特別研究報告書

パイプラインステージ統合を行う
プロセッサの設計と消費電力評価

指導教員　　富田 眞治 教授

京都大学工学部情報学科

鈴木 一範

平成19年2月9日

パイプラインステージ統合を行う プロセッサの設計と消費電力評価

鈴木 一範

内容梗概

近年，モバイル・プロセッサの消費電力が増加する傾向にあり，いかにして消費電力を削減するかということが大きな課題となっている．この要求に応えるため，現在の商用プロセッサには動的電圧制御 (DVS: Dynamic Voltage Scaling) と呼ばれる方式が導入されている．DVS とはプロセッサの負荷が低い時に，クロック周波数と電源電圧を低下させることで消費電力を削減するという手法である．だが，今後の半導体の製造プロセス技術の進歩によって有効性が低下する．原因はサブスレッショナルド・リーク電流の増大と SRAM などにおける閾値電圧のばらつきの拡大により，プロセッサ自体の電源電圧が下げづらくなるからである．

DVS に対し，半導体の製造プロセス技術に依存しない方式として，パイプライン・ステージ統合 (PSU: Pipeline Stage Unification) と呼ばれる方式が提案されている．PSU はプロセッサの負荷が低い時に，クロック周波数を低下させるという点では DVS と同じである．だが PSU は周波数が低下することにより発生したクロック・サイクル時間の余裕を利用して，1 サイクルの間に信号が複数のパイプライン・ステージを通過できるようにする点が異なる．その際，パイプライン・ステージ間にあるパイプライン・レジスタをバイパスする．PSU はバイパスされるパイプライン・レジスタへのクロック信号の供給を停止することでクロック・ドライバの総負荷を減少させる．またパイプライン・ステージの統合によってプロセッサのパイプライン段数を短くすることで，プログラムの実行に必要なサイクル数を削減する．この 2 つの結果として消費電力が削減される．

今までの PSU に関する研究はアーキテクチャ・レベルのシミュレータ上で行われてきた．このシミュレータ上での PSU による消費電力の削減の評価においては，商用プロセッサが公表しているデータを用いて評価を行っている．しかし，公表されているデータは大まかなものであり，個々のステージ間のパイプライン・レジスタのばらつき，PSU を適用できないパイプライン・レジスタ，個々のパイプライン・レジスタのアクティビティ・ファクタ等を考慮したシミュレー

ションはできていない．そこで，本研究では Verilog HDL を用いてプロセッサを設計/実装し，パイプライン・レジスタ等の消費電力の詳細な評価を行う．

作成するプロセッサは，MIPS 命令セットを実行する 4 命令同時発行の out-of-order スーパスカラ・プロセッサである．開発には Altera 社の QuartusII Ver 6.0 を使用した．MIPS 命令セットを選択した理由は主に 2 つある．第 1 に，MIPS 命令セットの場合，命令の種類が 132 個と少なく，かつ命令の構造が単純なので，デコーダなどの機能モジュールが比較的容易に作成できるということである．第 2 に，シミュレータである SimpleScalar Tool Set が MIPS ベースの命令を使用しているので今までの評価結果との比較が可能なように，それに合わせたということである．

プロセッサのうち設計したのはフロントエンド部分，実行コアの一部 (命令ウィンドウ，ALU の一部の機能)，およびバックエンド部分の一部 (Register File への読み書き) であり，バックエンド部分におけるコミットや例外処理の部分は実装していない．これはフロントエンド部分とバックエンド部分の処理に割り当てられているステージ数を比べた場合，フロントエンド部分のほうがバックエンド部分に比べて圧倒的に多いので，PSU による消費電力の削減のほとんどはフロントエンド部分で達成されているためである．

評価には QuartusII に付属する PowerPlay Power Analyzer Tool を使用し，パイプライン・レジスタを備えたプロセッサに対して簡単な命令列を実行させた上で消費電力を測定した．

プロセッサの総消費電力におけるパイプライン・レジスタの消費電力の割合は 7.30%，またクロック・ドライバの消費電力の割合は 15.86% を示した．このプロセッサに PSU を適応してパイプライン・ステージを統合すると，パイプライン・レジスタの消費電力を 56.16%削減できるという結果を得た．なお，クロック・ドライバ側の消費電力の削減については，シミュレータの都合によって評価を行えなかった．

この評価結果により，アーキテクチャ・レベルのシミュレーションの評価結果の信頼性をより高めることができると考えられる．

Processor Design for Pipeline Stage Unification and Power Consumption Evaluation

Kazunori SUZUKI

Abstract

Power consumption of recent mobile processors have the inclination to increase and we are required how to exhibit low-power consumption. To satisfy these requirements, a method called dynamic voltage scaling or DVS is currently employed in current commercial processors. DVS reduces power consumption by decreasing the supply voltage when a processor runs at a low clock frequency under low workload. Although DVS is an effective method for reducing power consumption, its effectiveness will be limited in future process technology because of subthreshold leakage current incrementation and threshold voltage deviation incrementation caused by process deviation incrementation.

As an alternative, we proposed a method called pipeline stage unification or PSU. PSU is similar to DVS at the point which reduces power consumption by decreasing the supply voltage when a processor runs under low workload. But PSU is different from DVS at which permits signals to pass several pipeline stages in each cycle by utilizing increased clock cycle time. At that time, PSU bypasses pipeline registers between unified pipeline stages. Power consumption reduction with PSU is achieved by two reasons. Firstly, PSU saves power consumption by reducing the total load capacitance of clock driver. This is accomplished by stopping the clock signal to bypassed pipeline registers. Moreover, PSU reduces cycle count for program execution by reducing pipeline stages. It reduces the time that the processor consuming power.

Previous PSU researches have only done with architecture level simulation. In evaluation of power consumption reduction with PSU, we used the values which are published by commercial processors. However, the published data are rough to use in our research because it doesn't mentioned about dispersion of pipeline registers among stages, pipeline registers which can't adapt PSU, and the effect of each pipeline register's activity factor. In this paper, I designed and implemented processor with Verilog HDL and evaluated detailed power consumption of pipeline register and so on.

I designed 4-way out-of-order superscalar processor which executes MIPS instruction set architecture. I used QuartusII Ver 6.0 (Altera Co.) for development. The reason why I chose MIPS instruction set is as follows. Firstly, in MIPS instruction set, the number of instructions is no more than 132 and structure of instructions is simple, so that I could easily develop function module like decoder. Secondly, the architecture level simulator SimpleScalar Tool Set which is used to evaluate PSU in previous research executes MIPS based instruction set. So, I chose MIPS instruction set to be able to compare with previous result.

There are many function modules in the processor, I designed front-end, the parts of execution core, Instruction Window, some of ALU, and the parts of back-end, Read/Write to register files. However, I didn't design commit function module and exception module. This is because the stage number which assigned to front-end is much larger than that of back-end, so that almost all of power consumption reduction by PSU is accomplished in front-end.

In evaluation, I used PowerPlay Power Analyzer Tool which is pertained to QuartusII and measured power consumption by executing simple instructions on the designed processor.

The evaluation results show that the ratio of the power consumption of pipeline registers to the power consumption of whole processor is 7.30%, and the ratio of the power consumption of clock driver to the power consumption of whole processor is 15.86%. Also our results shows that the power consumption of pipeline registers become 56.16% smaller if we adapt PSU in this processor. Yet I couldn't evaluate power consumption reduction of clock driver under PSU because PowerPlay Power Analyzer Tool couldn't evaluate it.

From this result, we can improve the reliability of simulation result which is evaluated in architecture level simulation.

