特別研究報告書

低電力 Java プロセッサのための
投機的クロック制御方式

指導教官　　富田 眞治 教授

京都大学工学部情報学科

木村 篤彦

平成 13 年 2 月 13 日

低電力 Java プロセッサのための
投機的クロック制御方式

木村 篤彦

内容梗概

近年，地球温暖化が深刻な問題になってきている．増加の一途をたどっている電力需要量は，二酸化炭素の発生，ひいては地球温暖化に拍車をかける結果となっており，電子機器の低消費電力化は，危急の課題である．一方，Java 技術が大きな注目を集めている．プログラムをネットワークにおいて安全に転送できることが，ネットワーク社会の進展により重要視されてきたためであり，今後，様々な電気製品に組み込まれていくことが考えられる．

本論文では，OMRON 株式会社が開発した Java プロセッサ JeRTy をモデルとし，使用する演算ユニットのみを起動することにより消費電力を削減する方法の提案および定量的評価を行った．具体的には，クロックを停止することのできるゲート付きクロック（Gated Clock）を用いて演算ユニット単位のクロック制御を行い，使用する演算ユニットのみを起動することにより消費電力の削減を図る．ここで，命令デコードの結果から使用する演算ユニットを判定し，上流のクロックを投入/停止するクロック制御機構を仮定する．しかし，クロック分配系統の無視できない遅延のため，演算ユニットの判定が演算ステージの動作開始に間に合わない場合，性能が低下する．本論文では使用する演算ユニットを予測する小さなハードウェア機構を導入し，投機的に上流クロックを制御することにより，性能の低下を抑えつつ電力を削減する方法の提案および評価を行った．

上流クロックを起動してから演算ユニットが使用可能となるまでの遅延を起動ペナルティと呼ぶことにする．予測に反して演算ユニットが使用されない場合，演算ユニット確定後に上流クロックを起動するため，起動ペナルティ分性能が低下する．逆に，予測に反して演算ユニットを使用する場合，演算ユニット確定後に上流クロックを停止するため，起動ペナルティ分使用しないユニットが動作することになり，無駄な電力が消費される．

予測は，過去の命令実行パターンに基づいて作成した予測表を参照することによって行う．予測表には各インデックスに対して各演算ユニットの使用/不使

用の情報が格納される．インデックスは，プログラムカウンタの下位数ビット，命令のオペコード，命令を機能別に分類したグループ，または，これらの組み合わせを用い，合計 6 通りの方法を考案した．予測表のサイズを大きくすることにより予測ヒット率は向上するものの，予測表自身が電力消費を増加させるため，256 エントリまたは 16 エントリという比較的小さな予測表とした．この予測表にしたがってクロックを投機的に制御することにより性能の低下を抑えつつ電力を削減する．

効果の測定には，以上に述べた予測機構を実装した Java 仮想マシン Kaffe を用い，ワークロードには，ベンチマークプログラム SPEC JVM98 使用した．ここで，起動ペナルティを 1 および 2 サイクルを仮定した．

測定の結果，予測を行わない場合は消費電力を最大 90%，平均 85%削減できるが，サイクル数は，起動ペナルティ=1 の場合，最大 16.8%，平均 14.8%，起動ペナルティ=2 の場合，最大 33.6%，平均 29.6%増加することが明らかになった．

256 エントリの予測表を用いる場合，PC の下位 8 ビットによる予測が最も効果があった．起動ペナルティ=1 の場合は，消費電力は最大 89%，平均 85%削減することができ，サイクル数は最小 2．8%，平均 4．3%の増加に抑えることができた．起動ペナルティ=2 の場合は，消費電力は最大 89%，平均 84%削減することができ，サイクル数は最小 5．5%，平均 9．1%の増加に抑えることができた．

16 エントリの予測表を用いる場合，用いる情報による差はほとんどないものの命令を機能別に分類したグループの識別子 4 ビットによる予測が比較的効果的であった．起動ペナルティ=1 の場合は，消費電力は最大 88%，平均 84%削減することができ，サイクル数は最小 6．5%，平均 7．7%の増加に抑えることができた．起動ペナルティ=2 の場合は，消費電力は最大 87%，平均 83%削減することができ，サイクル数は最小 13%，平均 17%の増加に抑えることができた．

結論として，256 エントリの予測表においては PC の下位 8 ビット，16 エントリの予測表においてはグループの識別子 4 ビットの情報を用いることにより，使用する演算ユニットの予測が十分に可能であることがわかった．また，256 エントリの予測表では，予測による性能向上が，消費電力増加を上回るものの，16 エントリの予測表では，逆転することを明らかにする．最後に，PC の下位数ビットとメソッド識別子を用いる方法により，予測の精度をより向上させることができる可能性を示す．

A Speculative Clock Control Technique for Low-Power Java Processers

Atsuhiko Kimura

Abstract

Recently, the greenhouse effect is being recognized as a serious problem. The power demand accelerates the generation of CO_2 and the greenhouse effect as well, and the power consumption of electric machines is immediately needed to be reduced. On the other hand, Java technology is gaining much more attention, since it becomes important that Java program can be safely transferred in a network. Therefore, Java technology is being embedded to various electric appliances.

This paper shows a low-power technique for a Java processor so called JeRTy developed by OMRON Corporation. Higher-level gated clock controllers drive clock-trees by execution units and only execution units that are used are driven. In this way, I plan to save the power consumption. It is assumed that the instruction decoder decides the execution units and these controllers start and stop related higher-level clocks. But, the performance degrades if the decision of execution units in the decode stage misses the execute stage by the clock delay. In this paper, I introduce a small hardware that predicts the execution units. And, I propose and evaluate the high-performance and low-power technique, that is to say the speculative higher-level gated clock controllers.

The penalty is the delay which it takes before a higher-level clock is distributed to an execution unit. When an execution unit is used against the prediction, a higher-level clock is driven after the decision of execution units. And the performance degrades for the "penalty" cycle. On the contrary when an execution unit is unused against the prediction, a higher-level clock stop after the decision of execution units. And the power increases for the "penalty" cycle, since execution units that are unused are driven in the "penalty" cycle.

How to predict execution units is to refer to the prediction table from an index. The prediction table has the information that every execution unit is used or unused. I try 6 different ways about how to select the index. The index is lower some-bits of the program counter, opecodes of the introductions and

groups divided the introductions by their functions or these combinations. The larger table size is, the better hit rate is. But a large table increases power consumption in itself. And I assume the table with 256-entries and 16-entries, a relatively small table. To control clock with these prediction tables maintain the performance and reduce the power dissipation.

I implemented these prediction schemes on Java Virtual Machine Kaffe and evaluate with SPEC JVM98 benchmark programs. I assume that the "penalty" is 1 or 2.

Non-prediction scheme reduces 95% (maximum) and 85% (average) power consumption of execution units, increases 16.8% (max.) and 14.8% (av.) cycle in the case that the "penalty" is 1, and increases 33.6% (max.) and 29.6% (av.) cycle in the case that the "penalty" is 2.

A prediction scheme with lower 8-bits of program counter is most effective in the table with 256-entries. In the case that the "penalty" is 1, this scheme reduces 89% (max.) and 85% (av.) power consumption, and increases 2.8% (minimum) and 4.3% (av.) in cycle. In the case that the "penalty" is 2, this scheme reduces 89% (max.) and 84% (av.) power consumption, and increases 5.5% (min.) and 9.1% (av.) in cycle.

In the table with 16-entries the effect has little difference by every information. A prediction scheme with 4-bits each corresponds to the groups divided the introductions by their functions is relatively effective. In the case that the "penalty" is 1, this scheme reduces 88% (max.) and 84% (av.) power consumption, and increases 6.5% (min.) and 7.7% (av.) in cycle. In the case that the "penalty" is 2, this scheme reduces 87% (max.) and 83% (av.) power consumption, and increases 13.7% (min.) and 17.1% (av.) in cycle.

The conclusion is that the prediction table indexed by lower 8-bits of program counter in the table with 256-entries, 4-bits each corresponds to the groups in the table with 16-entries is remarkable effectiveness for unit level usage prediction. And the table with 256-entries gains higher performance than power consumption, though the table 16-entries dissipates the power. Finally, there is possibility that hit rate can improve more by the prediction scheme with lower some-bits of program counter and some-bits each correspond to methods.

