計算機アーキテクチャ　121－11
設　計　自　動　化　82－11
（１９９６．　１２．　１２）

関数型プログラムの実行に適した マルチスレッド型プロセッサ・アーキテクチャの提案

伊藤 英治　　相原 孝一　　丹 康雄　　日比野 靖

北陸先端科学技術大学院大学 情報科学研究科
〒923-12 石川県 能美郡 辰口町 旭台1-1

マルチスレッド型プロセッサ・アーキテクチャと関数型プログラムの特徴を組み合わせることにより、高性能化を実現するプロセッサ・アーキテクチャを提案する。本プロセッサでは、プロセッサの各パイプライン・ステージをすべて異なるスレッドからの命令で埋める機構、各ハードウェア資源の多重化によって、データハザード、制御ハザード、および構造ハザードの発生を回避する。さらに、キャッシュミスが生じた場合でもパイプラインをストールさせない機構を加えることによって、高いスループットを実現する。本稿では、プロセッサ・アーキテクチャの概要と簡単な性能見積りについて報告する。

A Multithreaded Processor Architecture for Functional Programs

Eiji ITOH　　Kouichi AIHARA　　Yasuo TAN　　Yasushi HIBINO

School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1, Asahidai, Nomi-gun, Ishikawa, 923-12 Japan

E-mail address: e-itou@jaist.ac.jp aihara@jaist.ac.jp
ytan@jaist.ac.jp hibino@jaist.ac.jp

Abstract

A high-performance multithreaded processor architecture for functional programs is proposed. This processor has multiple hardware resources for every thread, and a thread select unit which picks out the next thread to be executed. Therefore data hazards, branch hazards, and structure hazards are avoided. The processor makes no stalling the pipeline by a control unit dealing with cache miss. This paper describes the outline of the processor architecture and primitive performance evaluation.

1 はじめに

MOSデバイスは、その物理寸法を縮小することによって、動作速度が高速になる性質を持つ。近年のLSI製造技術の進歩によって、MOSデバイスの微細化が進み、素子の高速な動作速度が実現されている。しかし、物理寸法を縮小しても配線遅延は減少しない。この結果、動作クロックを上げることによって、プロセッサの高性能化を実現することが困難な状態になっている [3]。この点を解消する手段として、プロセッサのパイプライン・ステージを細分化することによって配線遅延を減少させ、動作クロックの高速化を図る方法が考えられる [6]。しかし、パイプライン・ステージを細分化すると、単一ストリームから命令を発行する限り、命令間の依存関係によって、パイプラインの持つ性能を引き出すことができないという問題が生じる。

この問題を解決する方法の1つとして、マルチスレッド型プロセッサがある [5] [10] [8]。マルチスレッド型プロセッサは、単一のプロセッサで複数の命令ストリーム（スレッド）を実行するプロセッサである。その特徴として、複数のスレッドを独立に制御するために、複数のプログラムカウンタと各種状態レジスタを持ち、複数のスレッドが機能ユニットを共有して命令を実行する形態をとる。これによって、命令発行の抑止されたスレッドに代わり、他のスレッドから命令を発行することによってパイプラインのスループットを向上させることができる。また、マルチスレッド型プロセッサでは、レジスタ類（プログラムカウンタやレジスタファイルなど）をスレッド数分必要とするが、集積度の向上によってチップ上の素子数が増大しているので設計上問題はない。

これらの点から、今後LSI製造技術の進歩を生かすためのプロセッサ・アーキテクチャは、マルチスレッド型プロセッサ・アーキテクチャであると考えられる。

本稿では、マルチスレッド型プロセッサ・アーキテクチャを、並列処理構造を持つ関数型プログラム [1] の実行に適したアーキテクチャにすることによって、並列処理のための高性能なプロセッサ・アーキテクチャを提案する。

以降の章で、マルチスレッド型プロセッサと関数型プログラムとの関係、そして今回提案するマルチスレッド型ウルトラパイプライン・プロセッサ・アーキテクチャを説明する。

2 マルチスレッド型プロセッサと関数型プログラムとの関係

関数型プログラムは、“互いに依存関係を持たない独立な関数同士を関数単位で並列実行することが可能である”という特徴を持つ。このため、プログラムの実行によって、並列実行可能な関数がプログラム中の複数箇所で起動され（起動された関数を関数活性体という）、並列に実行することができる関数活性体が増大する。