パイプラインステージ統合を行う
プロセッサの設計と消費電力評価

目次

第1章　はじめに

近年のモバイル・プロセッサに求められる技術の1つに消費電力の削減がある．例えばノート型コンピュータの場合，消費電力を削減することで駆動時間が延長できる．あるいは従来より小さい電池で従来と同じ駆動時間を確保でき，機器の小型化やコスト削減となる．また消費電力の削減により，放熱用のファンを小さくしたり数を減らすことができ，コンピュータの駆動音が静かになるという利点がある．

この要求に応えるために，現在の商用プロセッサには動的電圧制御 (DVS: Dynamic Voltage Scaling) と呼ばれる方式が導入されている．DVS とはプロセッサの負荷に応じて，動的にクロック周波数と電源電圧を変更する消費電力の削減システムである．バッテリ持続時間の要求が強いか，与えられた負荷が低い場合はクロック周波数を低下する．さらに，延びたクロック・サイクル時間に信号の遅延を合わせ，電源電圧を低下させる．これにより，プログラムを実行するために必要な消費電力が削減できる．

DVS は消費電力を削減するには有効な手段であるが，現在は半導体の製造プロセス技術の進歩によりプロセッサの電源電圧が下げづらい状況にある．その理由は主に2つある．1つはサブスレッショナルド・リーク電流の増大である．サブスレッショナルド・リーク電流とはゲート電位が0ボルトであってもソース・ドレイン間を流れつづける電流のことである．このリーク電流は回路が動作していなくても流れつづけるので，結果として消費電力の増大になる．もう1つは SRAM などにおける閾値電圧のばらつきである．通常，1つの SRAM には6つないしそれ以上のトランジスタが使用されている．それらのトランジスタの閾値電圧に大きなばらつきがある場合，SRAM に対するデータの読み書きが正常にできなくなる可能性がある．また，近年は製造プロセスの微細化によるトランジスタ自体のばらつきの増大や，トランジスタを動作させるのに必要な電源電圧の低下による信号線の電位の減少により，閾値電圧のばらつきの影響がさらに拡大する傾向にある．ばらつきが拡大すると，閾値電圧を低く設定することができなくなり，結果としてプロセッサ自体の電源電圧も下げづらくなる．以上，上記2つの理由から半導体の製造プロセス技術が進むに従って閾値電圧が下げづらくなり，同時に電源電圧は閾値電圧の2倍以上を確保する必要があるので，電源電圧は下げづらい状況にある．

DVS に対し，半導体の製造プロセス技術に依存しない手法として**パイプラインステージ統合** (PSU: Pipeline Stage Unification) と呼ばれる手法が提案されている．PSU はプロセッサの負荷が低い時にクロック周波数を低下させるという点では DVS と同じである．だが，PSU はクロック周波数の低下によって延長したクロック・サイクル時間を使って複数のパイプライン・ステージを統合するという点が異なる．この際，統合されたステージ間にあるパイプライン・レジスタはバイパスされる．PSU によって消費電力が削減される理由は次の 2 点である．まず，第 1 にバイパスされるパイプライン・レジスタへのクロック信号の供給を停止することにより，クロック・ドライバの総負荷を減少させることができる．これにより消費電力が削減される．第 2 に，パイプライン・ステージの統合によりプロセッサ全体のパイプラインが短くなる．これにより，プログラムの実行完了に必要なサイクル数が削減され，結果として消費電力が削減される．

今までも PSU の研究は行われてきたが，その研究は主に MIPS 命令セット [1] をベースとした SimpleScalar PISA を実行するシミュレータの SimleScaler Tool Set 上でのみで行われてきた．このシミュレータ上での PSU による消費電力の削減の評価においては，商用プロセッサが公表している値を用いて評価を行っている．しかし，公表されているデータは大まかなものであり，個々のステージ間のパイプライン・レジスタのばらつき，PSU を適用できないパイプライン・レジスタ，個々のパイプライン・レジスタのアクティビティ・ファクタ等を考慮したシミュレーションはできていない．また，パイプライン・レジスタに関する動作もシミュレータ上と実際のプロセッサでは異なる点が多い．

そこで本研究では Verilog HDL を用いてパイプライン化されたプロセッサを設計/実装し，PSU を適用した時の消費電力の削減量等を調査する．作成するプロセッサは MIPS 命令セットを実行する 4 命令同時発行の out-of-order のスーパスカラ・プロセッサである．開発には Altera 社の QuartusII Ver 6.0 を使用した．MIPS 命令セットを選択した理由は主に 2 つある．第 1 に，MIPS 命令セットの場合，命令の種類が 132 個と少なく，かつ命令の構造が単純なので，デコーダなどが比較的容易に作成できるということである．第 2 に，SimpleScalar Tool Set が MIPS ベースの命令を使用しているので今までの評価結果との比較が可能なように，それに合わせたということである．

残りの論文の構成は以下のとおりである．2 章では PSU について述べる．こ

の章では過去の PSU の研究，PSU を実装する際の回路の詳細，そして PSU による消費エネルギーの削減について解析的に説明する．3 章では本研究で設計したプロセッサについて説明する．この章では設計したプロセッサの各モジュールについて説明する．4 章で消費電力の測定結果を示し，考察を行う．最後に，5 章でこの論文についてのまとめを行う．

第 2 章　パイプラインステージ統合 (PSU)

パイプラインステージ統合 (PSU: Pipeline Stage Unification) とは嶋田らによって提案された消費電力の削減手法 [2] であり，現在の商用プロセッサの多くに搭載されている削減手法である動的電圧制御 (DVS: Dynamic Voltage Scaling) と比較されてきた．

PSU は負荷が低い時にクロック周波数を低下させる．周波数の低下により余裕のできたクロック・サイクル時間を使って，複数のパイプライン・ステージを統合する．アーキテクチャ・レベルのシミュレータ上においては，PSU は DVS よりも多く消費電力を削減することができ，さらに削減量の差は，将来の半導体製造技術においてより大きくなることが示されている [3]．

本章では PSU について説明する．まず 2.1 節では PSU の実装方法について説明する．2.2 節では PSU によって消費エネルギーが削減される理由を解析的に説明する．2.3 節では現在の PSU の研究における問題点を説明する．

2.1　PSU の実装

図 2.1 に PSU に関連する信号線とパイプライン・レジスタとの結線関係を示す．説明を簡単にするために，2 ステージの統合を例としている．図 2.1 に示すように，PSU のための信号線として**統合信号**と呼ばれる信号線を追加する．また，パイプライン・レジスタに供給するクロック信号は常時クロック信号と統合信号の論理和とする．

図 2.1(a) はステージが統合されていない状態を，図 2.1(b) は統合された状態を示す．図中の黒い部分は動作部分を示し，灰色の部分は動作していない部分を示す．図 2.1(a) は通常のパイプラインとして動作し，統合信号は 1 である．隣接する組み合わせ論理回路 A と B は，それらの回路の間にあるパイプライン・レジスタにクロック信号が供給されているため，異なったステージとして動作

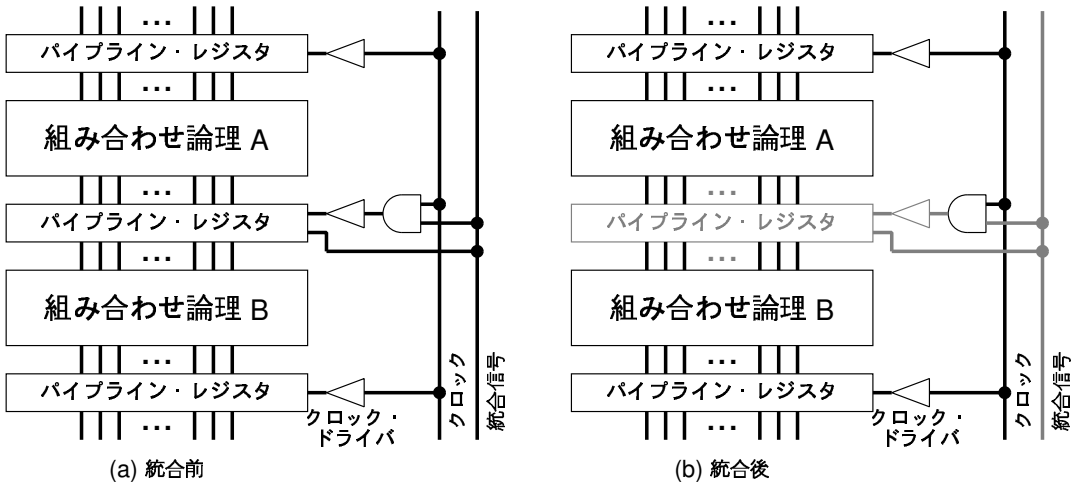

図 2.1: クロック分配線の変更と統合信号の追加

する．一方，図 2.1(b) では，統合信号を 0 にすることによって組み合わせ回路 A と B の間のパイプライン・レジスタへのクロック信号の供給が削減され，信号はバイパスされる．従って，パイプライン・レジスタは動作せず，2 つの組み合わせ回路は 1 つのステージとして動作する．

以上では 2 ステージ統合の場合についてのみ述べたが，統合信号を複数用意し，パイプライン・レジスタへ供給するクロック信号を適切に制御することにより，さらに多くのステージの統合が行えるように拡張可能である．

ここで，パイプライン・レジスタには，フロントエンド，実行コア，バックエンド間にある命令ウィンドウなどデカップリング用の記憶素子を含まないことを注意しておく．これらは，パイプライン・レジスタと同じくステージをつなぐ記憶素子であるが，複数の命令の状態を記憶する機能が必要なので，バイパスさせることはできない．

