(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2006-134186
(P2006-134186A)
(43)公開日　平成18年5月25日(2006.5.25)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	9/38	(2006.01)	G06F	9/38	310X	5B013
G06F	9/30	(2006.01)	G06F	9/30	310Z	5B033

審査請求 未請求　請求項の数 8　OL　（全 38 頁）

(21)出願番号　特願2004-324348 (P2004-324348)
(22)出願日　平成16年11月8日 (2004.11.8)

(71)出願人　504132272
国立大学法人京都大学
京都府京都市左京区吉田本町３６番地１
(74)代理人　100080034
弁理士　原　謙三
(74)代理人　100113701
弁理士　木島　隆一
(74)代理人　100116241
弁理士　金子　一郎
(72)発明者　中島　康彦
京都府京都市左京区吉田本町　国立大学法人京都大学大学院経済学研究科内

Fターム(参考)　5B013 AA20
5B033 BE00

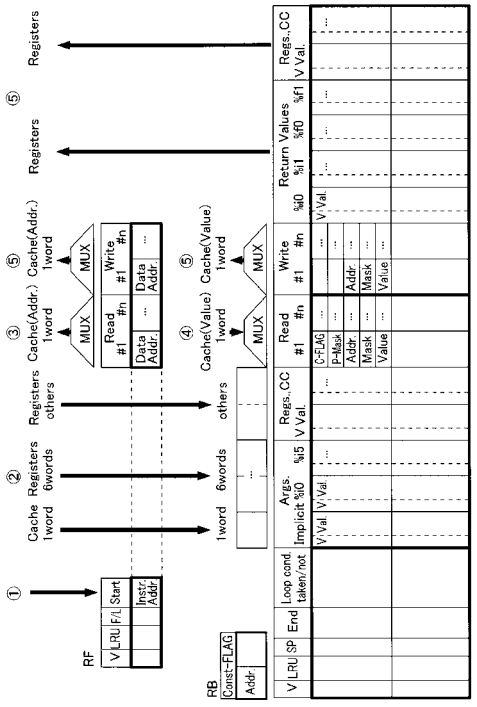
(54)【発明の名称】データ処理装置、データ処理プログラム、およびデータ処理プログラムを記録した記録媒体

(57)【要約】

【課題】　主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、予測の的中率を向上させることによって、より効果的な命令区間の事前実行を実現するデータ処理装置を提供する。

【解決手段】　ＲＢに、入力に用いられたレジスタの各アドレスに対して設けられる定数フラグ（Const-FLAG）、ならびに、入力要素のアドレスに対して設けられる変更フラグ（C-FLAG）および履歴マスク（P-Mask）を記憶する領域が設けられている。履歴マスクは、該アドレスのロード命令実行時に、該アドレスを生成したレジスタアドレスに上記定数フラグがセットされている場合にセットされる。予測処理部は、ＲＢに記憶されている入力要素のアドレスのうち、上記変更フラグがセットされ、かつ、履歴フラグがセットされているアドレスに関して、入力要素の変化の予測を行う。

【選択図】　図１



【特許請求の範囲】
【請求項１】
主記憶手段から命令区間を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、
上記主記憶手段から読み出した命令区間に基づく演算を行う第１の演算手段と、上記第１の演算手段による上記主記憶手段に対する読み出しおよび書き込み時に用いられるレジスタと、複数の命令区間の実行結果としての入力パターンおよび出力パターンを記憶する入出力記憶手段とを備え、
上記第１の演算手段が、命令区間を実行する際に、該命令区間の入力パターンと、上記入出力記憶手段に記憶されている入力パターンとが一致した場合、該入力パターンと対応して上記入出力記憶手段に記憶されている出力パターンをレジスタおよび／または主記憶手段に出力する再利用処理を行うとともに、
上記第１の演算手段による命令区間の実行結果を、上記入出力記憶手段に記憶する際に、入力パターンに含まれる入力要素のうち、予測を行うべき入力要素と予測を行う必要のない入力要素とを区別し、この区別情報を上記入出力記憶手段に登録する区別処理手段と、
上記区別情報に基づいて、上記入出力記憶手段に記憶されている入力要素のうち、予測を行うべき入力要素の値の変化の予測を行う予測処理手段と、
上記予測処理手段によって予測された入力要素に基づいて、該当する命令区間を事前実行する第２の演算手段とをさらに備え、
上記第２の演算手段による命令区間の事前実行結果が上記入出力記憶手段に記憶されることを特徴とするデータ処理装置。
【請求項２】
上記区別処理手段が、入力に用いられた上記レジスタの各アドレスに対して、スタックポインタまたはフレームポインタとして用いられる場合、および、該アドレスに対する書き込み命令が定数セット命令である場合に、該当アドレスに対して区別情報として定数フラグをセットし、上記以外の場合に、該当アドレスに対して上記定数フラグをリセットすることを特徴とする請求項１記載のデータ処理装置。
【請求項３】
上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として変更フラグをリセットし、上記入出力記憶手段に記憶された後に、該当アドレスに対してストア命令が実行された場合に、該当アドレスに対して変更フラグをセットすることを特徴とする請求項１または２記載のデータ処理装置。
【請求項４】
上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として履歴フラグをリセットし、該アドレスに対するロード命令実行時に、該アドレスを生成したレジスタアドレスに上記定数フラグがセットされている場合に、該アドレスに対して履歴フラグをセットすることを特徴とする請求項２記載のデータ処理装置。
【請求項５】
上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として変更フラグをリセットし、上記入出力記憶手段に記憶された後に、該当アドレスに対してストア命令が実行された場合に、該当アドレスに対して変更フラグをセットするとともに、
上記予測処理手段が、上記入出力記憶手段に記憶されている入力要素のアドレスのうち、上記変更フラグがセットされ、かつ、履歴フラグがセットされているアドレスに関して、入力要素の変化の予測を行うことを特徴とする請求項４記載のデータ処理装置。
【請求項６】
上記予測処理手段が、上記入出力記憶手段に記憶されている入力要素のうち、該入力要