低電力 Java プロセッサのための
投機的クロック制御方式

目次

第 1 章　はじめに

現在，地球環境に関するさまざまな指摘がなされている．なかでも，地球温暖化の問題は21 世紀初頭の最大の問題になりつつある．現代社会を支えている電気製品を動かす電力の多くの部分は，石油をはじめとする化石燃料を燃やして供給されており，電気製品の消費電力も地球温暖化の一因となっている．通信情報化社会の急速な進展に伴い，パソコンや携帯電話をはじめとして急速に台数を増やしている電子機器に対して，低消費電力化が求められている．

さらにネットワーク社会の進展に伴い，現在より機能が増強された次世代電気製品が登場し，また，常時電源を ON にしたままの電子機器が増えることが予想される．低消費電力化は地球環境，社会負担など広範な影響を与える重要な問題となってきている．

一方，次世代電気製品などの組込み分野に，Java 技術を適用しようという動きが広まっている．Java 言語は，Smalltalk や C++に代表されるオブジェクト指向プログラミング言語に分類される．ネットワークコンピューティングを設計思想の中心に据えている点が従来の言語と大きく異なる点であり，ネットワーク対応が重要視されつつある次世代電気製品において，より高速かつ高機能なサービスを提供するという要求に合致する．

Java 言語は，コンパイラによって Java バイトコードと呼ばれる中間コードに変換することにより実行可能となる．Java バイトコードにはネットワークに適応するための様々な工夫がなされている．まず，Java バイトコードのサイズは C++から得られる実行形式ファイルよりも極めて小さい．このため，ネットワークにおいてファイルの転送を高速に行うことができる．また，Java 言語には有害なプログラムを排除するようなセキュリティ機構を組み込んであるため，安全性が高く，ネットワーク上のプログラムのやりとりに適している．

Java バイトコードは Java 仮想マシン（JVM）と呼ばれる仮想の計算機上で実行される．Java バイトコードは特定のプラットフォームに依存しないコード形式であり，特定のプラットフォームを意識したものではなく，JVM さえ移植すれば，あらゆるシステムで同一の Java プログラムを実行できる．Java のソースコードを書き直したり，ソースコードを再コンパイルする必要はない．これが，Java プログラムがプラットフォームに依存しないと言われる理由である．また，JVM はクラスファイル・フォーマットの知識のみを保持しており，Java

言語の知識は保持していない．クラスファイルとは，Java 言語をコンパイルして生成される実行形式ファイルであり，Java バイトコード，実行に使用する定数を一括して管理するコンスタントプールおよび他の付随的な情報を保持したものである．つまり，JVM は単にクラスファイルを読み込んで，それを実行するだけである．[5, 6]

さて，現在までに低消費電力化技術は様々な方式が考えられている．その中にクロックそのものをオン/オフするという重要な方式がある．長い間，クロック設計の一般的な考え方は，信号はできるだけ乱してはならず，回路設計者がクロック信号を中断してはならないということであった．しかし，近年，電力消費の主な原因は，演算ユニットの不必要な動作とクロック分配系統自身によるものであることが理解されてきた．したがって，クロックを停止することができるゲート付きクロック（Gated Clock）を導入することにより，消費電力を大きく削減できると考えられる．

クロック分配系統の消費電力をより多く削減するには，より上流のクロックを制御する必要がある．しかし，上流のクロックが下流クロックに届くには遅延が存在するため，クロック周波数が高いほど性能が低下する．Gated Clock による低消費電力化はいくつかの研究報告があるものの，それによる性能低下をいかに小さくするかに関する報告はまだなされていない．

本論文では，OMRON 株式会社が開発した JeRTy[1] をモデルとし，Java プロセッサにおける演算ユニットおよびクロック分配系統の消費電力を抑える低消費電力化技術の提案および評価を行った．使用する演算ユニットは命令デコードの結果によって決まる．デコードを待っていては演算開始が間に合わない場合，デコードより前の段階において使用する演算ユニットを予測する必要がある．本論文では，予測ハードウェア機構を導入することにより性能の低下を防ぐ方式を提案し．消費電力および性能について定量的評価を行う．

まず第 2 章において，現在までに報告されている低消費電力化のための様々な技術を述べる．次に第 3 章において，本研究における低消費電力化技術の詳細を述べ，第 4 章において，本方式の評価に用いた Java 仮想マシン Kaffe とベンチマークプログラム SPEC JVM98 の概要，評価結果および考察を述べる．最後に第 5 章において結論を述べる．

第 2 章　低消費電力化技術の現状

低電力化のためには，プロセッサの全体や一部をスリープモードに遷移させる方法，周波数や電圧を可変とする方法 [14, 15]，状態遷移ループを検出して遷移を止める方法 [10, 12]，パストランジスタ等低電力論理の採用 [16, 17]，SOI 等素子レベルの工夫 [11] など，様々なレベルの方法が提案されており，現在に至るまでに極めて多くの研究成果が報告されている．

低電力化をめざした商用マイクロプロセッサとしては，周波数と電圧をダイナミックに変化させる SpeedStep 技術を採用した Intel Corp. のモバイル PentiumIII，および，PowerNow!技術を採用した AMD,Inc. の K6-2+，同様の LongRun 機能に加えて VLIW の採用によりゲート数および電力を抑えた Transmeta Corp. の Crusoe があげられる．SpeedStep 技術は AC 電源使用時とバッテリー駆動時で，プロセッサの動作電圧と動作周波数を 2 段階に変化させ，それぞれの状態において最適な性能を得られるというものである．これにより，AC 電源使用時には，デスクトップパソコンに近い性能を持ち，バッテリー駆動時には，消費電力を低減することができる．一方，PowerNow!技術および LongRun 機能はプロセッサの負荷状況に応じて動作周波数と動作電圧を細かくダイナミックに変化させることにより，バッテリー駆動時においても，性能の低下を抑えることができる．さらに，Crusoe は既存の x86 系マイクロプロセサの機能を，VLIW 構造の CPU コアと Code Morphing Software と呼ぶ x86 命令を VLIW のネイティブコードにコンパイルする専用ソフトウエアの組み合わせにより実現する．この結果，他のプロセッサと比べて回路を単純化することができるため，消費電力を大幅に削減することができる．しかし，コード変換にはネイティブ命令を実行するよりも多くの演算サイクルがかかるため，Crusoe では実行時間がその分長くかかる．

最近ではアーキテクチャレベルでの低電力化に関する研究が活発である [7, 13, 9]．電力消費モデルを付加したサイクルシミュレータにより，SPEC95 などのベンチマークプログラムの消費電力をモデル化する研究などが報告されている [8, 19]．

さて，N 型と P 型トランジスタを対称に配置する CMOS 回路は，スイッチング時にのみ電荷が移動するため，本来消費電力が小さい性質がある．しかし，最近では，プリチャージが必要なダイナミック回路を多用したり，周波数を極

限まで高めることにより，特にクロック分配系統における消費電力が極めて多くを占めるようになってきている．プロセッサを構成する機能ブロックのうち，未使用ブロックへの電源供給そのものを遮断することにより，電力を削減する方法が考えられる．しかし，CMOS 回路は全体が巨大なコンデンサであるため，電源供給を再開してからそのブロックの電圧が安定するまでには，かなりの時間を要する．また，機能ブロックへの突入電流が周辺の信号線にカップリングノイズを引き起こす恐れがあることから，演算サイクル程度の時間間隔で機能ブロックの電力をこまめに節約する方法としては不向きである．このような状況では，クロックを停止することのできるゲート付きクロック（Gated Clock）を導入することにより，機能ブロック単位の電力の大幅な削減が期待できる [7, 18].

第 3 章　投機的クロック制御方式の提案

3.1　Java 仮想マシンが参照する主記憶データ構造

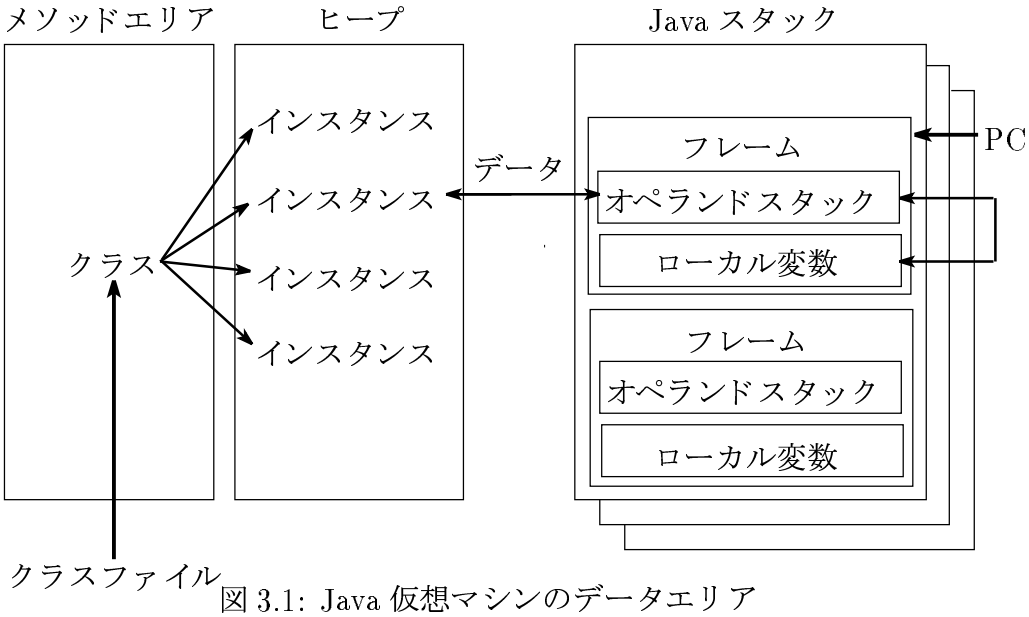


図 3.1: Java 仮想マシンのデータエリア

Java 仮想マシン（JVM）が使用するデータエリアは図 3.1に示すように大きく 3 つの領域から構成される．全体の流れとしては，まずメソッドエリアにクラスファイルを展開し，次に展開されたクラスファイルに従って，インスタンスをヒープに作成し，メソッドエリア，ヒープ，Java スタックを作業領域としてバイトコードを実行する．[5, 6]