パイプライン・レジスタをバイパスさせるには，2 つの方法が考えられる．第 1 の方法は，統合時にクロック信号とは無関係に信号が通過するようパイプライン・レジスタの論理を構成することである．透過型のラッチでパイプライン・レジスタを実現する場合，この論理の変更は容易である．第 2 の方法は，パイプライン・レジスタの後方にマルチプレクサを配置し，パイプライン・レジスタの出力と前方のステージの出力を統合信号により選択する方法である．この方法は，パイプライン・レジスタをどのような回路で構成しても適用可能である．

インターロック回路にもわずかな変更が必要である．図 2.2 に変更したイン

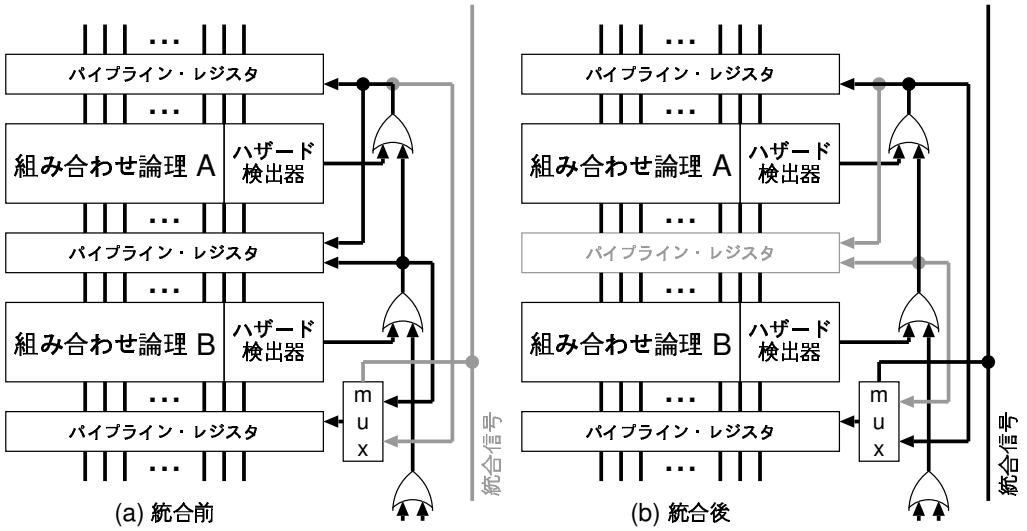

図 2.2: インターロック回路

ターロック回路を示す．図 2.2 (a) に統合していない状態での信号の流れを，図 2.2 (b) に統合した状態での信号の流れを示す．

一般に，あるステージのストール信号がアサートされるのは，そのステージでハザードを検出するか，または，後方のステージのストール信号が真のときである．図 2.2 に示した例では，$S(A)$ と $H(A)$ を，それぞれ，ステージ A のストール信号，ハザード信号とすると，$S(A)$ は以下の式により与えられる：

$$S(A) = H(A) + S(B) \tag{2.1}$$

ステージ A と B を統合する場合，統合後のステージのストール信号 $S(A+B)$ は，A または B がストールするときアサートされる必要があるので，以下のようになる：

$$S(A + B) = S(A) + S(B) \tag{2.2}$$

式 (2.1) を式 (2.2) に代入すると，

$$S(A + B) = H(A) + S(B) \tag{2.3}$$

となり，結局，$S(A)$ と等しいことが分かる．

ところで，ストール信号によりステージを実際にストールさせるためには，そのステージの前方にあるパイプライン・レジスタの更新を抑止し，後方のパイプ

ライン・レジスタへ渡す信号を NOP に変更する必要がある．組み合わせ回路 A の前方のパイプライン・レジスタへ渡す信号は，統合前後で，それぞれ，$S(A)$, $S(A+B)$ なので，式 (2.1)，(2.3) より変更の必要がないことが分かる．一方，組み合わせ回路 B の後方のパイプライン・レジスタへ渡す信号については，統合するかどうかに応じて $S(B)$ と $S(A+B)$ を切り替える必要がある．従ってマルチプレクサが必要である．

以上では 2 ステージ統合の場合についてのみ述べたが，統合信号，および，関連する回路を複数用意して適切に動作させることにより，さらに多くのステージの統合を行えることが可能になる．その場合でも，統合するステージ数に比例する程度の複雑さで拡張可能となる．

2.2 PSU による消費エネルギー削減量の解析

本章では，PSU による消費エネルギーの削減について解析的に説明する．一般に，プログラムの実行の際に消費するエネルギー E は以下の式で表される：

$$E = P \times T_{ex} \tag{2.4}$$

ここで，P は消費電力，T_{ex} は実行時間である．P と T_{ex} は以下の式で与えられる：

$$P = a \times f \times C \times {V_{DD}}^2 \tag{2.5}$$

$$T_{ex} = \frac{N}{IPC \times f} \tag{2.6}$$

ここで，a はアクティビティ・ファクタ，つまり，各ノードの平均スイッチング確率，f はクロック周波数，C はスイッチするノードの全容量，${V_{DD}}^2$ は電源電圧，N は実行命令数，IPC は 1 サイクルあたりの実行命令数の平均を示している．

2.1 節で述べたように，PSU は一時クロック信号のドライバを停止することによって消費電力を削減する．U ステージ統合 (以下，これと**統合度** U と呼ぶ) でプロセッサが動作しているとき，理想的にはクロック・ドライバが消費する電力は $1/U$ となる．また，通常のプロセッサと同じく，総消費電力はクロック周波数の低下率に比例して削減される．従って，クロック周波数 f_{low} で動作す

る，統合度 U の PSU プロセッサの消費電力は以下の式で表される：

$$P_{PSU}(f_{low}, U) = \left(P_{total} - P_{clock} + \frac{P_{clock}}{U}\right) \times \frac{f_{low}}{f_{max}} \tag{2.7}$$

ここで，P_{total} と P_{clock} はそれぞれ，通常モードにおけるプロセッサの総消費電力とクロック・ドライバの消費電力である．

式 (2.4) を用い，通常モードで正規化した消費エネルギーは以下の式で表される：

$$E_{PSU}, n(f_{low}, U) = \frac{P_{PSU}(f_{low}, U) \times T_{ex}(f_{low}, U)}{P_{total} \times T_{ex}(f_{max}, 1)} \tag{2.8}$$

ここで，$T_{ex}(f, U)$ はクロック周波数 f と統合度 U における実行時間である．$T_{ex}(f_{max}, 1)$ は通常モードにおける実行時間を示す．式 (2.6) と (2.7) を式 (2.8) に代入することにより，以下の式を得ることができる：

$$E_{PSU}, n(f_{low}, U) = \frac{IPC_{max}}{IPC_{low}} \times \left\{1 - k \times \left(1 - \frac{1}{U}\right)\right\} \tag{2.9}$$

ただし，

$$k = \frac{P_{clock}}{P_{total}} \tag{2.10}$$

である．

式 (2.9) から分かるように，消費エネルギーは IPC の向上に反比例する (パイプラインが短縮されるため，$IPC_{max} < IPC_{low}$ であることに注意)．また，消費エネルギーの削減は，クロック・ドライバにより消費される電力が，プロセッサの総消費電力のどれだけの割合を占めているかということに依存する．この割合は近年の高速なプロセッサでは非常に大きいと言われているが，実際の値ははっきりしていない．

2.3 現在の PSU の研究における問題点

2.2 節にも記した通り，PSU は解析的には消費電力を削減できるという結果が示されている．しかし，現在の PSU の研究は主にアーキテクチャ・レベルのシミュレータである SimpleScalar Tool Set 上で行われており，実際のプロセッサ上では消費電力がどれほど削減されるのか，あるいはパイプライン・ステー

ジ統合を行う回路をプロセッサに実装することで，プロセッサ全体の回路がどれほど複雑になるのかという事に関してははっきりしていない．また，現在の消費電力評価では，パイプライン・レジスタの消費電力がプロセッサ全体の消費電力に占める割合は，公表されている商用プロセッサのデータを用いている．このデータは，それほど詳細ではなく，個々のステージ間のパイプライン・レジスタの消費電力の差異は分からない．そのため，次にあげる項目の評価があいまいになっていたり，未評価であったりする．