素の履歴における値の変化量が０ではない入力要素のみに対して、入力要素の値の変化の予測を行うことを特徴とする請求項１または４記載のデータ処理装置。
【請求項７】
請求項１ないし６のいずれか一項に記載のデータ処理装置が備える各手段が行う処理をコンピュータに実行させることを特徴とするデータ処理プログラム。
【請求項８】
請求項７に記載のデータ処理プログラムを記録したコンピュータ読取り可能な記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置に関するものである。
【背景技術】
【０００２】
従来、ＣＰＵ(Central Processing Unit)を始めとするマイクロプロセッサにおいて、演算速度の高速化技術に関する研究開発が盛んに行われている。高速化技術としては、例えばパイプライン、スーパースケーラ、アウトオブオーダー実行、および、レジスタリネーミングなどが挙げられる。
【０００３】
パイプラインは、命令の実行処理を数段階に分解し、複数の命令を流れ作業的に同時処理を行う技術である。スーパースケーラは、命令の実行回路を２組以上用意し、複数の命令を同時に並行して実行する技術である。アウトオブオーダー実行は、命令の記述順序を無視して、いくつかの連続する命令の中から先に実行可能なものを探して先行処理を行う技術である。レジスタリネーミングは、例えばＣＩＳＣ(Complex Instruction Set Computer)タイプのプロセッサにおいて、従来のプロセッサにおける命令の互換性を保ちながら、汎用レジスタの数を増やすことによって並行処理が行われる確率を増大させる技術である。
【０００４】
このように、マイクロプロセッサにおける演算速度の高速化を図る際には、命令の実行を並行して行うことが重要となっている。しかしながら、プログラム中には、ある命令の結果に応じて異なる命令が行われるような依存関係、言い換えれば分岐が含まれている場合がほとんどである。このような分岐が含まれている場合、並行処理によって先行して処理を行っていると、分岐の結果によって先行処理した内容が無駄になるという状況が発生することになり、演算速度の高速化の効果が小さくなるという問題がある。
【０００５】
そこで、プログラム中に分岐がある場合に、分岐先を予測することによって先行処理が無駄になる確率を低減し、並行処理の効果を向上させる技術、いわゆる分岐予測に関する研究が数多く行われている。
【０００６】
しかしながら、分岐予測に基づいて投機的先行処理を行う場合には、一般的に次のような問題がある。第１の問題としては、予測の正当性を常に検証する必要があるので、先行命令列の実行時間そのものを削減することはできない、という点である。第２の問題としては、誤った予測に基づく一連の先行演算結果を全て無効化する必要があるので、一度に投機的先行処理できる命令数を多くするには、相応のハードウェアコストを要する、という点である。第３の問題としては、命令間の依存関係が多いほど、多重に投機的先行処理をする必要が生じ、予測の正当性の検証処理、および誤った予測に基づく処理の無効化処理が極めて複雑になる、という点である。
【０００７】
一方、分岐予測とは異なる高速化技術として、値再利用という技術も提案されている。

この値再利用とは、プログラムの一部分に関する入力値および出力値を再利用表に登録しておき、同じ箇所を再度実行する際に、入力値が再利用表に登録されているものである場合には、登録されている出力値を出力する、という技術である。この値再利用による効果としては次のようなものが挙げられる。（１）入力値が、再利用表に登録されている入力値と一致すれば、実行結果を検証する必要がない。（２）入力値および出力値の総数によってのみハードウェアコストが決定され、省略可能な命令列の長さが制約されない。（３）命令間の依存関係の多少は、再利用機構の複雑さに影響を与えない。（４）冗長なロード／ストア命令を削減することができるとともに、これに伴う消費電力の削減も実現される。

【０００８】

後記する非特許文献１には、プログラムにおける関数に関して値再利用を行う技術が示されている。この従来技術では、一般的にロードモジュールがＡＢＩ(Application Binary Interface)に従って作られることを利用しており、特に、ＳＰＡＲＣ(Scalable Processor ARChitecture) ＡＢＩを利用している。そして、このＡＢＩにおいて関数の入出力を特定することによって値再利用を実現している。すなわち、値再利用のためのコンパイラによる専用命令の埋め込みが不要となっており、既存ロードモジュールへの適用が可能となっている。

【０００９】

また、関数の多重構造を動的に把握することにより、関数内局所レジスタやスタック上の局所変数を値再利用における入出力値から除外するようにしており、これによって効率を向上させている。特に関数については、関数の複雑さに拘わらず、最大６のレジスタ入力、最大４のレジスタ出力、および、局所変数を含まない最小限の主記憶値の登録による再利用および事前実行が可能となっている。この従来技術について以下に詳細に説明する。

【００１０】

まず、単一の関数を対象として、何が入力で何が出力であるかを明らかにし、１レベルの再利用を行うために必要な機構について説明する。プログラムにおいては、一般的に関数は多重構造を形成している。関数Ａ（Function-A）が関数Ｂ（Function-B）を呼び出す構造を図１７（ａ）に示す。

【００１１】

帯域変数（Globals）は、関数Ａの入出力（A_{in}／A_{out}）および関数Ｂの入出力（B_{in}／B_{out}）になりうるものである。関数Ａの局所変数（Locals-A）は、関数Ａの入出力ではないが、ポインタを通じてＢの入出力になりうるものである。また、関数Ａから関数Ｂへの引数（Args）は、関数Ｂへの入力となりうるものであり、関数Ｂから関数Ａの返り値（Ret.Val.）は、関数Ｂからの出力となりうるものである。なお、関数Ｂの局所変数（Locals-B）は、関数Ａおよび関数Ｂの入出力には含まれない。

【００１２】

コンテクストに依存せずに関数Ｂを再利用するには、関数Ｂの実行時に、関数Ｂの入出力B_{in}／B_{out}のみを入出力として登録しなければならない。ここで、図１７（ａ）に示すプログラム構造を実行する際の主記憶におけるメモリマップを図１７（ｂ）に示す。このメモリマップにおいて、B_{in}／B_{out}を含まない領域はLocals-Bのみとなっている。よって、B_{in}／B_{out}を識別するには、GlobalsとLocals-Bとの境界、および、Locals-BとLocals-Aとの境界をそれぞれ確定しなければならない。前者については、一般的にＯＳ(Operating System)が実行時のデータサイズおよびスタックサイズの上限を決めることを利用し、ＯＳが設定する境界(LIMIT)に基づいてGlobalsとLocals-Bとの境界を確定することができる。後者については、Ｂが呼び出される直前のスタックポインタの値（SP in A）を用いることによって、Locals-BとLocals-Aとの境界を確定することができる。

【００１３】

次に、与えられた主記憶アドレスが、大域変数であるか、または、どの関数の局所変数

であるかを識別する方法について説明する。ロードモジュールは、ＳＰＡＲＣ　ＡＢＩに規定されている以下の条件を満たすと仮定する。なお、％fpはフレームポインタ、％spはスタックポインタを意味するものとする。
(1)％sp以上の領域のうち、％sp＋０～６３はレジスタ退避領域、％sp＋６８～９１は引数退避領域であり、いずれも関数の入出力ではない。
(2)構造体を返す場合の暗黙的引数(Implicit Arg.)は％sp＋６４～６７に格納される。
(3)明示的引数(Explicit Arg.)はレジスタ％o０～５、％sp＋９２以上の領域に置かれる。

【００１４】
まず、大域変数と局所変数とを区別するために、一般的に、ＯＳが実行時のデータサイズおよびスタックサイズの上限を決めることを利用し、次の事項を仮定する。
(1)大域変数はLIMIT未満の領域に置かれる。
(2)％spは、LIMIT以下になることはなく、LIMIT～％spの領域は無効である。

【００１５】
以上の条件を満たしながら、関数Ａが関数Ｂを呼び出す場合の、メモリマップにおける引数およびフレームの概要を図１８に示す。同図を参照しながら、以下にＡの局所変数およびＢの局所変数を区別する方法について説明する。