一方で、マルチスレッド型プロセッサは、プロセッサ自身が持つ性能（パイプラインの高いスループット）を引き出すために、複数のスレッドを必要とする。

“関数型プログラムは並列実行可能なスレッドを多数供給できる”という点を生かして、マルチスレッド型プロセッサの高性能化を図ることが可能である。

3 マルチスレッド型ウルトラパイプライン・プロセッサ・アーキテクチャ（MUP）

3.1 ハードウェア構成

マルチスレッド型プロセッサ・アーキテクチャに対して、関数型プログラムの特徴である“多数のスレッドを供給することができる”という点を生かすと以下に示すことが可能になる。

- パイプラインを長くすることが可能

  長いパイプラインに対して命令を供給することができるスレッドの数が少ないと、命令間の依存関係によってパイプライン中にバブルが生じ、パイプラインの持つ性能を引き出すことができない。これに対して、関数型プログラムは、十分な数のスレッドを長いパイプラインに対して供給することができる。これによってパイプラインの性能を引き出すことができる。このようにパイプラインを長くすることが可能であるため、パイプラインの各ステージを非常に細分化し、プロセッサの動作クロックの高速化によりパイプラインのスループットを向上させることができる。

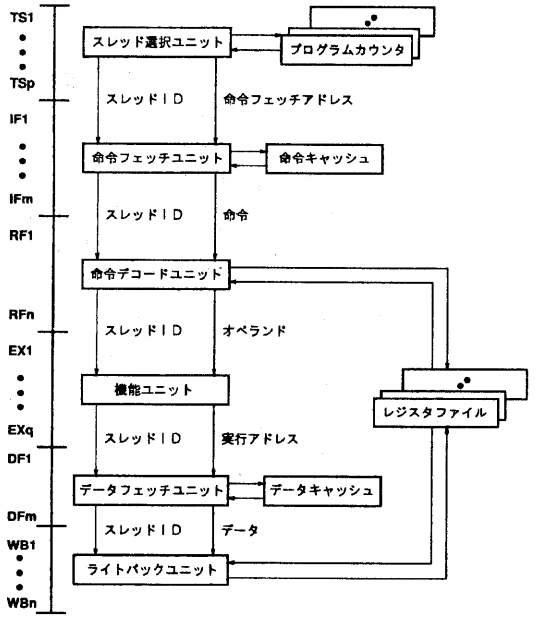

図 1: ハードウェア構成

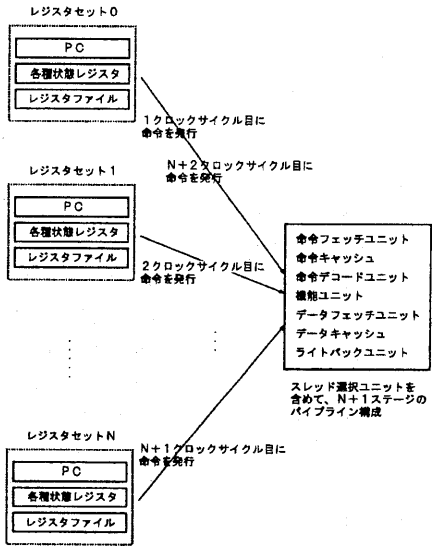

図 2: 実行スレッドの切り替え

- 各パイプライン・ステージをすべて異なるスレッドからの命令で埋めることが可能

  各パイプライン・ステージをすべて異なるスレッドからの命令で埋めることができると、プロセッサは1サイクル1命令の速度で命令を実行することが可能になる。また、パイプライン中のすべての命令が異なるスレッドから発行されたものであれば、互いに依存関係を持たないのでデータハザードや制御ハザードが発生することがない。このため、パイプラインの制御を非常に簡単化することができる。

上記の点を考慮したマルチスレッド型プロセッサ・アーキテクチャが、本稿で提案するマルチスレッド型ウルトラパイプライン・プロセッサ・アーキテクチャ（MUP）である。MUP のハードウェア構成を図1 に示す。