- シミュレータではパイプライン・ステージ統合を行った場合，パイプライン・ステージ間にあるパイプライン・レジスタに対しては必ずクロック・ドライバの供給を停止できる，という前提でシミュレーションを行っている．しかし，実際のプロセッサ上においてはパイプライン・ステージ統合を行った場合であっても，クロック・ドライバの供給を停止することができないパイプライン・レジスタが存在する．以前の評価では，このようなパイプライン・レジスタは少ないと仮定し，無視して評価を行っていた．
- プロセッサの各ステージ間で受け渡されるデータは，ステージ間によって変化する．例えば，デコード後のステージ間では，デコード済みのデータが追加されるため，デコード前のステージ間よりも多くなると考えられる．よって，プロセッサの各ステージ間に存在するパイプライン・レジスタの数やそのビット幅はステージ間ごとに異なっている．以前の評価では，ステージ間のパイプライン・レジスタ数やビット幅の差異はないともの仮定して評価を行っていた．

上記に示したような項目を評価するためには，実際にプロセッサを設計/実装する必要がある．

第3章　PSUによる消費電力削減の評価のためのプロセッサの設計

本章では，本研究で設計した，実際にPSUを搭載する元となるプロセッサについて示す．まず3.1節ではプロセッサ概要について，3.2節で機能モジュールについて，3.3節ではRAMを構成要素とするモジュールについて説明する．

3.1 プロセッサ概要

本節では設計したプロセッサ概要について説明する．図 3.1 に作成するプロセッサの概要を示す．図 3.1 中における各モジュール内の括弧は．そのモジュールが使用されるステージを示す．

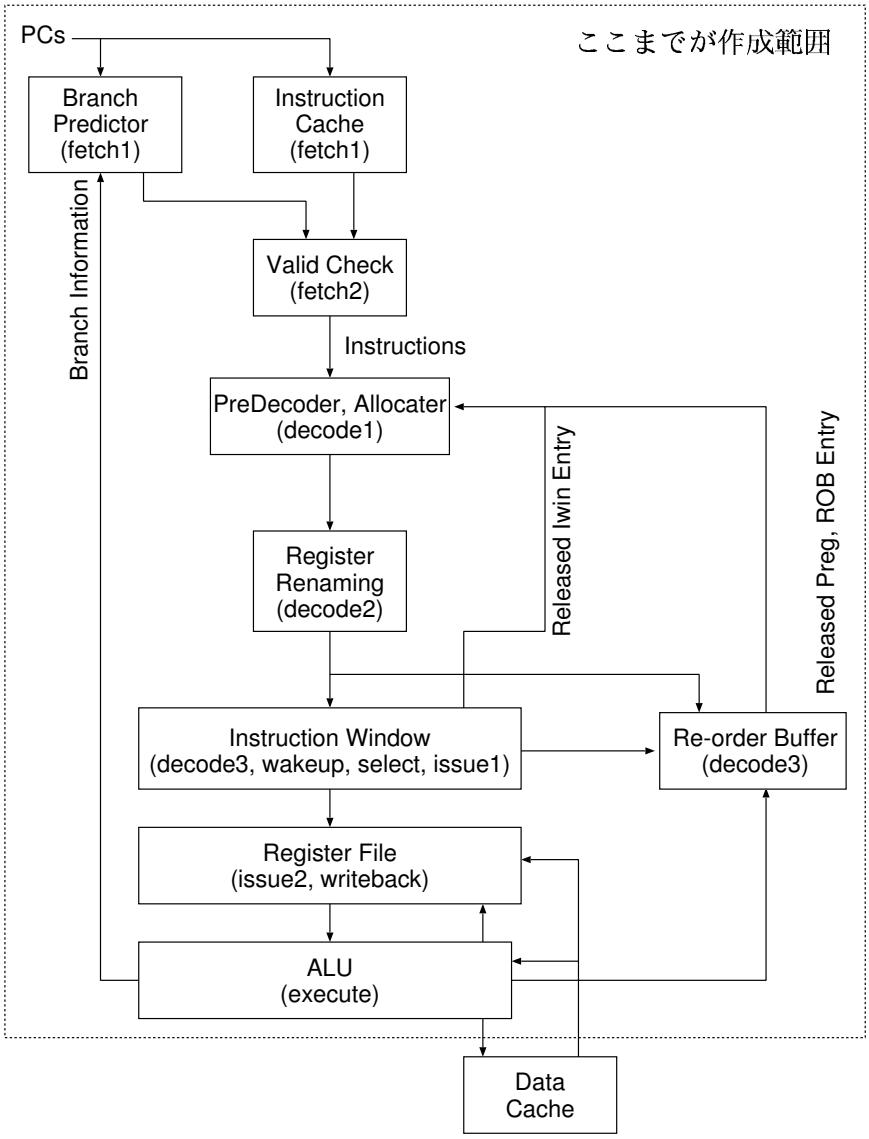

図 3.1: プロセッサ概要

今回作成するプロセッサでは FPU は搭載しない．浮動小数点の計算を行う場合は必要に応じて soft-float ライブラリを使用する．また命令ウィンドウは従来より提案されてきた，データに付けられたタグの連想処理を用いる形式ではな

表 3.1: プロセッサの構成

命令セット		MIPS 命令セット
命令発行		アウトオブオーダ
フェッチ，デコード幅		4
命令ウィンドウ		16 エントリ
機能ユニット		整数 ALU × 2 整数乗除算 × 1 ロード／ストア × 1
分岐予測	予測方式	gshare
	履歴	10 ビット
	インデクス	10 ビット
	BTB	1024 エントリ/4-way
キャッシュ	L1 命令	16KB/32B ライン/1-way
	L1 データ	16KB/32B ライン/1-way

く，依存行列テーブル (DMT: Dependency Matrix Table) と呼ばれる方式を使用する．DMT とは五島らによって提案された手法 [4, 5] で，従来より少ないメモリ量かつ連想記憶メモリ (CAM: Contents Addressable Memory) を使用せずに命令の依存関係を判断する事が可能となる．

図 3.1 に記述してあるように，実際に作成したモジュールは．フロントエンド部分，実行コアの一部 (命令ウィンドウと ALU の一部の機能) とバックエンド部分の一部 (Register File への読み書き) であり，バックエンド部分におけるコミットや例外処理の部分は実装していない．これはフロントエンド部分とバックエンド部分の処理に割り当てられているステージ数を比べた場合，フロントエンド部分の処理に必要なステージ数のほうがバックエンド部分の処理に必要なステージ数に比べて圧倒的に多く，PSU による消費電力の削減のほとんどはフロントエンド部分で達成されているためである．

3.2 機能モジュール

本節では作成した機能モジュールの解説を行う．表 3.2 にプロセッサ全体の処理の流れを示す．

表 3.2: 各ステージごとの処理

ステージ	作業内容
fetch1	命令キャッシュから命令を読み込む
	BTB，gshare から分岐判定を読み込む
fetch2	分岐判定と PC から命令が有効かどうか調べる
decode1*	プリデコード
	Dst に対して Preg，ROB，Iwin のエントリを割り当てる
	Map Table から Src1，Src2 に割り当てられた Preg，Iwin のエントリを取得する
decode2*	decode1 で割り当てられた Dst の Preg，Iwin のエントリを Map Table に登録する
	IDD を使って同時にデコードする命令間の依存関係を調べる
	IDD の結果と Map Table の結果から Src1，Src2 に割り当てる Preg，Iwin のエントリを決定する
decode3	リオーダ・バッファ，DMT，Payload RAM にエントリを登録する
wakeup	DMT のレディ・ビットを更新する
select	DMT のレディ・ビットが立っている命令から発行する命令を選択する
issue1*	発行される命令を Payload RAM から読み出す
	発行される命令に対応する DMT のエントリをクリアする
issue2*	発行された命令の Src1,Src2 に対応するレジスタ値を Register File から読み出す
execution	発行された命令を実行する
writeback*	実行が終わった命令のうち，Resister File に書き込む命令がある場合は書き込む

(*は 2 サイクル必要な場合があるステージを示している)

表 3.2 に書いてあるようにこのプロセッサにおいては，命令フェッチに 2 サイクル，デコードおよびディスパッチに 3 サイクル，スケジューリングに 2 サイクル，発行に 2 サイクル，実行に 1 サイクル，そしてライトバックに 1 サイクルを割り当てている．