【００１６】
同図において、（ａ）はＡ実行中の状態を示している。LIMIT未満の太枠部分に命令(Instructions)および大域変数(Global Vars.)が格納され、％sp以上に有効な値が格納されている。％sp＋６４には、Ｂが構造体を返り値とする場合の暗黙的引数として、構造体の先頭アドレスが格納される。Ｂに対する明示的引数の先頭６ワードはレジスタ％o０～５、第７ワード以降は％sp＋９２以上に格納される。ベースレジスタを％spとするオペランド％sp＋９２が出現した場合、この領域は引数の第７ワードすなわちＢの局所変数である。一方、オペランド％sp＋９２が出現しない場合、この領域はＡの局所変数である。このように、（ａ）の状態では、オペランドを検証することによってＡの局所変数とＢ局所変数とを区別することができる。

【００１７】
一方、（ｂ）はＢ実行中の状態を示している。引数が入力、返り値が出力、大域変数およびＡの局所変数が入出力となりうる。ただし、Ｂは可変長引数を受け入れる場合があるので、一般に％fp＋９２以上の領域がＡの局所変数の領域となるかＢの局所変数の領域となるかは判断できない。

【００１８】
局所変数を区別するには、まず、（ａ）の時点において引数の第７ワード以降を検出した関数呼び出しは再利用の対象外とし、第７ワード以降を検出しない関数呼び出しに関して、直前に％sp＋９２の値を記録しておくようにする。なお、第７ワード以降を使用する関数呼び出しの出現頻度が低いと予想されることから、第７ワード以降を使用する関数を再利用の対象外とする制限による性能低下は軽微なものと考える。

【００１９】
以上の準備により、（ｂ）における主記憶参照アドレスが、予め記録した％sp＋９２の値以上の場合はＡの局所変数、小さい場合はＢの局所変数であることがわかる。Ｂ実行時には、Ｂの局所変数を除外しながら、大域変数およびＡの局所変数を再利用表へ登録する。

【００２０】
再利用の際は、Ｂの局所変数は入出力から除外されるので、Ｂの局所変数のアドレスが一致している必要がない。このため、いかなるコンテクストであっても、入力さえ一致すれば、再利用することが可能である。ただし、Ｂが参照する大域変数やＡの局所変数については、アドレスおよびデータの両方が再利用表の内容と完全に一致する必要がある。すなわち、Ｂを実行する前に、どのようにして比較すべき主記憶アドレスを網羅するかがポイントになる。

【００２１】
Ｂが参照する大域変数やＡの局所変数のアドレスは、そもそもＢにおいて生成されるアドレス定数や、大域変数／引数を起源とするポインタに基づいているものである。よって、まず引数が完全に一致する再利用表中のエントリを選択した後に、関連する主記憶アドレスをすべて参照して一致比較を行うことにより、Ｂが参照すべき主記憶アドレスを網羅することができる。そして、全ての入力が一致した場合にのみ、登録済の出力（返り値、大域変数、およびＡの局所変数）を再利用することができる。
【００２２】
関数再利用を実現するために、再利用表として、関数管理表（ＲＦ）および入出力記録表（ＲＢ）を設けることにする。１つの関数を再利用するために必要なハードウェア構成を図１９に示す。複数の関数を再利用可能とするには、この構成を複数組用意することになる。
【００２３】
この表において、ＲＦおよびＲＢに保持されるVは、エントリが有効であるか否かを示すフラグであり、LRU(least recently used)は、エントリ入れ替えのヒントを示している。ＲＦは、上記のVおよびLRUの他に、関数の先頭アドレス(Start)、および参照すべき主記憶アドレス(Read/Write)を保持する。ＲＢは、上記のVおよびLRUの他に、関数呼び出し直前の％sp(SP)、引数(Args.)（V：有効エントリ、Val.：値）、主記憶値(Mask：Read/Writeアドレスの有効バイト、Value：値)、および、返り値(Return Values)(V：有効エントリ、Val.：値)を保持する。
【００２４】
返り値は、％ｉ０～１（リーフ関数では％ｏ０～１に読み替える）または％ｆ０～１に格納され、％ｆ２～３を使用する返り値（拡張倍精度浮動小数点数）は対象プログラムには存在しないものと仮定する。ReadアドレスはＲＦが一括管理し、MaskおよびValueはＲＢが管理することにより、Readアドレスの内容とＲＢの複数エントリをＣＡＭ(content-addressable memory)により一度に比較する構成を可能としている。
【００２５】
単一の関数を再利用するには、まず、関数実行時に、局所変数を除外しながら、引数、返り値、大域変数および上位関数の局所変数に関する入出力情報を再利用表に登録していく。ここで、読み出しが先行した引数レジスタは関数の入出力として、また、返り値レジスタへの書き込みは関数の出力として登録する。その他のレジスタ参照は登録する必要がない。主記憶参照も同様に、読み出しが先行したアドレスについては入力、書き込みは出力として登録する。
【００２６】
関数から復帰するまでに次の関数を呼び出した場合、または、登録すべき入出力が再利用表の容量を超える、引数の第７ワードを検出する、途中でシステムコールや割り込みが発生する、などの擾乱が発生しなかった場合、復帰命令を実行した時点で、登録中の入出力表エントリを有効にする。
【００２７】
以降、図１９を参照しながら説明すると、関数を呼び出す前に、(1)関数先頭アドレスを検索し、(2)引数が完全に一致するエントリを選択し、(3)関連する主記憶アドレスすなわち少なくとも１つのMaskが有効であるReadアドレスをすべて参照して、(4)一致比較を行う。全ての入力が一致した場合に、(5)登録済の出力（返り値、大域変数、およびＡの局所変数）を書き戻すことによって、関数の実行を省略することができる。
【００２８】
ここで、命令区間の一例として、図２０に示す命令区間が、図１９に示したＲＦおよびＲＢの構成によって実行された場合の例について説明する。同図において、ＰＣは、該命令区間が開始された際のＰＣ値を示している。すなわち、命令区間の先頭が１０００番地となっている。また、図２１は、図２０に示す命令区間が実行された場合に、ＲＢに登録される入力アドレスおよび入力データ、並びに出力アドレスおよび出力データを簡略化し

て示しており、図２２は、ＲＢにおける実際の登録状況を示している。

【００２９】

第１行目の命令（以降、単に第１の命令のように称する）において、アドレス定数Ａ１がレジスタＲ０にセットされる。第２の命令において、レジスタＲ０の内容をアドレスとする主記憶からロードされた４バイトデータ（00110000）がレジスタＲ１に格納される。この場合、アドレスＡ１、マスク（FFFFFFFF）（マスクにおいて、Fが有効バイトを示しており、0が無効バイトを示す）、データ（00110000）は、入力としてＲＢにおけるInput側の第１列に登録され、レジスタ番号Ｒ１、マスク（FFFFFFFF）、およびデータ（00110000）は出力としてＲＢにおけるOutput側の第１列に登録される。