メソッドエリア　読み込まれたクラスファイルを展開するデータ領域で，すべてのスレッド間で共有される．

ヒープ　クラス・インスタンスおよび配列の割り当てを行う実行時のデータ領域で，全てのスレッド間で共有される．使用されなくなったオブジェクトは，ガベージ・コレクションと呼ばれるプロセスによって自動的に解放され，ヒープ領域を確保する．

Java スタック　メソッドの呼出し毎にメソッドの作業用メモリであるフレームが積まれるスタックである．各スレッドごとに，スレッドと同時に生成される個別の Java スタックを保持している．各スレッドにはプログラムカウンタ（PC）が個別に存在している．また，フレームは主に次の 2 つの領域から構成される．

ローカル変数　現在実行しているメソッド中で宣言された変数の値を格納する．

オペランドスタック　現在実行しているメソッドのバイトコード命令が必要とする作業領域であり，メソッドの引数および返り値を積み上げるスタックである．JVM の大半の命令は，現在実行中のフレームのオペランドスタックから値を取得，演算した後，同一のオペランドスタックを返す．オペランドスタックはまた，メソッドに引数を渡し，その結果を受け取るためにも使用される．

プログラムカウンタ　現在実行しているメソッドで実行している命令のアドレスを保持するレジスタ．

JVM はスタックマシンであるため，オペランドスタック上の値のみが操作可能となっている．つまり，ローカル変数上の値は一度オペランドスタック上に値を移動しなければ扱うことができない．

3.2　プロセッサ構成の仮定

JeRTy をモデルとして仮定した，簡略化した Java プロセッサのモデルを図 3.2に示す．Java プロセッサは，JVM をハードウェアで構成したものである．パイプラインは Fetch，Decode，Execute，Write の 4 段とする．各ステージごとの動作を以下に示す．[4]

Fetch　プログラムカウンタにより指される次命令アドレスから次命令を読み出す．後述する予測は本ステージにおいて行う．

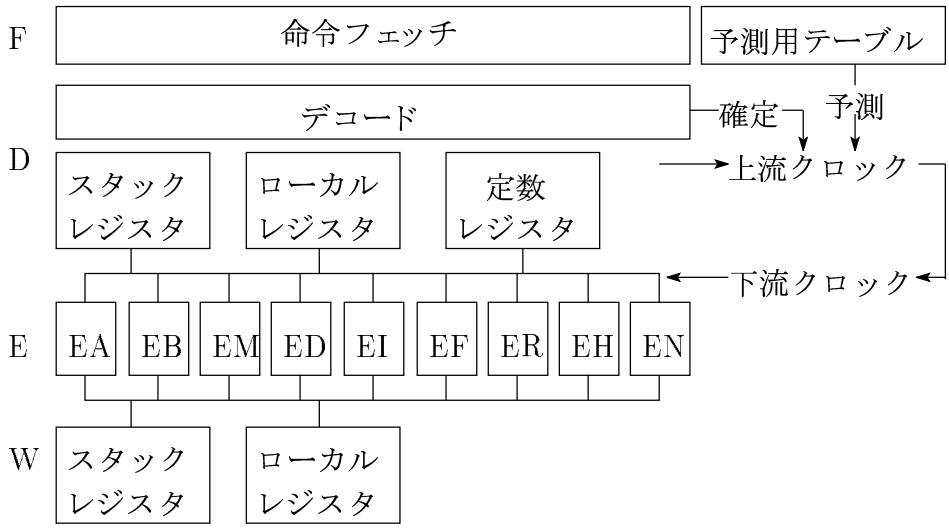


図 3.2: 仮定するパイプライン

Decode　読み出した命令をデコードし，必要なレジスタを読み出す．

Excute　演算，アドレス計算，ヒープ参照などを行う．

Write　演算結果をレジスタに格納する．

ローカルレジスタはローカル変数を格納するレジスタである．スタックレジスタはオペランドスタックに対応しており，ローカル変数の値を利用して演算を行う場合に，これらの値を一度保持しておくために用いる．また，インクリメント・デクリメント・符号拡張を行う際に必要とする定数を格納するために定数レジスタを備えている．

本論文では E ステージに関わる演算ユニットのみを評価対象とするため，E ステージについてさらに詳細な仮定を行う．このステージは 9 つの演算ユニットから構成されている．EA は基本的に 1 サイクルで終了する単純な命令，EB は分岐，EM は整数乗除算，ED は swap および複雑な dup 命令，EI はメソッド呼び出しおよびリターン，EF はフレーム生成および浮動小数点演算，ER はモニタ，EH はヒープ参照，EN は new 等オブジェクト生成に関する命令をそれぞれ担当するものとする．また，E ステージを構成する演算ユニット全体の消費電力は，プロセッサ全体の 80%を占めると仮定する．

E ステージの各演算ユニットには，対応する上流クロックから遅れて動作する下流クロックから生成される演算ユニット起動用制御信号により開閉するゲートが設けられている．各演算ユニットの上流クロックは，F ステージにおいて予測用テーブルに基づいて生成される演算ユニット選択用制御信号によりゲー

表 3.1: 所要サイクル数の仮定

演算ユニット名	EA	EB	EM	ED	EI	EF	ER	EH	EN
消費電力比率（%）	10	10	22	2	10	22	2	12	10
nop,popX,dupX	1	0	0	0	0	0	0	0	0
Xconst,Xipush	1	0	0	0	0	0	0	0	0
Xstore,Xstore_X	1	0	0	0	0	0	0	0	0
IALU,ISFT,Fneg	1	0	0	0	0	0	0	0	0
i2l,i2c,i2b,i2s,l2i	1	0	0	0	0	0	0	0	0
Xload,Xload_X	2	0	0	0	0	0	0	0	0
ifX,if_X,Icmp	0	2	0	0	0	0	0	0	0
gotoX,jsrX,ret	0	2	0	0	0	0	0	0	0
Xswitch	0	8	0	0	0	0	0	0	0
imul	0	0	4	0	0	0	0	0	0
idiv,irem	0	0	16	0	0	0	0	0	0
swap,dup_X	0	0	0	10	0	0	0	0	0
dup2_X	0	0	0	20	0	0	0	0	0
invokeX,Xreturn	0	0	0	0	18	18	0	0	0
monitorX	0	0	0	0	0	0	9	0	0
Xaload,Xastore	0	0	0	0	0	0	0	10	0
ldcX_X,astore_X	0	0	0	0	0	0	0	10	0
Xstatic,Xfield	0	0	0	0	0	0	0	10	0
arraylength	0	0	0	0	0	0	0	10	0
new,Xnewarray	0	0	0	0	0	0	0	0	20
Fadd,Fsub,Fmul,Fcmp	0	0	0	0	0	4	0	0	0
I2F,F2I,F2F	0	0	0	0	0	4	0	0	0
fdiv,frem	0	0	0	0	0	16	0	0	0
Ddiv,Drem	0	0	0	0	0	32	0	0	0
invokeinterface	0	0	0	0	0	18	0	0	0
athrow	0	0	0	0	0	18	0	0	0
checkcast,instanceof	0	0	0	0	0	10	0	0	0
multianewarray	0	0	0	0	0	10	0	0	0

トされ，続くDステージにおいて命令のデコード結果により検証される．予測が正しい場合はそのまま，間違っている場合は再び演算ユニット選択用制御信号が生成される．

表3.1の第2行に，1サイクルにつき各演算ユニットが消費する電力の演算ユニット全体に対する割合を示す．消費電力比は，各ユニットにおける論理素子の動作率が全て等しいと仮定し，JeRTyにおける論理素子数から概算したものである．また，第3行以降に，各命令の実行においてEステージに要する所要サイクル数を示す．但し，表中の略号はI=i,l，F=f,d，D=l,d，ALU=add,sub,inc,neg，SFT=shl,shr,ushr,and,or,xor，Xは任意の文字列である．

一般的なRISCプロセッサのTLBやデータキャッシュタグ機構に相当する，ヒープ参照を高速化するためのオブジェクトTLB（1024エントリ，4ウェイ，ミスペナルティ20サイクル）やヒープキャッシュタグ機構（1024エントリ，ラインサイズ64バイト，ミスペナルティ20サイクル）はEH内に実装すると仮定する．オブジェクトTLBミスおよびヒープキャッシュミスは，演算ユニットの所要サイクル数に加算され，この間演算ユニットの消費電力は0であるとする．