3.2.1 命令フェッチ

命令フェッチは fetch1，fetch2 のステージから成り，図 3.2 の構成をしている．

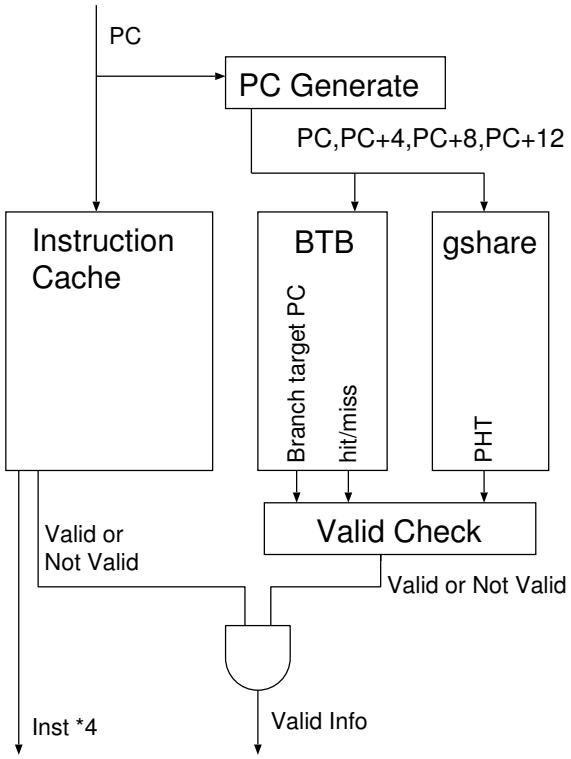

図 3.2: 命令フェッチ部分

- PC Generate
  与えられた PC の値を元に PC，PC+4，PC+8，PC+12 の値を生成し，BTB，gshare に送る．
- BTB(Branch Target Buffer)
  読み出し時は与えられた PC の値をインデクスとして，タグと分岐先アドレスを取得する．書き込み時には実行ステージで取得したタグと分岐先アドレスを書き込む．
- gshare
  読み出し時は，与えられた PC の値とグローバル履歴を XOR した値をインデクスとしてカウンタ値を取得する．書き込み時には実行ステージから送られてくる分岐の結果を元に，更新したカウンタ値を書き込む．
- Instruction Cache

与えられたPCの値をインデクスとしてキャッシュ・ラインを読み出す．PCの値から4命令が同じキャッシュ・ライン上にない場合と判断された場合は，キャッシュ・ライン上にない命令をNOP命令とし，その命令のValidフラグを0にする．

- Valid Check
  BTB，gshareの結果から，分岐成立と予測した命令がある場合はPCの値をBTBの出力した分岐先アドレスの値に更新し，分岐命令以降の命令のValidフラグを0にする．

最後に，Instruction Cacheから送られてくるValidフラグとValid Checkで取得したValidフラグの論理積を取り，取得した4つの命令のValidフラグとする．

3.2.2 デコーダ

デコーダはdecode1，decode2，decode3のステージから成り，図3.3の構成をしている．

- Pre-Decoder
  4つの命令列から以下の情報を抜き出す．
  - 各論理レジスタ番号 (Dst，Src1，Src2) およびそのレジスタが有効かどうかを示すフラグ
  - 即値およびオフセットの有効フラグ
  - 命令の種類 (命令がint ALU，int Multi.，ロード/ストアのどれか)

  以下のデコード・ステージでは，Pre-Decoderによって取得された論理レジスタ番号を使用する．
- Allocate
  デスティネーション・オペランドに対して，物理レジスタ番号 (Preg)，そのPregに結果を出力する命令のDMT内の位置 (Iwin)，リオーダ・バッファ (ROB) のエントリを割り当てる．
- Map Table
  Map Tableより，decode1のフェーズで取得した各ソース・オペランドに対するエントリを読み出し，物理レジスタ番号とIwinを取得する．同時に各デスティネーション・オペランドに対するエントリを読み出し，物理レジスタ番号 (prev-Preg) とIwin(prev-Iwin) を取得する．また，decode2のフェーズで，デスティネーション・オペランドに対して新しく割り当てられた物理レジスタ番号とIwinをエントリに記入する．

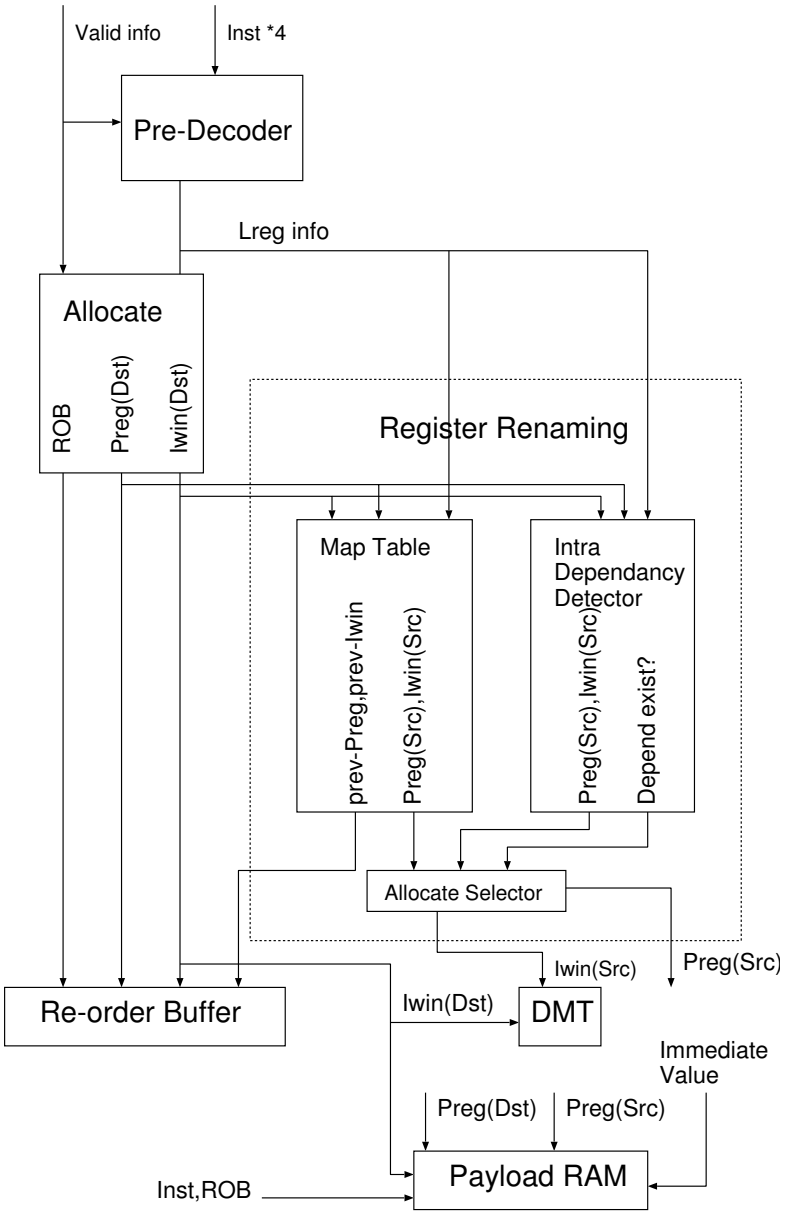

図 3.3: デコーダ部分

- Intra Dependancy Detector(IDD)
  同時にデコードされた命令間の依存関係を調べる．依存関係が存在する場合は，依存関係が存在するというフラグを立て，先行する命令に割り当てられた物理レジスタ番号と Iwin を取得する．
- Allocate Selector
  IDD から送られてきた依存関係の信号を調べる．フラグが立っている場合は，ソース・オペランドに対する物理レジスタと Iwin を IDD から取得し

た値に，立っていない場合は Map Table から取得した値とする．

- DMT
  内部には Src1 用と Src2 用の RAM モジュールが存在する．DMT の Iwin(Src) と Iwin(Dst) の交互にビットを立てることにより，DMT に命令間の依存関係を登録する．
- Payload RAM
  Iwin(Dst) に対応するエントリに対して，命令列，リオーダ・バッファ(ROB) のエントリ，Dst,Src1,Src2 のオペランドに対する物理レジスタ番号，即値もしくはオフセットを書き込む．これらの値は，命令の発行時に読み出され，実行時に使用される．
- Re-order Buffer
  ROB に対応するエントリに対して，Preg(Dst)，Iwin(Dst)，Map Table より取得した prev-Preg，prev-Iwin を書き込む．これらの値は，分岐予測ミスや例外発生時に Map Table や DMT を元の状態に戻す際に利用される．