【００３０】

第３の命令において、アドレス定数Ａ２がレジスタＲ０にセットされる。第４の命令において、レジスタＲ０の内容をアドレスとする主記憶からロードされた１バイトデータ（02）がレジスタＲ２に格納される。この場合、アドレスＡ２、マスク（FF000000）、およびデータ（02）は入力としてＲＢにおけるInput側の第２列に登録される。この際、アドレスＡ２の残り３バイトについては、Don't Careを意味する「－」が格納される。レジスタ番号Ｒ２、マスク（FFFFFFFF）およびデータ（00000002）は出力としてＲＢにおけるOutput側の第２列に登録される。

【００３１】

第５の命令において、アドレス（Ａ２＋Ｒ２）からロードされた１バイトデータ（22）がレジスタＲ２に格納されている。アドレスＲ２の値は（02）であったので、アドレス（Ａ２＋02）、およびデータ（22）が、入力としてＲＢにおけるInput側の第２列に追加登録される。この際、アドレス（Ａ２＋02）の部分に登録が行われ、アドレス（Ａ２＋01）および（Ａ２＋03）に対応する部分は、Don't Careを意味する「－」のままとなる。すなわち、アドレスＡ２に対応するマスクは（FF00FF00）となる。レジスタ番号Ｒ２、マスク（FFFFFFFF）、およびデータ（00000022）は、出力としてＲＢにおけるOutput側の第２列に上書きされる。

【００３２】

第６の命令において、アドレス定数Ａ３がレジスタＲ０にセットされる。第７の命令において、レジスタＲ０の内容をアドレスとする主記憶からロードされた１バイトデータ（33）がレジスタＲ３に格納される。この場合、アドレスＡ３、マスク（00FF0000）、およびデータ（33）は入力としてＲＢにおけるInput側の第３列に登録される。レジスタ番号Ｒ３、マスク（FFFFFFFF）、およびデータ（00000033）は出力としてＲＢにおけるOutput側の第３列に登録される。

【００３３】

第８の命令において、アドレス（Ｒ１＋Ｒ２）からロードされた１バイトデータ（44）がレジスタＲ４に格納される。この場合、アドレスＲ１とアドレスＲ２は命令区間の内部にて上書きされたレジスタのアドレスとなるので、アドレスＲ１およびアドレスＲ２は命令区間の入力とはならない。一方、アドレス（Ｒ１＋Ｒ２）によって生成されたアドレスＡ４は命令区間の入力であるので、アドレスＡ４、マスク（00FF0000）、およびデータ（44）は入力としてＲＢにおけるInput側の第４列に登録される。レジスタ番号Ｒ４、マスク（FFFFFFFF）、およびデータ（00000044）は出力としてＲＢにおけるOutput側の第４列に登録される。

【００３４】

第９の命令において、レジスタＲ５から値が読み出され、読み出された値に１が加えられた結果が再びレジスタＲ５に格納される。この場合、レジスタＲ５、マスク（FFFFFFFF）、およびデータ（00000100）は入力としてＲＢにおけるInput側の第５列に登録される。また、レジスタ番号Ｒ５、マスク（FFFFFFFF）、およびデータ（00000101）は出力としてＲＢにおけるOutput側の第５列に登録される。

【００３５】

以上のように、命令実行時におけるメモリ／レジスタからの読み出しに際しては、以下

の処理が行われる。
（１）ＲＢにおけるOutput側が検索され、読み出されたアドレス／レジスタ番号が既登録であれば、該アドレス／レジスタ番号はInput側に登録されずに終了する。
（２）ＲＢにおけるOutput側になければＲＢにおけるInput側が検索され、読み出されたアドレス／レジスタ番号が既登録であれば該アドレス／レジスタ番号は登録されずに終了する。
（３）ＲＢにおけるInput側にもなければ、ＲＢに新たにエントリが追加されて、該アドレス／レジスタ番号および値が登録される。
【００３６】
また、命令実行時におけるメモリ／レジスタへの書き込みに際しては以下の処理が行われる。
（１）ＲＢにおけるOutput側が検索され、読み出されたアドレス／レジスタ番号が既登録であれば値が更新されて終了する。
（２）ＲＢにおけるOutput側になければ、新たにエントリが追加されて読み出されたアドレス／レジスタ番号および値が登録される。
【００３７】
また、後述する特許文献１では、上記のような再利用を行う構成において、プロセッサを複数設け、並列事前実行を行う構成が開示されている。この並列事前実行が行われる際の入力の予測方法として、最後に出現した引数および最近出現した２組の引数の差分に基づいて、ストライド予測を行う方法が開示されている。
【００３８】
以上のように入力予測を行えば、上記した入力パラメータが単調に変化し続けるような場合に、事前に予測しておいた結果に基づいて効果的に再利用を行うことが可能となる。
【非特許文献１】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，ＨＰＳ５，pp.1-12，Sep.(2002)，“関数値再利用および並列事前実行による高速化技術”（中島康彦、緒方勝也、正西申悟、五島正裕、森眞一郎、北村俊明、富田眞治）（発行日２００２年９月１５日）
【特許文献１】特開２００４－２５８９０５号公報（公開日２００４年９月１６日）
【発明の開示】
【発明が解決しようとする課題】
【００３９】
図２３は、図２０に示す命令区間が繰り返し実行された場合における、ＲＢの入力側に登録される履歴の例を示している。この例では、Timeが１～４まで変化するごとに命令区間が実行され、命令区間が実行される度に、アドレスＡ２の値は、（02）、（03）、（04）、（05）と変化しており、これに伴って他の入力要素における値が変化している。
【００４０】
また、各履歴の間に示されるdiffは、対応する入力要素の値の変化量を示している。上記した従来の入力予測は、このdiffを用いて予測を行うことになる。図２４は、この従来の入力予測による予測結果を示している。
【００４１】
例えばループ制御変数のように、単調変化するアドレス（上記の例ではアドレスＡ２に対応）の内容については正確に予測することができている。しかしながら、命令区間に配列要素が含まれている場合、配列要素の添字が単調変化していても、配列要素値は一般に単調変化するとは限らない。図２３に示す例では、アドレスＡ２からロードした値が配列要素の添字に該当しており、この添字をアドレスとして用いる主記憶参照はアドレスが変化するために、履歴として登録される入力要素の数そのものが変化することになる。このような状況では、同一列の変化に規則性がなくなるために、図２４におけるアドレスＡ３に対応する列に示すように、予測的中率が極めて悪化することになる。
【００４２】
入力予測を行う際に、内容が変化しないアドレスに関する値の予測をすることはハード

ウェア資源の無駄となる。また、値の変化に規則性がない場合は、差分を０と仮定して予測するしかないが、無理に予測することにより、かえって的中率を下げることがある。図２４に示す例では、Ａ２＋4に対応するアドレスについてはマスク位置そのものの変化を予測すべきであるが、マスク位置の変化まで予測することは困難である。この場合には、予測せずに直接主記憶値を参照することが得策であることがわかる。

【００４３】

以上の課題はいずれも、登録された全てのアドレスを一律に扱ったことにより生じた問題である。

【００４４】

本発明は上記の問題点を解決するためになされたもので、その目的は、主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、予測の的中率を向上させることによって、より効果的な命令区間の事前実行を実現するデータ処理装置、データ処理プログラム、およびデータ処理プログラムを記録した記録媒体を提供することにある。