3.3 クロック制御方法

本説では，本論文が提案する投機的クロック制御方式について詳述する．上流クロック再開が下流クロックに届くまでの時間を無視できない場合，Dステージにおいて，使用するユニットが判明してから直ちに演算ユニットが起動することはできない．使用する演算ユニットの確定がEステージに間に合う場合を起動ペナルティ=0，クロック分配系統に無視できない遅延が存在するため間に合わない場合を起動ペナルティ>0と呼ぶことにする．起動ペナルティの単位はマシンサイクル数である．

図3.3に，起動ペナルティ=0におけるクロック制御方法の概略を示す．まず，Dステージにおいて命令デコードの結果に従って，上流クロックを投入する．次に上流クロックに遅れて下流クロックが動作することにより，Eステージの開始と同時に演算ユニットが動作し，電力が消費される様子を表している．命令デコードの結果に従っているため，当然起動した演算ユニットは使用される．このように，Eステージにおいて必要な演算ユニットだけを起動することができるため，性能を低下させることなく，消費電力を最小とすることができる．

また，起動ペナルティ>0の場合は，演算ユニットの確定を待っていると，演

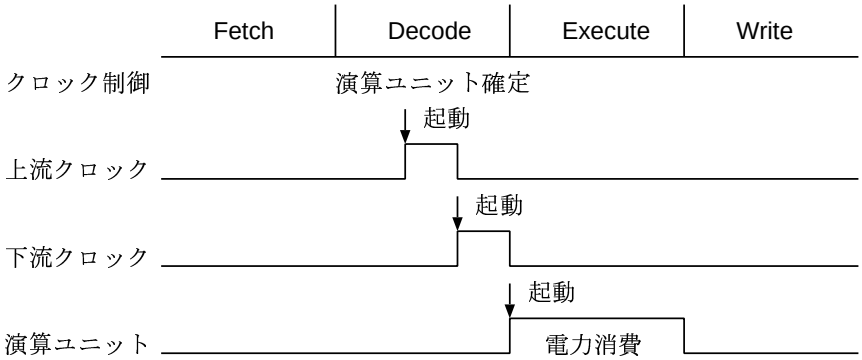


図 3.3: 起動ペナルティ=0 の場合

算ユニットの起動が E ステージに間に合わず 1 サイクル遅れるため，パイプラインがストールする．パイプラインを止めないために，演算ユニットの起動が E ステージに間に合うよう，演算ユニットが確定する以前に，使用する演算ユニットを予測し，上流クロックを投入する．

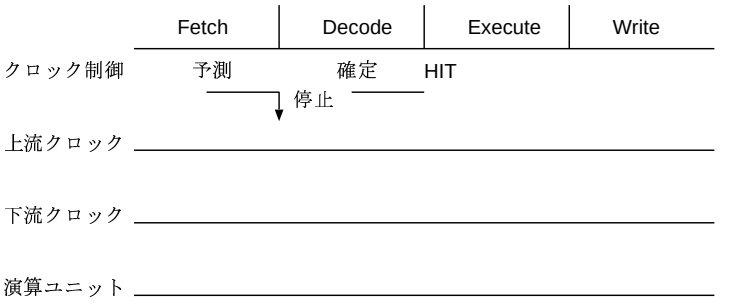
図 3.4に起動ペナルティ=1 の場合の投機的クロック制御方法を示す．ただし，(b) および (d) の場合，D ステージにおける使用ユニット確定時に予測表を更新する．

(a) 待機ヒット　まず，F ステージにおいて予測表に基づいて上流クロックを停止する．D ステージにおいて予測の検証を行うものの，予測どおり未使用であり，上流クロックは停止のままにする．下流クロックも当然起動しないため，演算ユニットは動作せず，電力は消費しない．

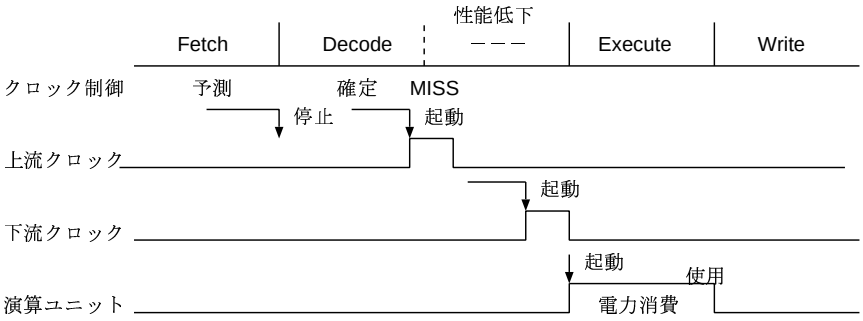
(b) 待機ミス　F ステージにおいて上流クロックを停止したものの，D ステージにおいて演算ユニットを使用することが判明し，上流クロックを投入する．遅れて下流クロックが起動し，E ステージから 1 サイクル遅れて演算ユニットが動作するため，1 サイクル分性能が低下する．

(c) 起動ヒット　F ステージにおいて予測表に基づいて上流クロックを投入する．D ステージにおいて予測の検証を行うものの，予測通り演算ユニットを使用することが判明し，上流クロックを変化させない．遅れて動作する下流クロックにより，E ステージの開始と同時に演算ユニットが動作するため，性能は低下しない．

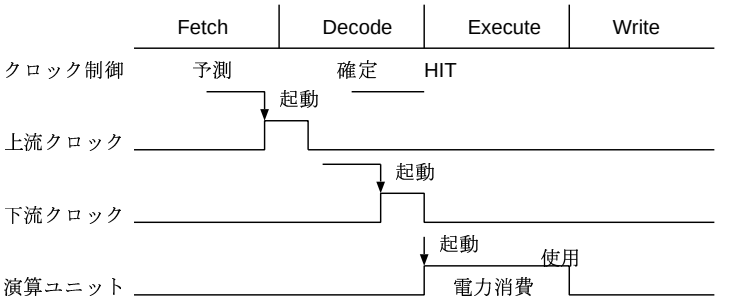
(d) 起動ミス　F ステージにおいて上流クロックを投入したものの，D ステージにおいて演算ユニットを使用しないことが判明し，上流クロックを停止する．遅れて下流クロックが停止され，E ステージの開始から 1 サイクル



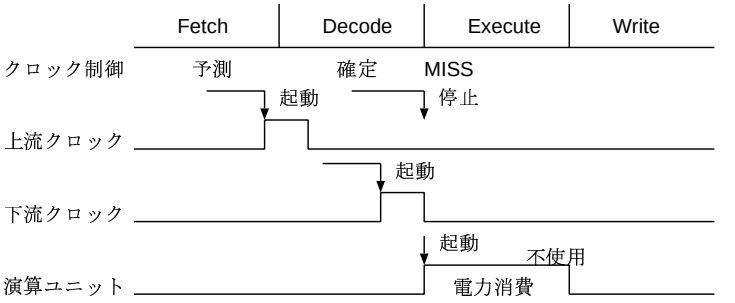
(a)待機ヒット



(b)待機ミス



(c)起動ヒット



(d)起動ミス

図 3.4: 起動ペナルティ=1 の場合

遅れて演算ユニットが停止するため，1 サイクル分だけ無駄に電力が消費される．

演算ユニットの使用予測が 100 %的中した場合，性能の低下を防ぎ，電力の増加を抑えることができる．しかし，完璧な予測をするためには大規模な予測機構が必要となり，かえって消費電力が増加してしまう．また,(b）を防ぐためにより多くの演算ユニットを起動しておくことは,(d）の増加に繋がり，消費電力が増加する．逆に,(d）を防ぐためになるべく演算ユニットを起動しないことは,(b）の増加に繋がり，性能が低下する．本論文では後者の考え方を基本方針とし，1 命令につき演算ユニットを 1 つ（メソッド関連の命令の場合は 2 つ）だけ起動するものとする．この方針では，予測が外れた場合，起動ミスおよび待機ミスの両方が起こることになる．

以下では，いかに小さな予測ハードウェア機構により,(b）および（d）を削減できるかについて，議論を進める．

3.4 演算ユニットの使用予測方法

起動ペナルティ=1 の場合の演算ユニット予測方法について述べる．この場合，予測を行う時点は命令デコード完了の 1 サイクル前，つまり命令フェッチ完了時である．この時点について用いることのできる予測に使える情報としては，以下に示すものが考えられる．

(a) F ステージ開始時のプログラムカウンタ（PC）の下位 n ビット．
(b) 直前の命令のオペコード．
(c) 直前の命令のオペコードをグループ化したもの．
(d) （a）と（c）の組み合わせ．
(e) （c）と（c）の組み合わせ（直前および 2 命令前のオペコード）．