3.2.3 スケジューリングおよび発行

スケジューリングおよび発行は wakeup，select，issue1，issue2 のステージから成り，図 3.4 の構成をしている．

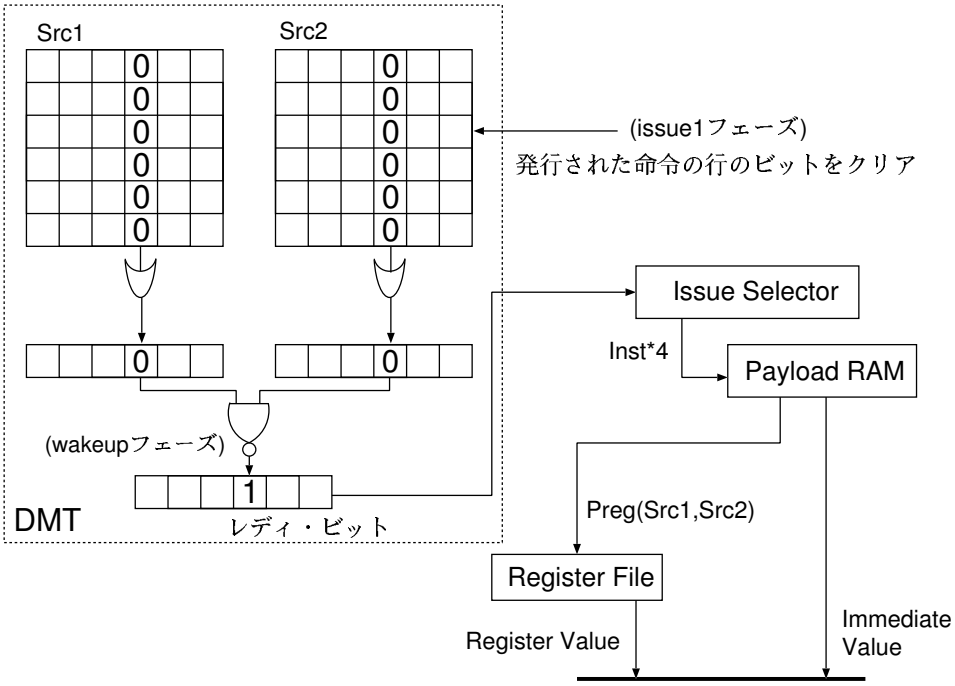

図 3.4: スケジューリングおよび発行

- DMT

  wakeup フェーズで，Src1，Src2 それぞれの列のビットの論理和となるビット列を求める．次にビット列同士の否定論理積を求め，そのビット列をレディ・ビットとする．これにより，Src1，Src2 両方の列のビットが 0 であるような命令に対して，レディ・ビットが立つことになる．また issue1 のフェーズで，発行された命令に対応する行のビットをクリアする．
- Issue Selector

  select フェーズにおいて DMT から送られてきたレディ・ビットからどの命令を発行するかを決定する．

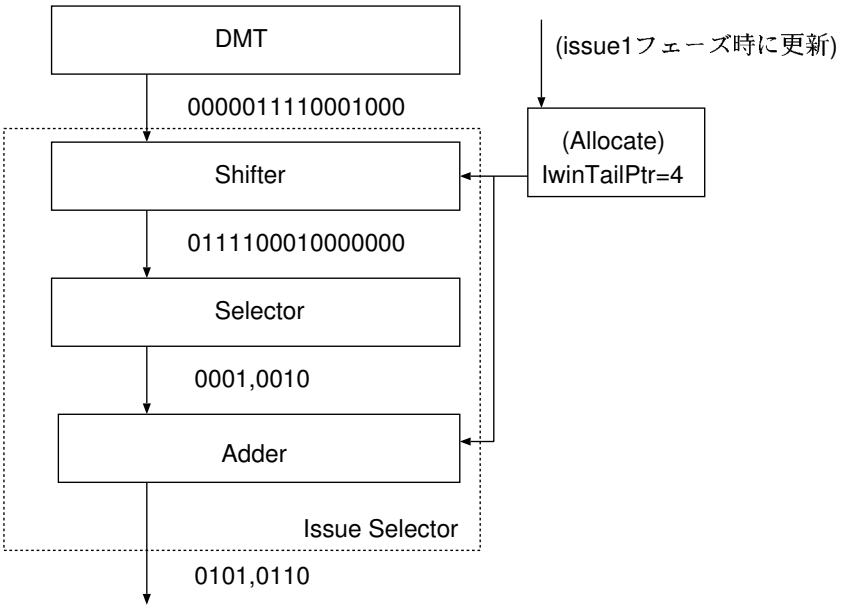

図 3.5: Issue Selector の実装

Issue Selector の内部は図 3.5 のようになっている．

あらかじめ，Allocate モジュール内に IwinTailPtr というカウンタを用意しておく．このカウンタは発行した命令数に合わせて値が増えていく．issue1 のフェーズで，発行した命令の数だけ IwinTailPtr のカウンタ値を増やす．select フェーズではまず，DMT より送られてきたレディ・ビットを IwinTailPtr 分だけ左にシフトしたビット列を新しいレディ・ビットとし Selector に送る．次に Selector では送られてきたビット列を左から見ていき，一番最初にビットの立っているインデクスを返す．その個数は int ALU では 2

個，int Multi. とロード/ストアでは 1 個である．最後に，Selector から送られてきたインデクスと IwinTailPtr のカウンタ値を加算して発行する命令のインデクスとする．

図 3.5 の例では，DMT から送られてきたビット列を IwinTailPtr のカウンタ値である 4 つ分左にシフトしている．その後，Selector でインデクス 0001,0010 を取得し，最後に IwinTailPtr のカウンタ値 4 を加算した 0101,0110 というインデクスを取得する．

- Payload RAM
  issue1 のフェーズにおいて，Issue Selector から送られてきた発行される命令のインデクスを元に，エントリから発行される命令の情報を読み出す．
- Register File
  Payload RAM より読み出された Preg(Src1,Src2) の値を元に，issue2 のフェーズにおいて Register File に格納されているレジスタ値を読み出す．

3.2.4 実行およびライトバック

実行およびライトバックは execution，writeback のステージであり，図 3.6 の構成をしている．作成範囲は図 3.6 に示してあるように点線の範囲のみである．

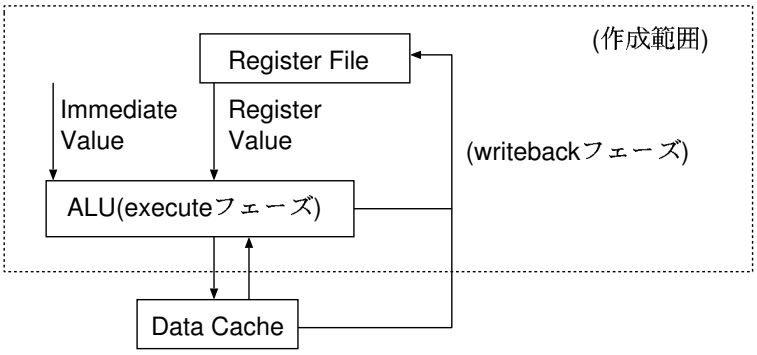

図 3.6: 実行およりライトバック部分

- ALU
  フロントエンドの設計に注力するため，今回作成したプロセッサでは ADD，SUB，ADDI の命令のみ実行可能な int ALU のみを実装した．
- Register File への書き込み
  writeback のフェーズで，ALU での演算結果を Register File に書き込む．

3.3 RAMを構成要素とするモジュール

本節ではRAMを構成要素とするモジュールについて説明する．作成するプロセッサにおいて，RAMを構成要素とするモジュールは以下の通りである．

- Re-order Buffer
- BTB
- gshare内のパターン履歴表
- gshare内のグローバル履歴
- DMT
- 命令キャッシュ
- データキャッシュ
- Register File
- Map Table
- Payload RAM

上記に記してあるように，RAMを構成要素とするモジュールは全部で10個存在する．作成するプロセッサにおいてモジュールは全部で18個なので，半分以上のモジュールがRAMを構成要素としていることになる．