【課題を解決するための手段】

【００４５】

上記の課題を解決するために、本発明に係るデータ処理装置は、主記憶手段から命令区間を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、上記主記憶手段から読み出した命令区間に基づく演算を行う第１の演算手段と、上記第１の演算手段による上記主記憶手段に対する読み出しおよび書き込み時に用いられるレジスタと、複数の命令区間の実行結果としての入力パターンおよび出力パターンを記憶する入出力記憶手段とを備え、上記第１の演算手段が、命令区間を実行する際に、該命令区間の入力パターンと、上記入出力記憶手段に記憶されている入力パターンとが一致した場合、該入力パターンと対応して上記入出力記憶手段に記憶されている出力パターンをレジスタおよび／または主記憶手段に出力する再利用処理を行うとともに、上記第１の演算手段による命令区間の実行結果を、上記入出力記憶手段に記憶する際に、入力パターンに含まれる入力要素のうち、予測を行うべき入力要素と予測を行う必要のない入力要素とを区別し、この区別情報を上記入出力記憶手段に登録する区別処理手段と、上記区別情報に基づいて、上記入出力記憶手段に記憶されている入力要素のうち、予測を行うべき入力要素の値の変化の予測を行う予測処理手段と、上記予測処理手段によって予測された入力要素に基づいて、該当する命令区間を事前実行する第２の演算手段とをさらに備え、上記第２の演算手段による命令区間の事前実行結果が上記入出力記憶手段に記憶されることを特徴としている。

【００４６】

上記の構成では、入出力記憶手段に、複数の命令区間の実行結果としての入力パターンおよび出力パターンが記憶されており、命令区間の実行時に、該命令区間の入力パターンと、入出力記憶手段に記憶されている入力パターンとが一致した場合に再利用を行う構成となっている。そして、予測処理手段によって、入出力記憶手段に記憶されている入力要素の今後の変化が予測され、この予測結果に基づいて、第２の演算手段が命令区間の事前実行を行うようになっている。

【００４７】

ここで、前記した従来技術のように、単純に入力要素の予測を行うと、予測の的中率が低くなることによって、予測による事前実行の効果が非常に低くなるという問題がある。これに対して、上記の構成によれば、まず区別処理手段によって、入力パターンに含まれる入力要素のうち、予測を行うべき入力要素と予測を行う必要のない入力要素とが区別される。そして、予測処理手段は、区別処理手段によって予測を行うべき入力要素と判断された入力要素について予測を行うようになっている。したがって、予測の的中率を向上させることが可能となるので、より効果的な命令区間の事前実行を実現することが可能となる。このような事前実行が行われることによって、次に、同じ命令列が出現し、予測入力値と同じ入力が行われた場合には、命令列記憶手段に記憶されている値を再利用すること

が可能となる。

【００４８】

また、本発明に係るデータ処理装置は、上記の構成において、上記区別処理手段が、入力に用いられた上記レジスタの各アドレスに対して、スタックポインタまたはフレームポインタとして用いられる場合、および、該アドレスに対する書き込み命令が定数セット命令である場合に、該当アドレスに対して区別情報として定数フラグをセットし、上記以外の場合に、該当アドレスに対して上記定数フラグをリセットする構成としてもよい。

【００４９】

上記の構成によれば、入力に用いられたレジスタのアドレスのうち、アドレスが固定しており、かつ、値が単調変化すると予測されるアドレスに定数フラグをセットすることが可能となる。よって、定数フラグがセットされているレジスタのアドレスに基づく入力要素に対して予測を行うようにすることによって、予測的中率を向上させることが可能となる。

【００５０】

また、本発明に係るデータ処理装置は、上記の構成において、上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として変更フラグをリセットし、上記入出力記憶手段に記憶された後に、該当アドレスに対してストア命令が実行された場合に、該当アドレスに対して変更フラグをセットする構成としてもよい。

【００５１】

上記の構成によれば、入出力記憶手段に記憶されたものの、その後一度も書き込みが行われないアドレスに対しては、変更フラグがリセットされた状態となる。このようなアドレスに記憶されている内容は変化していないことになるので、該アドレスに対して予測を行う必要はないことになる。すなわち、上記のような変更フラグが入力要素のアドレスに設けられることによって、予測が必要なアドレスのみに対して予測を行うことが可能となる。よって、予測処理のためのハードウェア資源を有効に利用することが可能となる。

【００５２】

また、本発明に係るデータ処理装置は、上記の構成において、上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として履歴フラグをリセットし、該アドレスに対するロード命令実行時に、該アドレスを生成したレジスタアドレスに上記定数フラグがセットされている場合に、該アドレスに対して履歴フラグをセットする構成としてもよい。

【００５３】

上記の構成によれば、入出力記憶手段に記憶されている入力要素のアドレスに対するロード命令実行時に、該アドレスを生成したレジスタアドレスに上記定数フラグがセットされている場合に、該アドレスに対して履歴フラグがセットされるようになっている。ここで、定数フラグがセットされているレジスタアドレスとは、上記のように、アドレスが固定しており、かつ、値が単調変化すると予測されるアドレスとなっている。よって、このようなレジスタアドレスに基づいて生成されたアドレスに関して予測を行うことによる予測的中率は高くなることが予想される。すなわち、上記のような履歴フラグを設けることによって、予測すべきアドレスを適切に設定することが可能となる。

【００５４】

なお、履歴フラグとしては、各アドレスに文字通りのフラグをたてるようにしてもよいし、複数のバイトデータからなるアドレスのうち、履歴保存対象とするバイト位置を示すマスクといった形式で履歴フラグを実現するようにしてもよい。

【００５５】

また、本発明に係るデータ処理装置は、上記の構成において、上記区別処理手段が、入力要素が新規に上記入出力記憶手段に記憶される際に、該入力要素のアドレスに対して、区別情報として変更フラグをリセットし、上記入出力記憶手段に記憶された後に、該当アドレスに対してストア命令が実行された場合に、該当アドレスに対して変更フラグをセッ

トするとともに、上記予測処理手段が、上記入出力記憶手段に記憶されている入力要素のアドレスのうち、上記変更フラグがセットされ、かつ、履歴フラグがセットされているアドレスに関して、入力要素の変化の予測を行う構成としてもよい。

【００５６】

ここで、変更フラグがセットされているアドレスとは、上記したように、予測を行うことによる効果が期待できるアドレスとなる。また、履歴フラグがセットされているアドレスとは、上記したように、予測的中率が高いことが期待できるアドレスとなる。したがって、上記の構成によれば、予測を行うことによる効果が高いと予想されるアドレスに関してのみ予測が行われることになる。よって、予測処理のためのハードウェア資源を有効に利用することが可能となる。

【００５７】

また、本発明に係るデータ処理装置は、上記の構成において、上記予測処理手段が、上記入出力記憶手段に記憶されている入力要素のうち、該入力要素の履歴における値の変化量が０ではない入力要素のみに対して、入力要素の値の変化の予測を行う構成としてもよい。