（a）は 2^n 以下の命令数で構成されるループ構造に対して高い効果があると予測できるものの，ループ構造がないメソッドまたは 2^n を大幅に上回るバイト数の命令により構成されるループ構造に対しては効果が小さいと考えられる.(b）は連続する 2 命令間に強い相関関係がある場合に有効である．Java 仮想マシンはスタックマシンであることから，一連のスタック操作を繰り返すバイトコードは一般の RISC 命令よりもオペコードの相関が強いと考えられる．また，同じオペコードの命令が少なく命令数が 256 以下のループ構造をもつメソッドにも有効である.(c）は（b）よりもハードウェアを削減する試みである．グルー

プ化の方法としては，使用する演算ユニットによる分類，命令の機能・操作内容による分類などが考えられる．適当なグループ分けを行うことにより，予測のヒット率を（b）と同程度のままハードウェアを削減できる可能性がある.(d）は（a）と（b）の相乗効果を狙ったものである.(e）は連続する 3 命令間の相関関係の方が 2 命令間の相関関係より強いことから，相関関係の強化による予測ヒット率の向上がグループ化による予測ヒット率の低下に勝るならば，有効であると考えられる．

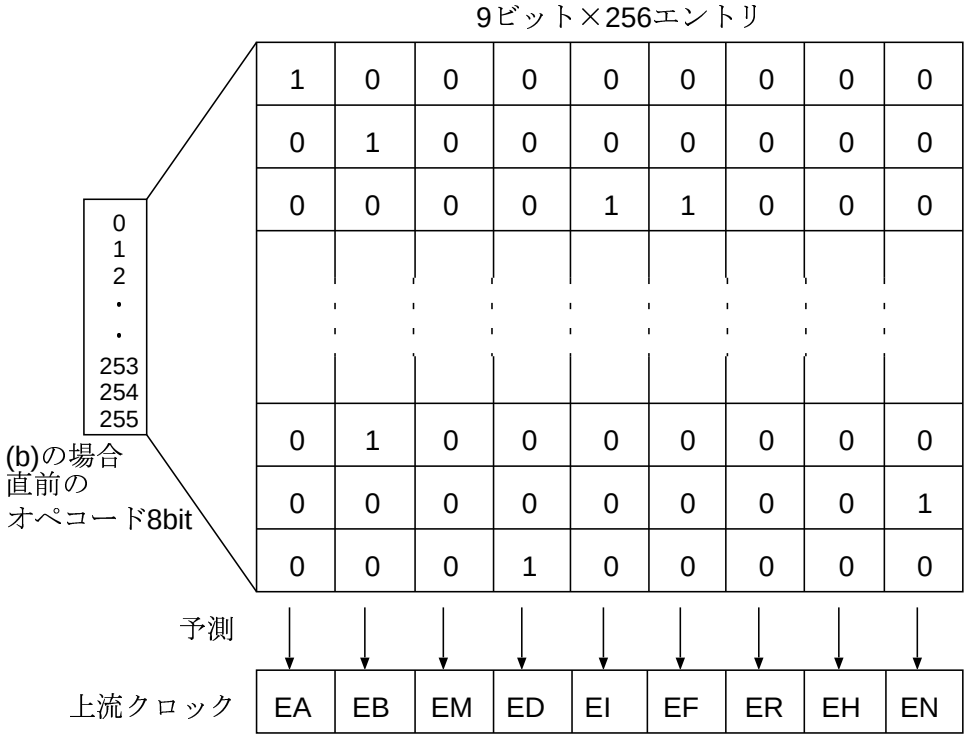


図 3.5: 予測用テーブル（（b）の場合）

図 3.5は（b）の場合の予測用テーブルの例である.(b）の場合，オペコードは 0 から 255 まであり，演算ユニットは 9 種類あるため，テーブルの行数は 256，列数は 9 である．各行は各オペコードに対応しており，2 列目以降には以前対応するオペコードの命令が使われた際の，演算ユニットの使用/不使用の情報が格納される．使用するユニットは 1 で表現され，使用しないユニットは 0 で表現される．命令が実行された際，F ステージにおいてオペコードをインデックスとして表を参照し，相当する行において 1 が格納されているユニットの上流クロックを投入し，0 が格納されているユニットの上流クロックを停止する．D

ステージにおいて予測ミスが判明した場合は，命令のデコード結果に基づいて相当する行を更新する．予測機構は図3.5のようなテーブルをハードウェアにより構成し，さらにテーブルに基づいてクロックを制御する機構を導入することにより行う．一般的に，n ビットの情報を用いて予測を行う場合，テーブルの行数を 2^n，インデックスをその情報とする．また，異なった2つの情報の組み合わせを用いる場合（(d),(e)）は，2次元テーブルになる．

さらに，起動ペナルティ=2 の場合も同様の方法によって予測を行う．この場合は,（a）は1命令前のPCになるため，2^n以内のループであってもその中に条件分岐命令を含むならば，完璧な予測ができなくなる.（b）は2命令前のオペコードになるため，相関関係が弱くなる.（c）および（e）についても，同様に相関関係が弱くなると推測できる.（d）は条件分岐による撹乱も発生し，相関関係も弱くなると推測できる．

第4章　本方式の定量的評価

4.1　測定方法

評価にはJava仮想マシンKaffe OpenVM 1.0 Beta 4[2]およびベンチマークプログラムSPEC JVM98 VERSION1.03[3]を用いた．Kaffeに仮定したハードウェア構成および予測機構を実装することにより，Javaプロセッサの動作をシミュレートし，サイクル数および消費電力率の測定を行った．

SPEC JVM98はJVMを搭載したマシンの性能を測定するベンチマークプログラムである．8種類のベンチマークテストから構成されている．

_201_compress　広く普及しているLZWファイル圧縮ユーティリティのJava版.

_202_jess　NASAのCLIPSで一般に使われるパズルの集合を解く.

_209_db　IBMが開発した住所録データベースに対するデータベース管理プログラム.

_213_javac　Sun MicrosystemsのJDK1.0.2に準拠するJavaコンパイラ.

_222_mpegaudio　MPEG Layer-3音声ファイルをデコードする.

_227_mtrt　レイ・トレーシング・ソフト.

_228_jack　Sun Microsystemsからライセンスを受けたパーサ・プログラム.

但し，Kaffe1.0b4上では_228_jackが動作しなかったため，評価の対象から外

した．

また，このベンチマークには，s1，s10，s100 の 3 段階の問題サイズを指定することができる．s10 はおよそ 10 分の 1 の実行時間を必要として，テストと調査目的のため，また，s1 は，JVM 上のベンチマークプログラムの動作確認のために提供されている．本論文では，特に記載のない限り s100 で測定を行う．

4.2 測定結果

4.2.1 起動ペナルティ=0 の場合

表 4.1: 演算ユニット使用率（上段%）および消費電力比（下段%）

	EA	EB	EM	ED	EI	EF	EH	ER	EN	全体
compress	15.7	2.2	0.0	2.5	10.9	10.9	0.0	44.9	0.0	**76.1%**
	1.6	0.2	0.0	0.1	1.1	2.4	0.0	5.4	0.0	**10.7%**
jess	12.4	3.6	0.4	0.2	33.4	36.4	0.0	35.0	1.2	**89.5%**
	1.2	0.4	0.1	0.0	3.3	8.0	0.0	4.2	0.1	**17.4%**
db	13.4	2.6	0.0	0.4	16.5	20.3	0.0	37.0	0.4	**74.1%**
	1.3	0.3	0.0	0.0	1.6	4.5	0.0	4.4	0.0	**12.2%**
javac	13.0	5.0	0.5	0.8	35.6	36.8	0.0	34.7	0.8	**91.5%**
	1.3	0.5	0.1	0.0	3.6	8.1	0.0	4.2	0.1	**17.8%**
mpegaudio	16.5	1.3	0.1	0.3	6.1	12.7	0.0	43.8	0.0	**74.7%**
	1.7	0.1	0.0	0.0	0.6	2.8	0.0	5.3	0.0	**10.5%**
mtrt	8.0	0.8	0.0	0.1	49.5	52.9	0.0	21.7	0.6	**84.0%**
	0.8	0.1	0.0	0.0	5.0	11.6	0.0	2.6	0.1	**20.1%**

起動ペナルティ=0 の場合の各演算ユニットの使用率および消費電力比を測定した．測定には，SPEC JVM98 を s100 により実行して得た命令トレースおよび，表 3.1に示した各演算ユニットの所要サイクル数と消費電力比率を用いた．

表 4.1に起動ペナルティ=0 の場合の各演算ユニットの使用率（上段%）および電力比（下段%）を示す．使用率は， 全体のサイクル数のうち各演算ユニットが動作したサイクル数の比である．電力比は， 使用率に表 3.1において仮定した消費電力比率を乗じて得たものである．右端列の上段は，全体の所要サイ

クル数のうち，少なくとも 1 つの演算ユニットが動作した割合を示している．演算ユニットが 1 つも動作していないサイクルは，前述したオブジェクト TLB ミスまたはヒープキャッシュミスが発生している期間に対応する．右端列の下段は，電力比の合計，すなわち，クロックを常時投入した場合の演算ユニット全体の消費電力に対する，本方式における消費電力の比率を表しており，起動ペナルティ=1，2 の場合において予測が全て的中した場合の消費電力の下限値に相当する．未使用の演算ユニットにおける電力をカットすることにより，約 80%から 90%の電力を削減できることがわかる．