MUP は、プログラムカウンタ、各種状態レジスタ、およびレジスタファイルをスレッド毎に備え、命令およびデータフェッチユニット、命令デコードユニット、命令キャッシュ、データキャッシュ、ライトバックユニット、および機能ユニットを複数のスレッドによって共有する。

3.1.1 MUP の特徴

MUP は、次に示す特徴によって、動作クロックの高速化により高いスループットを実現し高性能化を図る。

- パイプライン・ステージ数と同数のレジスタセット[1]

  各パイプライン・ステージをすべて異なるスレッドからの命令で埋めるためには、プロセッサがパイプライン・ステージ数と同数のスレッドを同時に扱う必要がある。このため、MUP ではステージ数と同数のレジスタセットを用意している。さらに、用意しているレジスタ類は、1つのレジスタファイルを使って複数のスレッドで共有する形ではなく、各スレッド毎に独立した形をとる。これによってパイプラインの構造ハザードを回避している。

- スレッド ID 用パイプライン

  複数スレッドからの命令を1つのパイプラインを共有して実行する場合、どのスレッドから発行された命令なのかを識別する必要がある（例えば、演算結果をレジスタファイルへ書き戻す場合、どのスレッドのレジスタファイ

[1] 1つのスレッドを制御するのに必要な、プログラムカウンタ、各種状態レジスタ、レジスタファイルの組を表す表現

ルに書き込むかを識別しなければならない)。このため、命令およびオペランドが流れるパイプラインの他にスレッド ID が流れるパイプラインを持つ。

- 実行スレッドの切り替え

  パイプラインの各ステージをすべて異なるスレッドからの命令で埋めるために、命令発行を行なうスレッドは毎サイクル切り替わる。命令発行を行なうスレッドの選択方式は、図2に示すラウンドロビン方式で行なわれる。このような実行スレッドの切り替えによって、同一スレッドからの命令発行は、先行命令が完了してから後続命令を発行する逐次実行の形態になる。このため、データハザードや制御ハザードが回避できパイプラインの制御が簡単になる。

- 命令キャッシュとデータキャッシュの分離

  命令キャッシュとデータキャッシュを分離することにより、構造ハザードの発生を回避している [9]。

- キャッシュメモリとレジスタファイルのパイプライン化

  MUP では、動作クロックの高速化を目的として、パイプライン・ステージの細分化を行なう。このため、レジスタファイルやキャッシュメモリに対しても高いスループットが求められる。この要求を満たすために、レジスタファイルとキャッシュメモリのパイプライン化を行なっている。キャッシュメモリのパイプライン化については、3.2 節で詳しく述べる。

- 大容量キャッシュの搭載

  MUP では複数のスレッドが 1 つのキャッシュメモリを共有する。さらに、実行スレッドは毎サイクル切り替わる。そのためメモリアクセスの局所性が無くなる可能性がある。この問題に対処するために、MUP は扱うスレッド数に応じた大容量のキャッシュメモリを持つ。

- キャッシュミスの取り扱い

  MUP では高いスループットを保つために、あるスレッドの命令がキャッシュミスを起こした場合でもキャッシュミスを起こした命令の実行を無効にすることによって、パイプラインをストールさせることなく他のスレッドから発行された命令の実行を続ける。また、キャッシュミスが発生してキャッシュ更新待ち状態になったスレッドを、ソフトウェアまたはハードウェアによって他の実行可能なスレッドと入れ換えることは行なわない。これは、ソフトウェアでスレッドの入れ換えを行なう場合には、ソフトウェアによるコンテキストスイッチ時間が、キャッシュミスによって生じる主記憶からの読み出しに必要な時間に対して長いからである。ハードウェアによってコンテキストスイッチを行なう場合には、ハードウェア自身が複雑になる。
  このキャッシュミスの取り扱いに関しては、3.3 節で詳しく述べる。

3.1.2 パイプラインの各ステージの役割

命令パイプラインは、図1の左側に示すように、おおまかに 6 つのステージから構成されている。6 つの各ステージは、それぞれ数段に細分化されたパイプライン構造になっている。以下に各パイプライン・ステージの役割の詳細を述べる。