【００５８】

上記の構成によれば、履歴における値の変化量が０ではない入力要素のみに対して、入力要素の値の変化の予測が行われることになる。ここで、履歴における値の変化量が０となっている入力要素とは、変化がないことが予想される入力要素であるので、該入力要素に対して予測を行う必要はないことになる。すなわち、上記の構成によれば、予測が必要なアドレスのみに対して予測を行うことが可能となる。よって、予測処理のためのハードウェア資源を有効に利用することが可能となる。

【発明の効果】

【００５９】

以上のように、本発明に係るデータ処理装置は、上記第１の演算手段による命令区間の実行結果を、上記入出力記憶手段に記憶する際に、入力パターンに含まれる入力要素のうち、予測を行うべき入力要素と予測を行う必要のない入力要素とを区別し、この区別情報を上記入出力記憶手段に登録する区別処理手段と、上記区別情報に基づいて、上記入出力記憶手段に記憶されている入力要素のうち、予測を行うべき入力要素の値の変化の予測を行う予測処理手段と、上記予測処理手段によって予測された入力要素に基づいて、該当する命令区間を事前実行する第２の演算手段とをさらに備え、上記第２の演算手段による命令区間の事前実行結果が上記入出力記憶手段に記憶される構成である。

【００６０】

これにより、予測の的中率を向上させることが可能となるので、より効果的な命令区間の事前実行を実現することが可能となる。このような事前実行が行われることによって、次に、同じ命令列が出現し、予測入力値と同じ入力が行われた場合には、命令列記憶手段に記憶されている値を再利用することが可能となるという効果を奏する。

【発明を実施するための最良の形態】

【００６１】

本発明の実施の一形態について図１ないし図１６に基づいて説明すれば、以下のとおりである。

【００６２】

（データ処理装置の構成）

本実施形態に係るデータ処理装置の概略構成を図２に示す。同図に示すように、該データ処理装置は、ＭＳＰ(Main Stream Processor)１Ａ、ＳＳＰ(Shadow Stream Processor)１Ｂ、再利用表としてのＲＦ／ＲＢ（命令列記憶手段）２、および主記憶（主記憶手段）を備えた構成となっており、主記憶３に記憶されているプログラムデータなどを読み出して各種演算処理を行い、演算結果を主記憶３に書き込む処理を行うものである。なお、同図に示す構成では、ＳＳＰ１Ｂを１つ備えた構成となっているが、２つ以上備えた構成となっていてもよい。

【００６３】
ＲＦ／ＲＢ２は、プログラムにおける関数およびループを再利用するためのデータを格納するメモリ手段であり、ＲＢ登録処理部（区別処理手段）２Ａおよび予測処理部（予測処理手段）２Ｂを備えた構成となっている。このＲＦ／ＲＢ２の詳細、ならびにＲＢ登録処理部２Ａおよび予測処理部２Ｂの詳細については後述する。
【００６４】
主記憶３は、ＭＳＰ１ＡおよびＳＳＰ１Ｂの作業領域としてのメモリであり、例えばＲＡＭ(Random Access Memory)などによって構成されるものである。例えばハードディスクなどの外部記憶手段からプログラムやデータなどが主記憶３に読み出され、ＭＳＰ１ＡおよびＳＳＰ１Ｂは、主記憶３に読み出されたデータに基づいて演算を行うことになる。
【００６５】
ＭＳＰ１Ａは、ＲＷ（再利用記憶手段）４Ａ、演算器（第１の演算手段）５Ａ、レジスタ６Ａ、およびＣａｃｈｅ７Ａを備えた構成となっている。また、ＳＳＰ１Ｂは、同様に、ＲＷ（再利用記憶手段）４Ｂ、演算器（第２の演算手段）５Ｂ、レジスタ６Ｂ、およびＣａｃｈｅ／Ｌｏｃａｌ７Ｂを備えた構成となっている。
【００６６】
ＲＷ４Ａ・４Ｂは、再利用ウィンドウであり、現在実行中かつ登録中であるＲＦおよびＲＢ（後述する）の各エントリをリング構造のスタックとして保持するものである。このＲＷ４Ａ・４Ｂは、実際のハードウェア構造としては、ＲＦ／ＲＢ２における特定のエントリをアクティブにする制御線の集合によって構成される。
【００６７】
演算器５Ａ・５Ｂは、レジスタ６Ａ・６Ｂに保持されているデータに基づいて演算処理を行うものであり、ＡＬＵ（arithmetic and logical unit）と呼ばれるものである。レジスタ６Ａ・６Ｂは、演算器５Ａ・５Ｂによって演算を行うためのデータを保持する記憶手段である。なお、本実施形態では、演算器５Ａ・５Ｂ、およびレジスタ６Ａ・６Ｂは、ＳＰＡＲＣアーキテクチャに準じたものとする。Ｃａｃｈｅ７Ａ・７Ｂは、主記憶３と、ＭＳＰ１ＡおよびＳＳＰ１Ｂとの間でのキャッシュメモリとして機能するものである。なお、ＳＳＰ１Ｂでは、Ｃａｃｈｅ７Ｂには、局所メモリとしてのＬｏｃａｌ７Ｂが含まれているものとする。
【００６８】
（ＲＦ／ＲＢの構成）
図１は、本実施形態におけるＲＦ／ＲＢ２によって実現される再利用表を示している。同図に示すように、ＲＦは、複数のエントリを格納しており、各エントリに対して、該エントリが有効であるか否かを示すV、エントリ入れ替えのヒントを示すLRU、関数の先頭アドレスを示すStart、参照すべき主記憶アドレスを示すRead/Write、および、関数とループとを区別するF/Lを保持している。
【００６９】
また、ＲＢは、ＲＦに格納されているエントリに対応して複数のエントリを格納しており、各エントリに対して、該エントリが有効であるか否かを示すV、エントリ入れ替えのヒントを示すLRU、関数またはループを呼び出す際の直前のスタックポイント％spを示すSP、引数(Args.)（V：有効エントリ、Val.：値）、主記憶値(C-FLAG：Readアドレスの変更フラグ、P-Mask：Readアドレスの履歴マスク、Mask：Read/Writeアドレスの有効バイト、Value：値)、返り値(Return Values)(V：有効エントリ、Val.：値)、ループの終了アドレス(End)、ループ終了時の分岐方向を示すtaken/not、および、引数や返り値以外のレジスタおよび条件コード(Regs.,CC)を保持している。また、ＲＢは、１つ以上のレジスタアドレスに対応して定数フラグ（Const-FLAG）を格納するメモリ領域を保持している。なお、定数フラグ（Const-FLAG）の詳細については後述する。
【００７０】
上記のＲＦおよびＲＢにおける各項目についてより詳細に説明する。上記Vは、上記のようにエントリが有効であるか否かを示すものであるが、具体的には、未登録時には「０