表 4.1から EI および EF ユニットにおける使用率が電力比に大きな影響を与えていることがわかる．使用率が 33%を越える jess，javac，mtrt は全体の電力比が約 20%弱に達するが，使用率が 17%未満の compress，db，mpegaudio は電力比が 10%強に抑えられている．EI および EF ユニットを両方使う命令はメソッド呼び出しおよびリターンである．つまり，メソッド関連の命令が多数出現する場合，消費電力は倍増することを示している．これは，表 3.1に示した通り，メソッド関連の命令は 18 サイクルを要する上，EI および EF ユニットの単位時間あたりの消費電力は合計で 10+22=32%を占めるためであると考えられる．

4.2.2　起動ペナルティ>0 の場合

本研究の目標は，起動ペナルティ>0 の場合について，起動ペナルティ=0 の場合の消費電力（電力下限値）に近く，かつ，起動ペナルティによる性能低下が小さくなるような，極力小さい使用演算ユニット予測機構を提案することである．測定は，第 3 章において述べた予測方法を次のように具体化して Kaffe に予測用テーブルを実装し，起動ペナルティ=0 の場合と同様の方法により行った．

（1）予測なし（NP） 予測を行わず，必要になるまでクロックを止める方法．消費電力の下限（最良）値および所要サイクル数の上限（最悪）値が得られる．

（2）PC の下位 8bit（PC8） 「起動ペナルティ−1」だけ手前の PC の下位 8 ビットを用いる．

（3）オペコード 8bit（OP8） 「起動ペナルティ」だけ手前の命令のオペコード 8 ビットを用いる．ただし WIDE 命令は引き続くオペコードを使用する．

（4）PC の下位 4bit+グループ 4bit（PG8） 「起動ペナルティ−1」だけ手前の PC の下位 4 ビットと「起動ペナルティ」だけ手前の命令のグループ 4 ビットを用いる（(7) 参照）．

（5）グループ 4bit+グループ 4bit（WG8） 「起動ペナルティ」および「起動ペナルティ−1」だけ手前の命令のグループ 4 ビットを用いる（(7) 参照）.

（6）PC の下位 4bit（PC4） 「起動ペナルティ−1」だけ手前の PC の下位 4 ビットを用いる.

（7）グループ 4bit（G4） 表 4.2に示すように，命令の機能を基準にオペコードを 16 個のグループに分類し,「起動ペナルティ」だけ手前の命令のグループ 4 ビットを用いる. 但し，略号は，ALU=add,sub,mul,div,rem，SFT=and,or,xor,shl,shr,ushr，Xは任意の文字列を表す.

図 4.1に起動ペナルティ=1 の場合，図 4.2に起動ペナルティ=2 の場合の測定結果を示す. 各図の上段のグラフは，起動ペナルティ=0 の消費電力を 1 とした

表 4.2: 命令の分類方法（グループ）

	機能	命令
0	定数をプッシュ	Xconst_X,Xipush
1	ローカル変数をプッシュ	Xload,Xload_X
2	ローカル変数にストア	Xstore,Xstore_X
3	ローカル変数をインクリメント	iinc
4	int 型を比較・演算	ifX,iALU,iSFT,nop
5	long 型を比較・演算	lALU,lSFT,lcmp
6	float 型を比較・演算	fALU,fSFT,fcmp
7	double 型を比較・演算	dALU,dSFT,dcmp
8	reference 型を比較・演算	ifXnull,if_acmpX
9	型変換	X2X
10	配列から読み込み	Xaload,arraylength
11	配列へ書き込み	Xastore,
12	スタック操作	popX,dupX,dupX_X,swap
13	無条件分岐, サブルーチン	gotoX,jsrX,ret
14	テーブルジャンプ	Xswitch
15	マルチフロー, フレーム生成	ldcX,invokeX,Xreturn,athrow, Xstatic,Xfield,new,Xnewarray, checkcast,instanceof,monitor