TS1～p (Thread Select) スレッド選択ユニットが、すべてのスレッドの中から、ラウンドロビン選択方式に従って、次に命令発行を行なうスレッドを選択する。さらに、この選択されたスレッドに対応するスレッド ID とプログラムカウンタを命令フェッチステージに渡す。

IF1～m (Instruction Fetch) 命令キャッシュを通じて、プログラムカウンタの指すアドレスから命令を読み出す。

RF1～n (Register Fetch) 命令をデコードし、EX ステージで使用するソースオペランドを決定する。レジスタファイルからオペランドを読み出す場合には、スレッド ID に対応するレジスタファイルから読み出す。

EX1～q (Execution) 与えられたソースオペランドを使用して指定された演算を行なう。ロードまたはストア命令の場合は実行アドレスを計算する。分岐命令の場合は分岐条件が真か偽かの計算を行なう。

DF1～m (Data Fetch) データキャッシュを通じて、EX ステージで得られた実行アドレスに対してロードまたはストアを行なう。

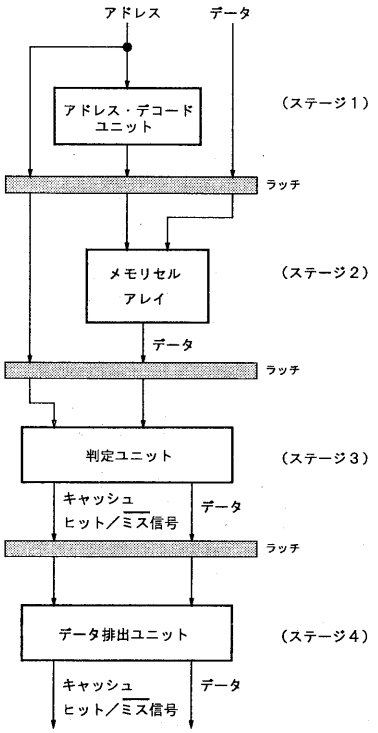

図 3: 4 ステージから構成されるパイプライン化キャッシュ

WB1〜n(Write Back) EX ステージで得られた演算結果や DF ステージでメモリからロードされたデータを、スレッド ID に対応するレジスタファイルへ書き戻す。

3.1.3 例外処理機能

例外の発生源には、プロセッサ内部（未定義命令実行、算術オーバーフローなど）とプロセッサ外部（割り込み）とがある。MUP では例外を発生源別に次のように処理する。

プロセッサ内部 例外の発生したスレッドの実行だけを中断し例外処理を行なう。

プロセッサ外部 実行中のスレッドの内、どれか 1 つのスレッドが命令の実行を中断し例外処理を行なう。どのスレッドが命令の実行を中断して例外処理を行なうかというのは、割り込みの発生するタイミングによる。また、割り込みに関する制御（割り込みマスク）はすべてのスレッド間で共有する。したがって、あるスレッドが 1 つの割り込み原因からの割り込みの受け付けをマスクすると、他のすべてのスレッドもその割り込み原因からの割り込みを受け付けない。

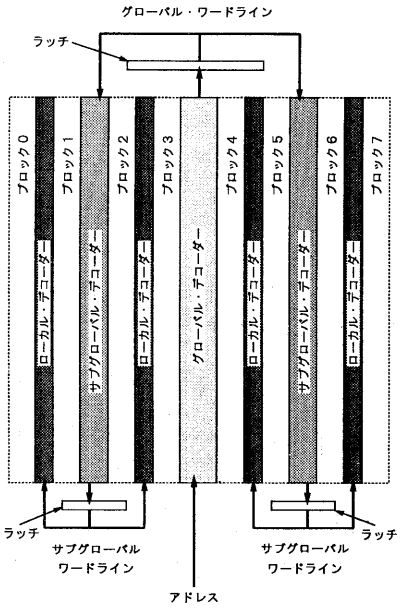

図 4: メモリ・セルの分割によるアドレスデコードのパイプライン化

3.2 キャッシュメモリのパイプライン化

パイプラインの高スループットを実現するためには、キャッシュメモリも高いスループットで動作する必要がある。この要求を満たすために、MUP ではキャッシュメモリのパイプライン化を行なっている。