」、登録中である場合には「２」、登録済である場合には「１」の値が格納されるようになっている。例えば、ＲＦまたはＲＢを確保する際に、未登録エントリ（V=0）があれば、これを使用し、未登録エントリがなければ、登録済エントリ（V=1）の中からＬＲＵが最小のものを選択して上書きすることになる。登録中エントリ（V=2）は使用中であるので上書きすることはできない。

【００７１】

上記LRUは、一定時間間隔で右へシフトされていくシフトレジスタの中の「１」の個数を示したものである。ＲＦの場合、このシフトレジスタは、該当エントリに関して、再利用のための登録を行ったか、もしくは再利用を試みた場合に、左端に「１」が書き込まれるようになっている。したがって、該当エントリが頻繁に使用されれば、LRUは大きな値となり、一定期間使用されなければ、LRUの値は０となる。一方、ＲＢの場合、シフトレジスタには、該当エントリが再利用された場合に「１」が書き込まれるようになっている。したがって、該当エントリが頻繁に使用されれば、LRUは大きな値となり、一定期間使用されなければ、LRUの値は０となる。

【００７２】

上記ＲＢにおける主記憶値のMaskについて説明する。一般に、アドレスとデータとを１バイトずつ管理することにすれば管理が可能であるが、実際には、４バイト単位でデータを管理する方がキャッシュ参照を高速に行うことができる。そこで、ＲＦでは、主記憶アドレスを４の倍数で記憶するようになっている。一方、管理単位を４バイトとする場合、１バイト分だけをロードすることに対応できるようにするために、４バイトのうちでどのバイトが有効であるかを示す必要がある。すなわち、Maskは、４バイトのうちでどのバイトが有効であるかを示す４ビットのデータとなっている。例えば、C001番地から１バイト分をロードした結果、値がE8であった場合、ＲＦには、アドレスC000が登録され、ＲＢのMaskに「0100」、Valueに「00E80000」が登録されることになる。なお、Readアドレスにおける変更フラグ（C-FLAG）および履歴マスク（P-Mask）の詳細については後述する。

【００７３】

上記の引数や返り値以外のレジスタおよび条件コード(Regs.,CC)について説明する。本実施形態では、ＳＰＡＲＣアーキテクチャレジスタのうち、汎用レジスタ%g0-7、%o0-7、%l0-7、%i0-7、浮動小数点レジスタ%f0-31、条件コードレジスタICC、浮動小数点条件コードレジスタFCCを用いるようになっている（詳細は後述する）。このうち、リーフ関数の入力は汎用レジスタ%o0-5、出力は汎用レジスタ%o0-1、また、非リーフ関数の入力は汎用レジスタ%i0-5、出力は汎用レジスタ%i0-1、になり、入力は、arg[0-5]、出力は、rti[0-1]に登録される。ＳＰＡＲＣ－ＡＢＩの規定では、これら以外のレジスタは関数の入出力にはならないので、関数に関してはＲＢにおける引数(Args.)の項で十分である。

【００７４】

一方、ＳＰＡＲＣ－ＡＢＩの規定では、ループの入出力に関しては、用いられるレジスタの種類を特定することはできないので、ループの入出力を特定するには、全ての種類のレジスタに関してＲＢに登録する必要がある。よって、ＲＢにおけるRegs.,CCには、%g0-7、%o0-7、%l0-7、%i0-7、%f0-31、ICC、FCCが登録されるようになっている。

【００７５】

以上のように、ＲＦ／ＲＢ２において、ReadアドレスはＲＦが一括管理し、MaskおよびValueはＲＢが管理している。これにより、Readアドレスの内容とＲＢの複数エントリをＣＡＭ(content-addressable memory)によって一度に比較する構成を可能としている。このことについて、以下により詳しく説明する。

【００７６】

一般的に、アドレスが与えられると、そのアドレスに格納された値を参照することができるメモリは、ＲＡＭと呼ばれるメモリである。一方、上記のＣＡＭとは、連想メモリと呼ばれるメモリであり、検索すべき内容が与えられると、そのエントリに対応する信号がＯＮとなるように動作するようになっている。通常は、ＣＡＭはＲＡＭとセットにして用いられる。

【００７７】
ここで、ＣＡＭとＲＡＭとの連携動作について、具体例を挙げて説明する。ＣＡＭに、「５，５，５，５，５」、「１，３，１，１，１」、「１，３，３，５，２」、「６，６，６，６，６」というデータ列がエントリとして登録されており、ＲＡＭに、ＣＡＭにおける各データ列に対応して、「５，５」、「１，１」、「１，２」、「６，６」というデータが登録されているとする。ここで、検索すべきデータ列として、「１，３，３，５，２」をＣＡＭに入力すると、一致するエントリがＯＮとなり、ＲＡＭに登録されている該当するデータ「１，２」が出力されることになる。この具体例と同様の構成および動作によって、上記ＲＢが実現されることになる。

【００７８】
（再利用処理の概略）
次に、関数およびループのそれぞれの場合について、再利用処理の概略について説明する。

【００７９】
まず、関数の場合について説明する。関数から復帰するまでに次の関数を呼び出した場合、または、登録すべき入出力が再利用表の容量を超える、引数の第７ワードを検出する、途中でシステムコールや割り込みが発生する、などの擾乱が発生しなかった場合、復帰命令を実行した時点で、登録中の入出力表エントリを有効にする。

【００８０】
以降、図１を参照しながら説明すると、関数を呼び出す前に、(1)ＲＦに登録されているエントリにおける関数の先頭アドレスに、該当関数の先頭アドレスと一致するものがあるかを検索する。一致するものがある場合には、(2)ＲＢに登録されている該当関数に関するエントリにおける引数が、呼び出す関数の引数と完全に一致するエントリを選択する。そして、(3)関連する主記憶アドレスすなわち少なくとも１つのMaskが有効であるReadアドレスをＲＦからすべて参照して、(4)ＲＢに登録されている内容と一致比較を行う。全ての入力が一致した場合に、(5)ＲＢに登録済の出力（返り値、大域変数、およびＡの局所変数）を主記憶３に書き戻すことによって、関数の実行を省略する、すなわち関数の再利用を実現することができる。

【００８１】
次に、ループの場合について説明する。ループが完了する以前に関数から復帰したり、前記した擾乱が発生したりするなど、ループの入出力登録が中止されなければ、登録中のループに対応する後方分岐命令を検出した時点で、登録中の入出力表エントリを有効にし、そのループの登録を完了する。

【００８２】
さらに、後方分岐命令が成立する場合は、次のループが再利用可能かどうかを判断する。すなわち、図１を参照しながら説明すると、後方分岐する前に、(1)ＲＦに登録されているエントリにおけるループの先頭アドレスに、該当ループの先頭アドレスと一致するものがあるかを検索する。一致するものがある場合には、(2)ＲＢに登録されている該当ループに関するレジスタ入力値が、呼び出すループのレジスタ入力値と完全に一致するエントリを選択する。そして、(3)関連する主記憶アドレスをＲＦから全て参照して、(4)ＲＢに登録されている内容と一致比較を行う。全ての入力が一致した場合に、(5)ＲＢに登録済の出力（レジスタおよび主記憶出力値）を主記憶３に書き戻すことによってループの実行を省略する、すなわちループの再利用を実現することができる。