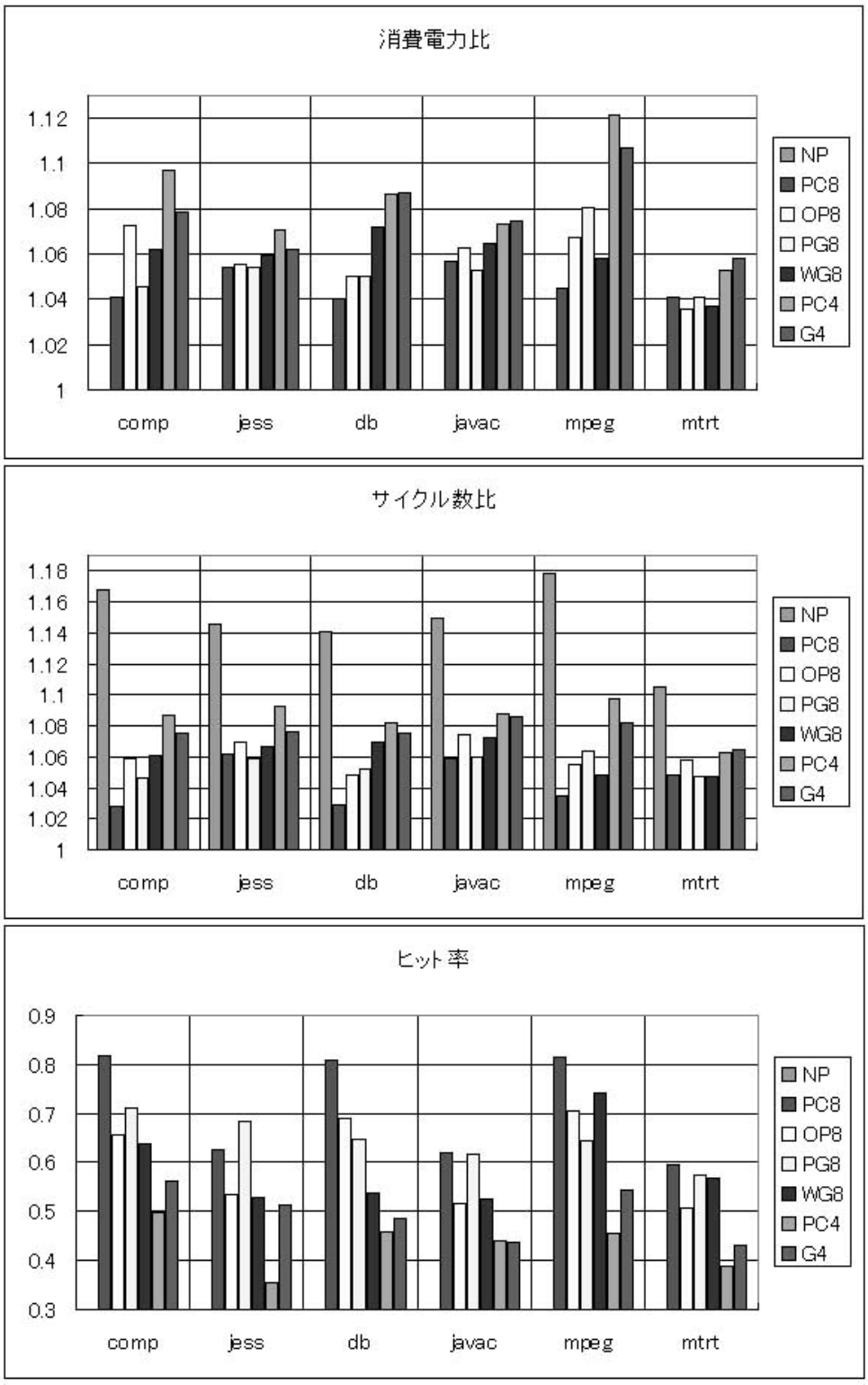


図 4.1: 起動ペナルティ=1 の場合

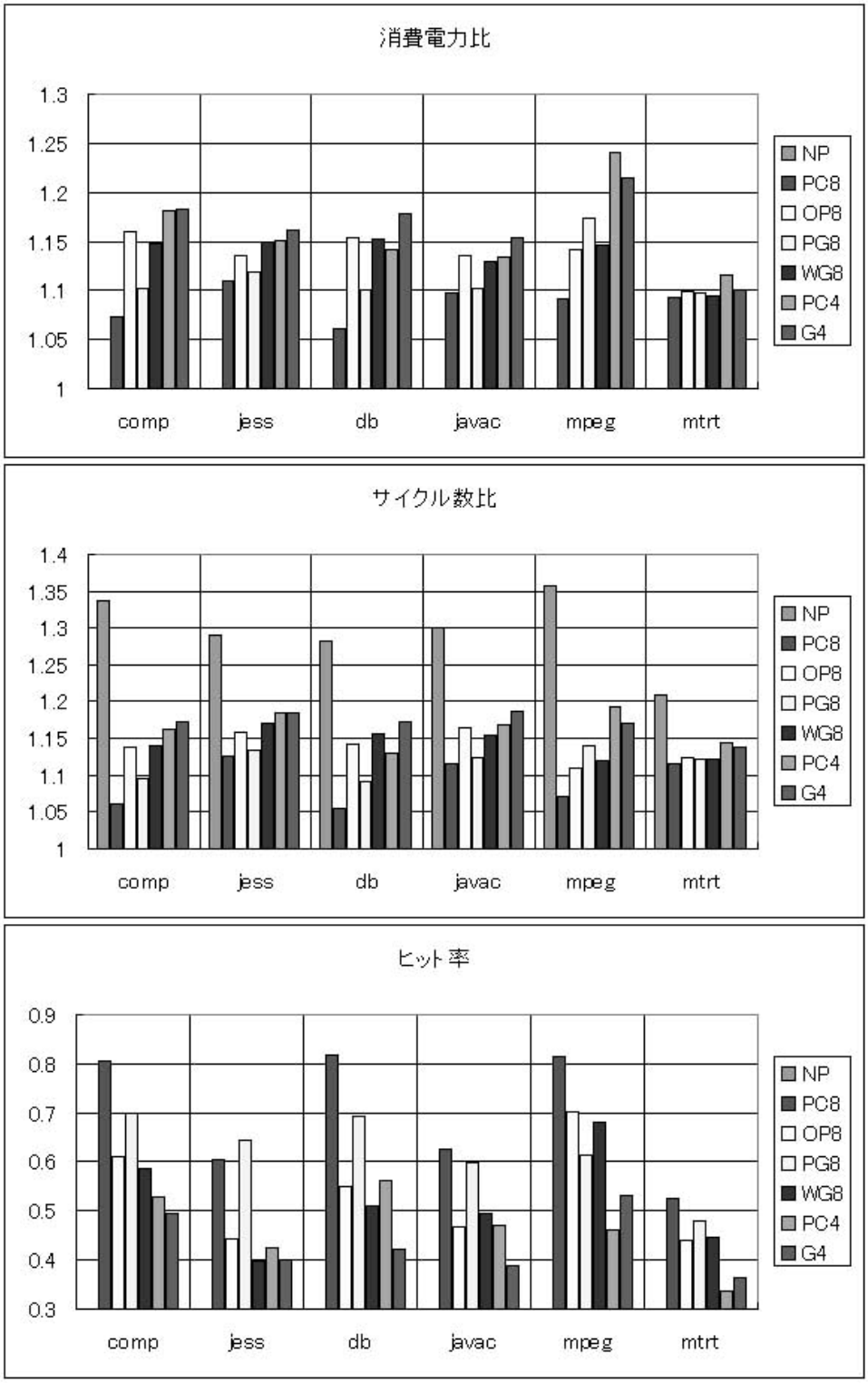


図 4.2: 起動ペナルティ=2 の場合

場合の，各予測方法における消費電力である．中段のグラフは，同様に起動ペナルティ=0 の所要サイクル数を 1 とした場合の，各所要サイクル数である．下段のグラフは，各予測方法によるヒット率，すなわち，起動ミスも待機ミスも起きない確率である．

予測を行わず必要になったらクロックを起動する NP では，消費電力率は 1，サイクル数比率は起動ペナルティ=1 の場合は 10%から 18%程度の増加，起動ペナルティ=2 の場合は 21%から 36%程度の増加である．これは，電力の下限値とサイクル数比率の上限値を示している．

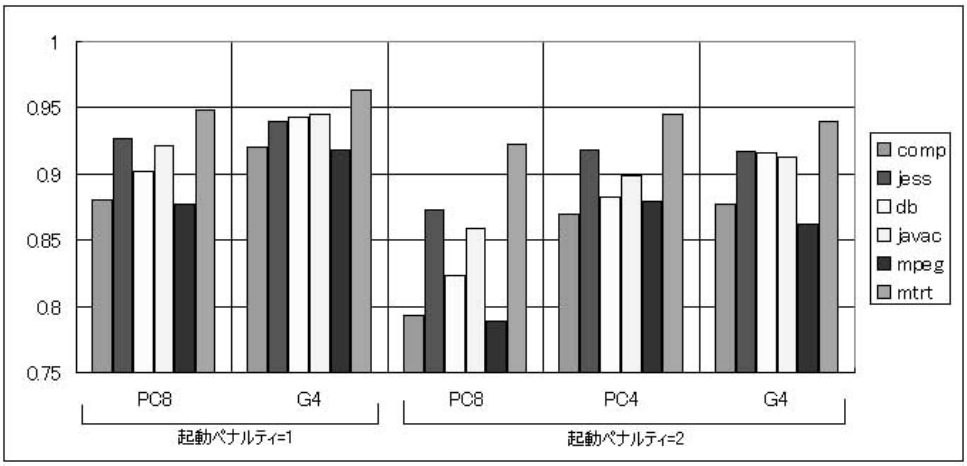


図 4.3: NP を 1 とした場合のサイクル数比率

8 ビットの情報を用いる方法の中では，起動ペナルティ=1 の場合，PC8 のヒット率が 60%から 81%程度と高く，消費電力率が 4%から 5%程度，サイクル数比率が 3%から 6%程度の増加に留まっている．起動ペナルティ=2 の場合も，PC8 のヒット率が 50%から 81%程度と高く，消費電力率が 6%から 11%程度，サイクル数比率が 5%から 11%程度の増加に留まっている．一部，PG8 と同程度のものがあるものの，起動ペナルティ=1，2 ともにほぼ全てのベンチマークプログラムにおいて PC8 による予測が最もよい効果を示している．また，図 4.3に示す通り，NP に対するサイクル数比率では，起動ペナルティ=1 の場合約 10%，起動ペナルティ=2 の場合約 15%の減少となっており，予測による性能向上が電力増加を上回っていることがわかる．

4 ビットの情報を用いる方法の中では，起動ペナルティ=1 の場合，G4 のヒット率が 43%から 56%程度となっており PC4 よりも高いものの，消費電力率，サ

イクル数比率では逆転しているベンチマークプログラムも存在する．逆転しているとはいえほぼ同程度であるため，起動ペナルティ=1 の場合は G4 が最も効果的と言える．消費電力率，サイクル数比率はともに 8%程度増加している．起動ペナルティ=2 の場合，mpegaudio と mtrt では G4 のヒット率が高く，これ以外では PC4 のヒット率が高いものの，概して 50%未満である．消費電力率，サイクル数比率はともに 15%程度増加している．また，図 4.3に示す通り，NP に対するサイクル数比率では，起動ペナルティ=1 の場合約 6%，起動ペナルティ=2 の場合約 10%の減少となっており，予測による性能向上が電力増加を下回ってしまうことがわかる．

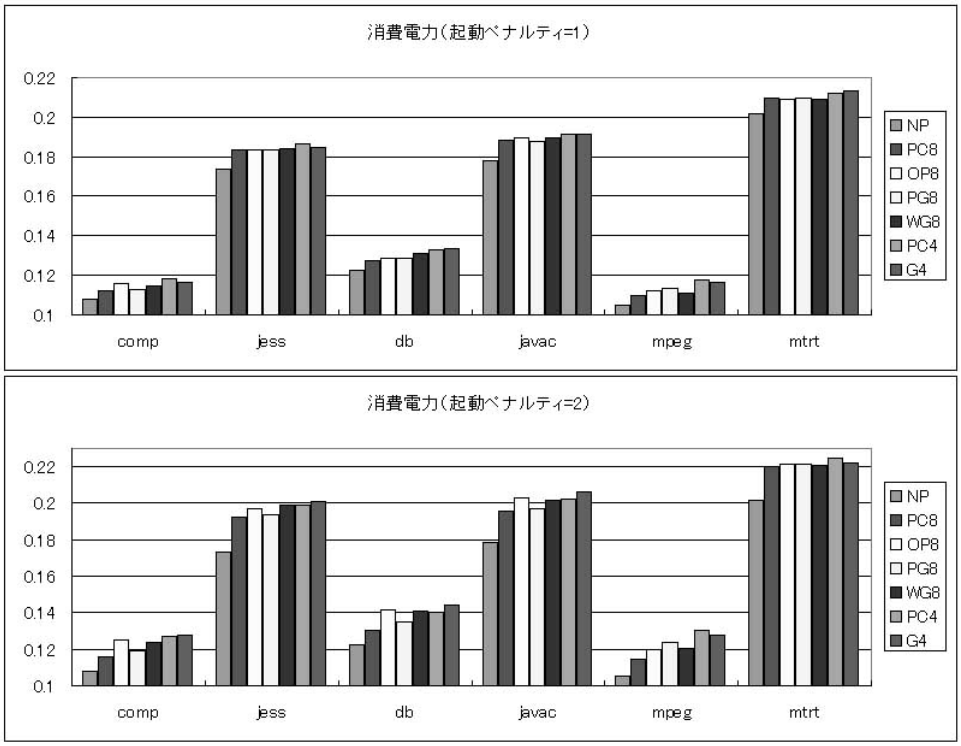


図 4.4: クロックを止めない方式を 1 とした場合の消費電力

最後に，実際どれだけの消費電力を削減することができるかについて，図 4.4 にクロックを止めない場合の消費電力に対する，起動ペナルティ=1，2 の場合の消費電力を示す．起動ペナルティ=1 の場合，PC8 では平均 15．5%，G4 では平均 15．9%まで削減でき，起動ペナルティ=2 の場合，PC8 では平均 16．1%，G4 では平均 17．1%まで削減できることがわかる．前述の通り，演算ユニット