キャッシュアクセスは、大まかには次の 4 つのフェーズによって行なわれる。

1. アドレスデコード
2. メモリセル・アレイの読み出し
3. 判定
4. データ排出

この 4 つのフェーズに分割することによって、図. 3に示す4 ステージから構成されるパイプライン化キャッシュを実現する。

また、1 つのアレイで構成される大容量のキャッシュメモリは、メモリセル・アレイをアクセスするためにかなりの時間を必要とする。そのため、パ

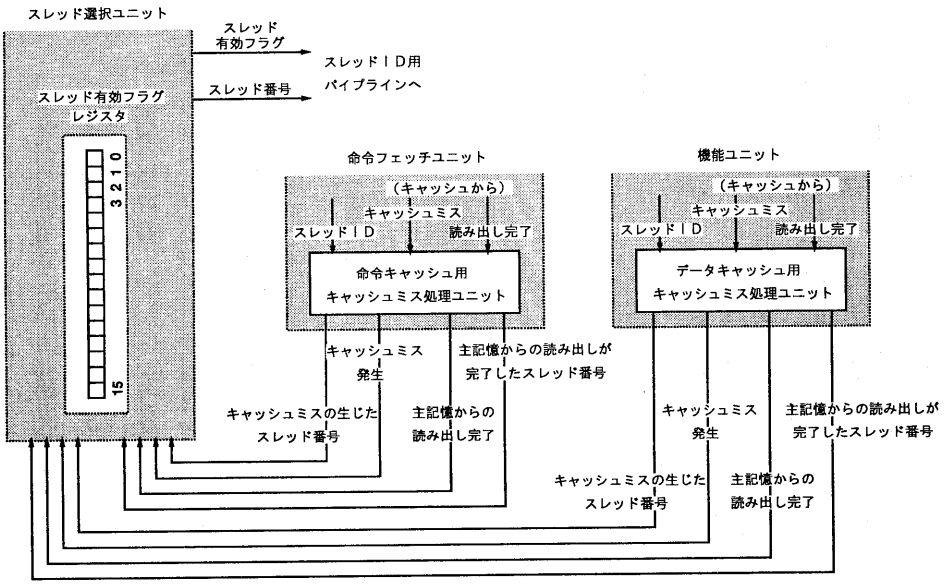

図 5: キャッシュミス取り扱い機構

イプライン化キャッシュメモリの 4 ステージの中で、最大遅延パスがあるのはメモリセル・アレイのアクセス部分（図 3のステージ 2 の部分）になる。パイプラインをさらに細分化したパイプラインにする必要がある場合には、この 1 つの大容量メモリセル・アレイを複数のサブアレイに分割 [2] [4] することによりパイプライン化を行なう。

例として、メモリセル・アレイを 8 分割し 3 段階にデコードするパイプライン化メモリアレイを図 4に示す。ここでは、メモリセルへアクセスするためのアドレスを、“グローバル・デコーダー”、“サブグローバル・デコーダー”、“ローカル・デコーダー” の 3 つのデコーダーによって 3 段階にデコードすることにより、分割されたメモリアレイのいずれかをアクセスすることになる。

3.3 キャッシュミスの取り扱い

MUP は、あるスレッドの命令を実行中にキャッシュミスが生じた場合でも、パイプラインをストールさせ主記憶からの読み出しを待つことを行なわず、キャッシュミスを起こしていない他のスレッドの命令の実行を続けることによって高いスループットを保つ。この動作を行なうための機構を図 5（スレッド数が 16 個の場合）に示す。この機構は、スレッド有効フラグ・レジスタと 2 つのキャッシュミス処理ユニット（命令キャッシュ用、データキャッシュ用）から構成される。