3.3.1 RAMモジュールにおける制限とその回避策

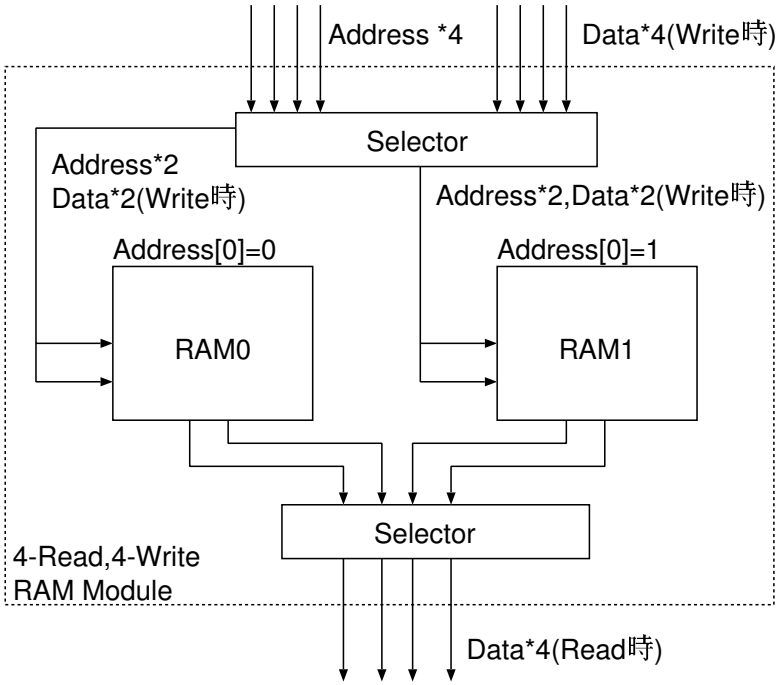

図3.7: RAMモジュールの実装方法

作成するプロセッサは 4 命令同時発行のため，基本的に RAM モジュールにおいては 4-Read，4-Write，つまり 1 回のアクセスにおいて与えられた 4 つのアドレスをインデクスとしてデータが読み書きできる性能が要求される．しかし，QuartusII がサポートしている RAM モジュールは，最大で 2-Read，2-Write である．そこで 4-Read，4-Write の RAM モジュールを図 3.7 のように実装した．

図 3.7 のように与えられた 4 つのアドレスを下位 1 ビットの位により 2 つに分けた上で，本来の半分の容量を持つ 2-Read，2-Write の RAM モジュールに対して読み書きを行う．この場合，もし与えられた 4 つのアドレスの下位 1 ビットが 3 つ以上同じ場合，正常な読み書きができない．

例えばデータ読み出し用として与えられた４つのアドレスが，***000，***010，***100，***011 のような場合だと下位 1 ビットが 0 のアドレスが 3 つ存在する．

そのような場合はパイプライン・ストールを発生させた上で，読み書きにもう 1 サイクル使用する．先程の例だと，1 サイクル目では***000，***010，***011 のアドレスに対して読み出しを行い，2 サイクル目で***100 のアドレスに対してデータの読み出しを行う．

連続した 4 アドレスの場合は読み出し，書き込みの際には問題は発生しない．また読み出し，および書き込み時に 4 ポートを全部使う，つまり 4 つのアドレスをすべて使用するとは限らないので，結果としてパイプライン・ストールは発生しにくい．

3.3.2 作成する RAM モジュール

ここでは作成する RAM モジュールの内部の構成について説明する．まずアドレスが連続しており，書き込み，あるいは読み出しが必ず 1 サイクルで終了する RAM モジュールについて説明する．

- Re-order Buffer
  リオーダ・バッファは 4-Read，4-Write となる．アドレスは Read/Write とも連続した 4 アドレスなので，読み書きに 2 サイクル発生する事はない．
- BTB
  分岐先バッファ(BTB: Branch Target Buffer) は 2 つに分かれている．一方には命令アドレスに対応するタグが，もう一方には分岐先アドレスが格納されている．4-Read，1-Write で構成されており，Read 時のアドレスは連続した 4 アドレスなので読み出しは 1 サイクルで終了する．
- gshare 内のパターン履歴表

パターン履歴表 (PHT: Pattern History Table) は 4-Read，1-Write で構成されており，Read 時のアドレスは連続した 4 アドレスなので，読み出しは 1 サイクルで終了する．

- gshare 内のグローバル履歴
  1-Read，1-Write の 10bit の FIFO で構成されている．
- DMT
  DMT は 3.2 節で説明したように各行のビットをクリアする，あるいは各列のビットの論理和を取る必要があるので，QuartusII の RAM モジュールを使用せず，レジスタを 256 個並べた構成となっている．そのため，3.3.1 節で記されているような RAM モジュールの制限はない．加えて，今回は命令がどの機能ユニット (int ALU，int Multi.，ロード/ストア) を使用するのかを記録しておくビット列 (16bit) を各機能ユニットごとに用意する．
- 命令キャッシュ
  命令キャッシュは 1-Read，1-Write で構成されている．今回設計するプロセッサでは命令キャッシュは読み出し専用とし，キャッシュのリプレースは行わないものとする．キャッシュ・ラインは 32byte であり，1 命令は 4byte のため，8 命令が格納されている．命令キャッシュはこのキャッシュ・ライン 512 本より構成されている．
- データ・キャッシュ(今回作成するプロセッサでは実装しない)
  命令キャッシュと同じ大きさであり，1-Read，1-write で構成される．

次にアドレスが連続ではなく，書き込み，あるいは読み出しに 2 サイクル必要な場合がある RAM モジュールについて説明する．

- Register File
  Register File は 32bit のレジスタ値が 128 個格納できる．MIPS の論理レジスタの個数である 32 個より多い理由はレジスタ・リネーミングを行うためである．8-Read，4-Write なので図 3.8 の様に 4-Read，4-Write の RAM モジュールを 2 つ並べた構成となる．1 つの RAM モジュールは命令 1，2 のソース・オペランド用であり，もう 1 つのモジュールは命令 3，4 のソース・オペランド用となる．
  読み出し時はそれぞれの RAM モジュールにアクセスしてデータを読み出し，書き込み時は同じデータを両方の RAM モジュールに書き込む．読み出し，書き込み共に連続した 4 アドレスとは限らないので，どちらと 2 サ

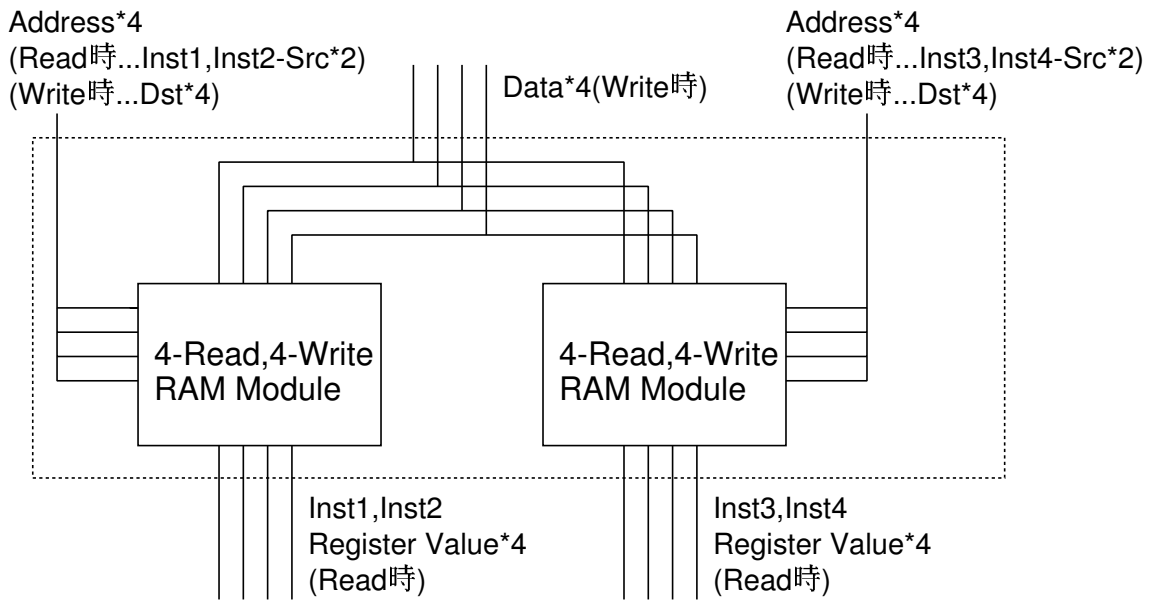

図 3.8: Register File の実装方法

イクル必要な場合がある．ただしすべての命令がソース・オペランドを 2 つ持つということはなく，また，すべての命令がデスティネーション・オペランドに対して書き込みを行うとは限らないので，結果としてパイプライン・ストールは発生しにくい．

- Map Table

  Map Table は 12-Read(デスティネーション・オペランド用，命令 1，2 のソース・オペランド用，命令 3，4 のソース・オペランド用)，4-Write なので 4-Read，4-Write の RAM モジュールを 3 つ並べた構成となる．Register File 同様，読み出し，書き込み共に連続した 4 アドレスとは限らないので，どちらとも 2 サイクル必要な場合がある．

