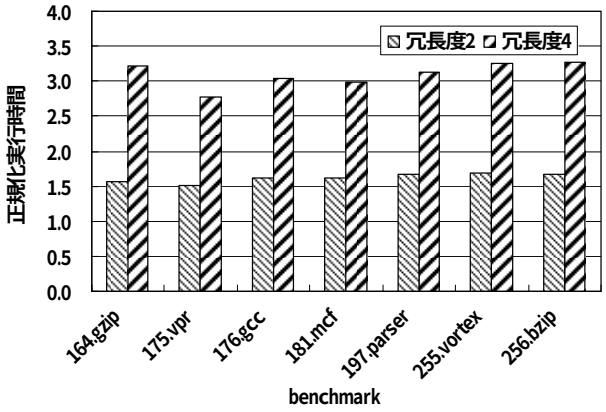

図 5：正規化実行時間

効演算ビット幅が 16 ビット以下）である．表 2 より，最も実行回数が多い ALU 命令に関しては，冗長度が 2 の場合，約半分程度の命令に関して圧縮可能である．しかしながら，全実行の約 37%を占める load/store 命令に関しては，ほとんど圧縮できていない．これは，load/store 命令を実行する際，ベース・アドレスが大きな値であったためである．よって，実行時間の増加を更に低減するには，コンパイラによるデータ再配置や，ベース･アドレス専用圧縮方式を確立する必要がある．また分岐命令は第 3.3 節で示したように条件判定を行う部分はデータパスを分割しているが，分岐先アドレスの計算は対象外であるため，圧縮可能命令率が高くなったと考えられる．

5. 関連研究

時間的冗長性や空間的冗長性を利用したフォールトトレランス技術として，AR-SMT[2]や DIVA[7]があげられる．AR-SMT は，空間並列マルチスレッド（SMT：simultaneous multithreading）の技術に基づき，対象とするスレッドを複製し，SMT で同時に実行した２つの結果を比較することで故障を検出するものである．DIVA は，メインプロセッサの実行結果をチェックするためのチェッカープロセッサを付け加えることで故障を検出する．文献[9]では，本稿で提案するデータパス分割方式に使用する有効演算ビット幅を考慮し，プロセッサやメモリのビット幅を小さくすることで低消費電力化を実現している．本稿では，信頼性向上の目的で有効演算ビット幅を使用した．

6. おわりに

本稿では，ハードウェア・コストの大幅な増加を招くことなく，空間的冗長性を実現するデータパス分割方式を提案した．また，最善ケースを想定し，実行時間増加率を評価することで，本手法の有効性について議論した．その結果，空間的冗長度が 2 の場合で平均

表 2：圧縮可能命令率

	全命令中の内訳	圧縮可能命令率	
		冗長度2	冗長度4
分岐命令	18.23%	82.43%	75.61%
load/store命令	37.32%	0.00%	0.00%
ALU命令	41.59%	52.62%	36.32%
その他	2.86%	0.00%	0.00%

1.62 倍，4 の場合で平均 3.09 倍の実行時間増加である事が分かった．

しかし本稿では，データパス分割方式の有効性を実行命令数だけを用いて評価している．今後，データパス分割をサポートしたプロセッサの設計，ならびに，第 4 節で説明したデータパス分割実行の実現方式を確立する予定である．

謝辞

多くの有用なアドバイスを頂いた福岡大学モシニャガ・ワシリー教授に感謝します．なお，本研究は一部，科学研究費補助金(課題番号：14GS0218, 14702064, 14102027）による．

文　献

[1] D. Brooks and M. Martonosi. “Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance.” *HPCA*, pp.13-22, 1999.

[2] Eric Rotenberg. “AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors,” *29th International Symposium on Fault-Tolerant Computing*, June 1999.

[3] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun, “Variable Size Analysis and Validation of Computation Quality,” *Proc. of Workshop on High-Level Design Validation and Test* , pp.95-100, Nov. 2000.

[4] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko, “Embedded System Design Using Soft-Core Processor and Valen-C,” *IIS Journal of Information Science and Engineering, vol. 14, no. 3*, pp. 587-603, Sep.1998.

[5] SimpleScalar Tool Sets, http://www.simplescalar.com/.

[6] SPEC (Standard Performance Evaluation Corporation), http://www.specbench.org/.

[7] T.M. Austin. “DIVA: a reliable substrate for deep submicron microarchitecture design,” *In Proc. MICRO-32*, pp. 196–207, Nov. 1999.

[8] Toshinori Sato, “Exploiting Instruction Redundancy for Transient Fault Tolerance,” *18th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp.547-554, November 2003.

[9] Y. Cao and H. Yasuura. “A System-level Energy Minimization Approach Using Datapath Width Optimization,” *Int. Symp. on Low Power Electronics and Design*, pp.231-236, Aug. 2001.