各スレッドは、“valid（キャッシュミスが生じていないため命令実行可能)” と “invalid（キャッシュミスによる主記憶からの読み出し待ちのため命令実行不可能)” の 2 つの状態に分類される。スレッド有効フラグ・レジスタの各ビットが、各スレッドの状態に対応し、すべてのスレッドの現在の状態を保持している。キャッシュミス処理ユニットは、スレッド番号を格納するキューを持つ。キャッシュミスが発生すると、ミスが生じたスレッドのスレッド番号がこのキューに格納され、主記憶からの読み出しが完了すると、このキューからスレッド番号が取り出される。またキャッシュミス処理ユニットは、スレッド有効フラグ・レジスタに対して、キャッシュミス情報（“キャッシュミス発生” と “キャッシュミスが発生したスレッド番号” の対から構成される）、または読み出し完了情報（“主記憶から読み出し完了” と “読み出しが完了したスレッド番号” の対から構成される）を伝える動作を行なう。

命令またはデータキャッシュへのアクセスでキャッシュミスが生じると、キャッシュミス処理ユニット

は、スレッド有効フラグ・レジスタに対して、キャッシュミス情報を伝える（これによって、キャッシュミスを引き起こした命令を発行したスレッドをinvalid状態とする）。同時に、パイプライン中に存在するキャッシュミスが生じたスレッドのスレッド有効フラグをinvalidに変更する。

invalid状態のスレッドがスレッド選択ユニットによって選択され、パイプラインに投入されると（同時に、スレッドに対応するスレッド有効フラグもスレッドID用パイプラインに投入される）、各パイプライン・ステージは、invalid状態を検知し、いかなる動作も行なわない。すなわち、パイプライン中には、invalid状態のスレッドが発行した命令がバブルとなって存在する形になる。

キャッシュミスによって生じた主記憶からの読み出しが完了すると、キャッシュミス処理ユニットは、スレッド有効フラグ・レジスタに対して、読み出し完了情報を伝える。これによって、invalid状態のスレッドがvalid状態に変更され、命令の実行を停止させられていたスレッドが、再び命令を実行することができる。

この機構によって、パイプラインをストールすることなく、命令を実行することが可能となる。

4 設計と性能見積り

4.1 設計

命令セット

本稿で設計するMUPは、命令セットとして表1に示す命令を持つ。表1に示す通り、性能見積りのために設計したMUPは、一般的な命令はほとんど含んでいるが、整数除算・乗算命令、浮動小数点に関する命令は持っていない。また、すべての命令は32ビット固定長である。

パイプライン構成

命令パイプライン構成は、図1において、各ステージを以下に示すように分割した16ステージから構成される。

TS1 命令発行を行なうスレッドを選択する。

TS2 TS1ステージで選択されたスレッドに対応するプログラムカウンタを読み出す。

IF1 命令キャッシュのアクセスを開始する。

IF2〜3 命令キャッシュへのアクセス中。

IF4 キャッシュ・アクセスが完了する。キャッシュミス処理ユニットが、キャッシュミスまたはヒットに応じて動作する。

表1: 命令セット

ロード	LB, LBU, LH, LHU, LW
ストア	SB, SH, SW
論理演算	AND, ANDI, OR, ORI XOR, XORI
算術演算	ADD, ADDI, ADDIU, ADDU SUB, SUBU
シフト	SLL, SLLV, SRA, SRAV SRL, SRLV
分岐	J, JAL, JALR, JR BEQ, BGEZ, BGEZAL, BGTZ BLEZ, BLTZ, BLTZAL, BNE
その他	SYSCALL, RFE, LDSTW MFPC, MFSR, MFCR, MFEPC MTPC, MTSR, MTEPC

RF1 命令のデコードを行なう。オペランドを得るためにレジスタファイルへのアクセスを開始する。

RF2 レジスタファイルからのオペランドのフェッチが完了する。

EX1 ソースオペランドを使用して指定された演算を行なう。加算以外の演算はこのステージで完了する。

EX2 指定する演算が加算命令の場合、このステージで演算が完了する。

DF1 メモリ命令ならば、データキャッシュへのアクセスを開始する。

DF2〜3 データキャッシュへのアクセス中。

DF4 キャッシュ・アクセスの完了。IF4ステージと同様に、キャッシュミス処理ユニットが動作する。

WB1 演算結果やメモリからロードされたデータをレジスタファイルへ書き戻す動作が開始される。

WB2 演算結果とロードされたデータをレジスタファイルへ書き戻す動作が完了する。

論理設計には動作記述論理設計支援ツールPARTHENON[7]を使用した。

4.2 論理合成の結果と性能

“ゲートのファンアウトを1に制限”、“ゲートのドライブ能力が不足する場合はバッファを挿入”という条件の下で、先に示したハードウェアの論理

表 2: 各ステージの論理段数

ステージ	段数	ステージ	段数
TS1	7	EX1	7
TS2	5	EX2	7
IF1	-	DF1	-
IF2	-	DF2	-
IF3	-	DF3	-
IF4	5	DF4	5
RF1	7	WB1	7
RF2	6	WB2	6