- Payload RAM

  Payload RAM は 4-Read，4-Write の構成となる．書き込み時は連続した 4 アドレスなのでパイプライン・ストールは発生しないが，読み出しは連続した 4 アドレスとは限らないので 2 サイクル必要な場合がある．また，Payload RAM は従来のリザベーション・ステーションと異なり，命令の発行を判断する機能が実装されていない (命令が発行できるかどうかは DMT の出力であるレディ・ビットによって決定される)．

第4章　消費電力の評価結果

4.1　評価方法，評価条件

消費電力の評価にはQuartusIIに付属するPowerPlay Power Analyzer Toolを使用した．簡単なMIPSの命令列を実行した上でプロセッサの消費電力を評価した．

4.2　PSUによる消費電力削減

今回作成したプロセッサではパイプライン・レジスタは設置したがパイプラインの動作は行っていない．

プロセッサ全体の消費電力およびその割合を表4.1に示す．

表4.1: プロセッサ全体の消費電力の割合

ブロック名	消費電力	%
クロック・ドライバ	7.24[mW]	15.86%
I/O	15.45[mW]	33.85%
機能ブロック	22.94[mW]	50.27%
プロセッサ全体	45.63[mW]	100.00%

次に各ステージおよびステージ間にあるパイプライン・レジスタにおける消費電力およびその割合を表4.2に示す．

まず，表4.1からクロック･ドライバの消費する電力がプロセッサ全体の15.86%を占めている事が分かる．2.2節より，PSUはクロック・ドライバの消費電力も削減できる事が示されている．クロック・ドライバについては，測定ツールの仕様により各パイプライン・レジスタで消費されるクロック・ドライバの消費電力を測定できなかったため，本研究ではクロック・ドライバの消費電力の削減の評価は行っていない．しかし，PSUは停止したパイプライン・レジスタ数に応じて最終段のクロック・ドライバを停止でき，なおかつ最終段のクロック・ドライバはクロック・ドライバの消費電力の88%を占めるというデータがあるため[6]，このクロック・ドライバの消費電力を大きく削減できる可能性がある．

次に，表4.2から，パイプライン・レジスタ全体の消費電力は3.33[mW]であると分かる．これはプロセッサ全体の消費電力のうち7.30%を占める．このプ

表 4.2: 各ステージおよびパイプライン・レジスタにおける消費電力

ステージおよびパイプライン・レジスタ	消費電力	%
fetch1	0.85[mW]	3.75%
Pipeline Register1(fetch1-fetch2)	0.69[mW]	3.05%
fetch2	0.06[mW]	0.26%
Pipeline Regisetr2(fetch2-decode1)	0.71[mW]	3.13%
decode1	3.52[mW]	15.54%
Pipeline Register3(decode1-decode2)	0.73[mW]	3.22%
decode2	0.12[mW]	0.53%
Pipeline Register4(decode2-decode3)	0.57[mW]	2.52%
decode3	2.68[mW]	11.83%
wakeup	1.06[mW]	4.68%
Pipeline Register5(wakeup-select)	0.26[mW]	1.15%
select	0.14[mW]	0.62%
Pipeline Register6(select-issue1)	0.05[mW]	0.22%
issue1	0.60[mW]	2.65%
Pipeline Register7(issue1-issue2)	0.19[mW]	0.84%
issue2	6.66[mW]	29.40%
Pipeline Register8(issue2-execute)	0.13[mW]	0.57%
execute	0.31[mW]	1.37%
writeback	3.32[mW]	14.66%
Total	22.65[mW]	100.0%

ロセッサに対してパイプライン・ステージ統合を行う機構を実装した場合，例えば，fetch1 と fetch2，decode1 と decode2，wakeup と select，および issue1 と issue2 のステージを統合することが可能となる．その場合パイプライン・レジスタのうち，Pipeline Register1，Pipeline Register3，Pipeline Register5，Pipeline Register7 へのクロック・ドライバの供給を停止することができ，1.87[mW] の消費電力を削減することができる．これはプロセッサ全体の消費電力のうち 4.10%，パイプライン・レジスタ全体の消費電力のうち 56.16%を占める．

表 4.3: 各モジュールにおける ALUT 等の使用量

モジュール	ALUT 数	レジスタ数	メモリ・ビット数
PC Generate	122	0	0
BTB	123	0	53248
gshare	10	0	2948
Instruction Cache	273	0	131072
Pipeline Register1	120	120	0
Valid Check	273	0	0
Pipeline Register2	183	178	0
Pre-Decoder	145	0	0
Allocate	309	18	896
Map Table	1766	36	768
Pipeline Regisere3	98	93	0
IDD	74	0	0
Allocate Selector	94	0	0
Pipeline Register4	95	95	0
DMT	1877	528	0
Payload RAM	1166	5	1408
Pipeline Register5	16	16	0
Issue Selector	163	0	0
Pipeline Register6	12	10	0
Register File	903	18	4096
ALU	198	0	0

最後に，表 4.3 に各モジュールにおける ALUT(Adaptive LookUp Table) 数，レジスタ数，メモリ・ビット数を示す．ALUT とは FPGA の基本的な回路構成要素のことである．なお，表 4.3 にはリオーダ・バッファおよび Pipeline Register7, Pipeline Register8 が含まれていない．リオーダ・バッファに関しては，書き込みは行われているが読み出しが行われていないので，QuartusII の最適化によって機能モジュールが生成されなかったためと思われる．また Pipeline Register7, Pipeline Register8 に関してはそれぞれのレジスタが，Payload RAM と Register

File 内に存在するため ALUT 等の値が Payload RAM と Register File の結果に含まれているためだと思われる．

第 5 章　おわりに

近年のモバイル・プロセッサには消費電力の削減が求められており，その要求に応えるために，PSU という方式が提案されている．過去の研究により，PSU は現在主流の消費電力削減手法である DVS よりも消費電力の削減量が大きいことが示されている．また，その有効性は半導体の製造技術が進歩するにつれて大きくなることが示されている．しかし，今までの PSU に関する研究は，アーキテクチャ・レベルのシミュレーション上でのみ行われており，実際のプロセッサ上ではどれほど消費電力が削減されるのかは不明であった．

そこで，本研究ではプロセッサにおける消費電力の評価として Verilog HDL を用いてプロセッサを設計/実装し，クロック・ドライバやパイプライン・レジスタ等の消費電力を評価した．

プロセッサの総消費電力におけるパイプライン・レジスタの消費電力の割合は 7.30%，またクロック・ドライバの消費電力の割合は 15.86% を示した．次にこのプロセッサに PSU を適応してステージを統合した場合，パイプライン・レジスタの消費電力を 56.16%削減できるという結果を得た．

この評価結果から，アーキテクチャ・レベルのシミュレーションの評価結果の信頼性がより高まると考えられる．

謝辞

本研究の機会を与えてくださった富田眞治教授に甚大なる謝意を表します．

また本研究に関して適切なご指導を賜わった富田研究室の嶋田創助手，奈良先端科学技術大学の中島康彦教授，福井大学の森眞一郎教授，本学の法学部情報担当助手である三輪忍氏に心から感謝いたします．

さらに，日頃からご討論頂いた京都大学情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝いたします．

参考文献

[1] MIPS Technologies: *MIPS R4400 Microprocessor User's Manual*, (1994).

[2] 嶋田創, 安藤秀樹, 島田俊夫: “低消費電力化のための可変パイプライン”, 情報処理学会研究報告, 2001-ARC-145, pp. 57–62, (2001).

[3] 嶋田創, 安藤秀樹, 島田俊夫: “パイプラインステージ統合によるプロセッサの消費エネルギーの削減”, 情報処理学会論文誌，コンピューティングシステム, Vol. 45, No. SIG 1(ACS 4), pp. 18–30, (2004).

[4] 五島正裕, 西野賢悟, グェンハイハー, 懸亮慶, 中島康彦, 森眞一郎, 北村俊明, 富田眞治: “スーパースケーラのための高速な動的命令スケジューリング方式”, 情報処理学会研究報告, 2001-ARC-142, pp. 121–126, (2001).

[5] 五島正裕, 西野賢悟, 小西将人, 中島康彦, 森眞一郎, 北村俊明, 富田眞治: “行列に基づく Out-of-Order スケジューリング方式の評価”, 情報処理学会論文誌, ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG 6(HPS 5), pp. 13–23, (2002).

[6] Anderson F.E., Wells J.S. and Berta E.Z.: The Core Clock System on the NextGeneration Itanuim Microprocessor, *2002 IEEE international Solid-State Circuits Conf. Visual Supplement to the Digest of Technical Papers*, pp. 110–111, (2002).