の全体の消費電力は，プロセッサ全体の 80%を占めるため，プロセッサ全体に対しては 20%程度まで削減できることが明らかになった．

4.3 考察

測定の結果，8 ビットまたは 4 ビットの情報を用いて次命令のオペコードを予測するためには，次命令の PC の下位 8 ビットまたは現命令のグループ分け 4 ビットを用いる方法が，それぞれ高い効果を示すことが明らかになった．加えて，いくつかの特徴および問題も明らかになった．

まず，PC を用いた予測とオペコードのみを用いた予測を比べると，起動ペナルティ=2 のときの方がヒット率の差が大きくなる．これは，命令間の相関関係が弱くなることが，分岐命令による撹乱よりもヒット率の低下に大きな影響を与えることを示している．

ヒット率の差が消費電力率，サイクル数比率の差に現れる影響がベンチマークプログラムによって異なる．これは，各命令の出現頻度に起因している．2 サイクル以上を要する命令が少ないほど，ヒット率の差は消費電力率，サイクル数比率に大きく影響する．例えば，起動ペナルティ=1 の場合，18 サイクルを要する命令では予測ミスによる影響は 18 サイクルにつき 1 サイクルだが，1 サイクルの命令では 1 サイクルにつき 1 サイクルのペナルティが生じる．すなわち，各命令はその所要サイクル数によって 1 サイクルに対するペナルティに違いが生じることになる．メソッド関連の命令が 50%程度を占める mtrt の消費電力率，サイクル数比率が予測方法によってほとんど差がないのも，そのためと考えられる．mpegaudio の消費電力が他のプログラムに比べて際立って大きい．4.1から，mpegaudio における EI の使用率が他に比べて極めて低いこと，また，EI と EF の使用率の差分が 6%程度と，他に比べて極めて大きいことが読みとれる．3.1からわかるように，EI と EF の使用率差分とは，主に浮動小数点演算が占めるサイクル数の比率である．浮動小数点演算が多いことが，消費電力を押し上げている主な要因であると考えられる．

さらに，ベンチマークプログラムによってヒット率に差が生じている．本予測方式では，基本的にループ構造でのヒット率が非常に高いと考えられるので，これはループ構造の数に起因しているものと考えられる．ループ外でも予測を的中させるために命令間の相関関係による予測方法も考案したのだが，測定の結果，命令間に相関関係はあまり存在しないことがわかった．ループ外の予測

を的中させることができれば，ヒット率のばらつきは小さくなると推測される．

以上に示した測定とは別に，PC の最大値を測定したところ，mpegaudio で 21103，それ以外のプログラムで 5450 までの PC しか使用していないことがわかった．つまり，PC の下位 13 ビットを用いて予測をすれば，100%近いヒット率が得られることが期待される．しかし，実際に予測用テーブルのエントリ数を 2^{13} とし，PC の下位 13 ビットを用いて同様に測定したところ，ヒット率が PC8 より 1%から 8%程度と若干の向上しか見られず，ほぼ同様の効果しか得られないことがわかった．

多くのビット数を用いても効果が上がらないのは，各メソッドの先頭 PC が常に 0 であるという Java 特有の特徴によるものと考えられる．すなわち，別のメソッドが呼び出された場合，予測表が書き換えられてしまうため，一度呼び出されたメソッドであっても予測表にはそのメソッドの情報は残っていないことになる．これを回避するためには，PC に加えメソッド識別子を検索キーとする方法が考えられる．各メソッド毎に数ビットの PC を保持することができれば，一度呼び出されたメソッドは 100%の予測が可能となるはずであり．ループ内の予測に加え，前述のループ外の予測ヒット率をも飛躍的に上げることが期待される．各プログラムにおいて動作するメソッドの種類は，たかだか 300 程度であることから，メソッドの種類を 8 ビットにより表現し，これに PC の下位 8 ビットを加えた合計 16 ビットを用いることにより，ヒット率を格段に挙げられると推測できる．しかし，初めに述べたように，大規模なハードウェアを導入することは，低消費電力化の目的に反する．従って，合計 8 ビット程度に抑えて，メソッド識別子と PC の下位ビットの割合を調節することにより，どの組み合わせが最も効果的かを調べる必要がある．この方法の評価は，今後の課題である．

第 5 章　おわりに

本研究では，JeRTy をモデルとし，使用する演算ユニットのみを起動することにより消費電力を削減する方法の評価を行った．しかし，この方法では上流クロックが起動してから演算ユニットが使用可能になるまでのペナルティが存在し，性能が低下してしまう．そのため，本研究では，使用ユニットを予測する小さなハードウェア機構を導入することにより性能の低下を抑える方法の提

案および評価も行った．ペナルティが 1 または 2 サイクルである場合，上流クロックを投機的に投入/停止することにより，性能低下を抑えつつ電力を削減できることを示した．

256 エントリの予測表では，PC の下位 8 ビットを用いる予測方法が最も効果的であり，16 エントリの予測表では，同程度ではあったが，命令を機能別に分類したグループ 4 ビットを用いた予測が比較的効果であった．また，8 ビットの情報を用いる場合，予測による性能向上が消費電力増加を上回るものの，4 ビットでは逆転することがわかった．

最後に，本方式では主にループ構造内で予測を的中させていると考えられることから，ループ外で予測を的中させる方法の考案によりヒット率のばらつきの低減が期待される．PC の下位数ビットに加えてメソッド識別子を用いることにより，ループ内外のヒット率をより向上できる可能性がある．このような方法の評価は，今後の検討課題である．

謝辞

本研究の機会を与えてくださった富田眞治教授に深甚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った中島康彦助教授，森眞一郎助教授，五島正裕助手に心から感謝いたします．さらに，日頃からご助力頂いた京都大学大学院情報学研究科通信情報システム専攻冨田研究室の諸兄に心より感謝いたします．最後に，JeRTy に関する貴重な資料をご提供頂きましたオムロン株式会社の宮田佳昭氏に感謝いたします．

参考文献

[1] OMRON Corp.: JeRTy,
http://www.jerty.com (2001).

[2] Kaffe.org: Welcome to Kaffe,
http://www.kaffe.org (2001).

[3] Standard Performance Evaluation Corporation: Welcome to SPEC,
http://www.spec.org/ (2001).

[4] 富田眞治: コンピュータアーキテクチャI, 丸善 (1994).

[5] Tim Lindholm and Frank Yellin: The Java Virtual Machine Specification,

Addison-Wesley (1997).

[6] Jon Meyer and Tory Downing: Java Virtual Machine, O'REILLY (1997).

[7] D.Brooks, et al.: Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors, IEEE MICRO, Nov/Dec, pp.26–44 (2000).

[8] D.Brooks, et al.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, ISCA'00 (2000).

[9] N.Vijaykrishnan, et al.: Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower, ISCA'00 (2000).

[10] M.Koegst, et al.: Low Power Design of FSMs by State Assignment and Disabling Self-Loops, EUROMICRO'97 (1997).

[11] R.V.Joshi, et al.: Low Power 900 MHz Register File (8 Ports, 32 Words x 64 Bits) in 1.8V, 0.25 μ SOI Technology, VLSI Design'00 (2000).

[12] N.Raghavan, et al.: Automatic Insertion of Gated Clocks at Register Transfer Level, VLSI Design'99 (1998).

[13] L.Benini, et al.: System-Level Dynamic Power Management, VOLTA'99 (1999).

[14] W.Athas: Low-Power VLSI Techniques for Applications in Embedded Computing, VOLTA'99 (1999).

[15] T.D.Burd and R.W.Brodersen: Design issues for dynamic voltage scaling, ISLPED'00 (2000).

[16] A.G.M.Strollo, et al.: New clock-gating techniques for low-power flip-flops, ISLPED'00 (2000).

[17] T.Vinereanu and S.Lidholm: An improved pass transistor synthesis method for low power, high speed CMOS circuits, ISLPED'00 (2000).

[18] D.Garrett, et al.: Challenges in clockgating for a low power ASIC methodology, ISLPED'99 (1999).

[19] 井上弘士, 村上和影: 実行履歴に基づいた低電力命令キャッシュ向けタグ比較回数削減手法, 情報処理学会研究報告 (2000).