合成を行なった（ただしキャッシュメモリ・システムは論理合成に含んでいない）。各ステージの論理段数は表 2に示す通りとなった。また、プロセッサは以下に示すゲート数から構成されている。

- ゲート数：75,345（フリップフロップ 3,472 個を含む）
- レジスタファイル等のビット数：17,440

表 2より、最大遅延パスの論理段数は 7 段であることがわかる。この 7 段にステージのラッチ分（論理段数 3 段）を含め、さらに信号の伝達時間を 1 段当たり 100$psec$ と仮定すると、本プロセッサは 1GHz で動作することが可能である。

5 おわりに

本稿では、マルチスレッド型プロセッサ・アーキテクチャと関数型プログラムの特徴を組み合わせることによって、高い動作クロックと高スループットを実現する、マルチスレッド型ウルトラパイプライン・プロセッサ・アーキテクチャを提案した。

また、16 ステージから構成されるマルチスレッド型ウルトラパイプライン・プロセッサを設計し論理合成することにより、その性能を評価した。その結果、MUP は、最大遅延パスの論理段数はパイプラインラッチを含め “10 段”、総ゲート数は “75k”、レジスタファイル等のビット数は “17k” で構成されることがわかった。

今後は、より精密なデバイスパラメーターを用いることによる性能評価、パイプラインの設計の最適化を行なう。

謝辞

PARTHENON の使用を快諾して下さいました日本電信電話株式会社に感謝致します。また、PARTHENON に関しお教え頂いた NTT 研究所の中村行宏氏、小栗清氏に感謝致します。

参考文献

[1] 雨宮 真人, 田中 護, “コンピューターアーキテクチャ,” オーム社, 1988

[2] Toshihiko Hirose, et al., “A 20ns 4Mb CMOS SRAM with Hierarchical Word Decording Architecture”, *ISSC DIGEST OF TECHNICAL PAPERS*, pp.132-133, 1990.

[3] 井口秀之, “次世代集積技術による RISC アーキテクチャプロセッサの設計と評価”, 北陸先端科学技術大学院大学修士論文, 1995

[4] Koichiro Ishibashi, et al., “A 300MHz 4-Mb Wave-Pipeline CMOS SRAM using a Multi-Phase PLL”, *ISSC DIGEST OF TECHNICAL PAPERS*, pp.308-309, 1995.

[5] Daniel C. McCrackin, “Eliminating Interlocks in deeply Pipelined Processor by DelayEnforced Multistreaming”, *IEEE Trans. on Computer*, Vol.40, No.10, pp.1125-1132, Oct 1991.

[6] MIPS Technology Inc. “R4400 MICROPROCESSOR PRODUCT INFORMATION”, 1996.

[7] 小栗 清, 名古屋 彰, 野村 亮, 雪下 充輝, “はじめての PARTHENON,” CQ 出版社, 1994

[8] Won Woo Park, Donald S. Fusell and Roy M. Jenevein, “Performance Advantages of Multithreaded Processors”, sl Proc. of the Third Int’l Conf. on Parallel Processing, Vol.1, pp.97-101, 1991.

[9] David A. Patterson and John L. Hennecy, “COMPUTER ORGANIZATION & DESIGN THE HARDWARE/SOFTWARE INTERFACE”, Morgan Kaufmann Publishers Inc, 1994.

[10] R.Guru Prasadh and Chuan-lin Wu, “A Benchmark Evaluation of a Multi-threaded RISC Processor Architecture”, sl Proc. of the 20th Int’l Conf. on Parallel Processing, Vol.1 pp.84-91, 1991.