【００８３】
再利用した場合、ＲＢに登録されている分岐方向に基づいて、さらに次のループに関して同様の処理を繰り返す。一方、次のループが再利用不可能であれば、次のループを通常に実行し、ＲＦおよびＲＢへの登録を開始する。

【００８４】
（命令区間の実行時における処理の流れ）
次に、命令がデコードされた場合の具体的な処理の流れについて説明する。以下では、

命令がデコードされた結果、関数呼び出し命令である場合、関数復帰命令である場合、後方分岐成立の場合、後方分岐不成立の場合、およびその他の命令の場合について、それぞれ処理の流れを説明する。

【００８５】

（関数呼び出し命令である場合）

命令がデコードされた結果、関数呼び出し命令である場合の処理を図３に示すフローチャートを参照しながら以下に説明する。まずステップ１（以降、Ｓ１のように称する）において、引数の第７ワードを検出したか否かが判定される。Ｓ１においてＹＥＳ、すなわち、引数の第７ワードを検出したと判定された場合には、ＲＷに登録されている登録中ＲＢエントリを全て無効化し、Ｓ６に移行して、プログラムカウンタを関数の先頭へ進め、処理を終了する。

【００８６】

一方、Ｓ１においてＮＯ、すなわち、引数の第７ワードを検出していないと判定された場合には、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されているか否かを検索する（Ｓ２）。Ｓ２においてＹＥＳ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていると判定された場合には、後述するＳ７のステップに移行する。

【００８７】

Ｓ２においてＮＯ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていないと判定された場合、該関数のためのＲＦエントリおよびＲＢエントリを確保しようと試み、(1)既存のＲＦエントリがあるか、(2)登録作業中につき追い出すことのできないＲＦエントリ以外に、使用可能なＲＦエントリがあるか、または(3)登録作業中につき追い出すことができないＲＢエントリ以外に、使用可能なＲＢエントリがあるかを判定する（Ｓ３）。

【００８８】

Ｓ３においてＮＯ、すなわち、使用可能なＲＦ・ＲＢエントリがないと判定された場合には、登録を開始せず、ＲＷに登録されているＲＢを全て無効化し（Ｓ５）、ＲＷを空にする。一方、Ｓ３においてＹＥＳ、すなわち、使用可能なＲＦ・ＲＢエントリがあると判定された場合には、該関数のためのＲＦエントリおよびＲＢエントリを確保し、ＲＷに登録する（Ｓ４）。ここで、ＲＷに登録した際に、ＲＷに登録されているＲＷエントリが溢れた際には、最も古いＲＷエントリを削除し、対応するＲＢを無効化する。Ｓ３またはＳ４が行われた後に、プログラムカウンタを関数の先頭へ進め（Ｓ６）、処理を終了する。

【００８９】

一方、Ｓ２においてＹＥＳ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていると判定された場合、該関数は再利用可能であることになる。すなわち、ＲＢから出力値を求めるとともに、レジスタおよび主記憶３にこの出力値を書き込む（Ｓ７）。そして、登録中の関数／ループがＲＷに登録されているか否かを判定し（Ｓ８）、登録されている場合には、再利用を行った関数のＲＢエントリの内容のうち必要なものをＲＷに登録されているエントリに追加する（Ｓ９）。ここで、ＲＷのＴＯＰから順に登録し、途中でＲＢがあふれた場合には、以降、ＲＷのＢＯＴＴＯＭまでに対するＲＢを無効化し、ＲＷから削除する。その後、プログラムカウンタを次の命令へ進め（Ｓ１０）、処理を終了する。

【００９０】

（関数復帰命令である場合）

命令がデコードされた結果、関数復帰命令である場合の処理を図４に示すフローチャートを参照しながら以下に説明する。Ｓ１１において、ＲＷのＴＯＰから順にたどり、関数に対応するＲＦ／ＲＢが検出されるまでに、ループに関するＲＢが検出されるか否かが判定される（Ｓ１２）。ここでループに関するＲＢが検出されると（Ｓ１２においてＹＥＳ）、該当ＲＢを全て無効化するとともに、ＲＷから削除する（Ｓ１３）。

【００９１】

一方、ＲＷ探索中に、該関数に対応するＲＦ／ＲＢを検出したか否かが判定される（Ｓ

１４）。ここで、該関数に対応するＲＦ／ＲＢが検出されると（Ｓ１４においてＹＥＳ）、該当ＲＢエントリを有効化するとともに、ＲＷから削除する（Ｓ１５）。

【００９２】

その後、復帰命令を実行し（Ｓ１６）、処理を終了する。

【００９３】

（後方分岐成立である場合）

命令がデコードされた結果、後方分岐成立である場合の処理を図５に示すフローチャートを参照しながら以下に説明する。まず、ＲＷのＴＯＰから順にたどり、関数に対応するＲＢを検出するか否かが判定される（Ｓ２１）。Ｓ２１においてＹＥＳ、すなわち、関数に対応するＲＢを検出した場合には、後述するＳ２４のステップに移行する。

【００９４】

一方、Ｓ２１においてＮＯ、すなわち、関数に対応するＲＢを検出しない場合には、次に、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致するか否かが判定される（Ｓ２２）。Ｓ２２においてＮＯ、すなわち、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致しないと判定されると、後述するＳ２４のステップに移行する。

【００９５】

Ｓ２２においてＹＥＳ、すなわち、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致すると判定された場合、ＲＷのＴＯＰから該ＲＢの手前までのＲＢを全て無効化し（Ｓ２３）、ＲＷから削除する。また、該ＲＢエントリを有効化し、かつtaken=1とし、ＲＷから削除する。

【００９６】

次に、Ｓ２４において、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されているか否かが判定される。Ｓ２４においてＹＥＳ、すなわち、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されている場合には、後述するＳ３０のステップに移行する。

【００９７】

一方、Ｓ２４においてＮＯ、すなわち、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されていない場合には、次ループのためのＲＦエントリおよびＲＢエントリを確保しようと試み、(1)既存のＲＦエントリがあるか、(2)登録作業中につき追い出すことができないＲＦエントリ以外に、使用可能なＲＦエントリがあるか、または(3)登録作業中につき追い出すことができないＲＢエントリ以外に、使用可能なＲＢエントリがあるかが判定される（Ｓ２５）。

【００９８】

Ｓ２５においてＮＯ、すなわち、使用可能なＲＦ・ＲＢエントリがないと判定された場合には、登録を開始せずに、ＲＷに登録されているＲＢを全て無効化し（Ｓ２６）、ＲＷを空にする。その後、Ｓ２９において、プログラムカウンタを条件分岐先へ進め、処理を終了する。

【００９９】

一方、Ｓ２５においてＹＥＳ、すなわち、使用可能なＲＦ・ＲＢエントリがあると判定された場合には、その使用可能なＲＦ・ＲＢエントリを確保し、確保したＲＦ・ＲＢをＲＷに登録する（Ｓ２７）。また、ＲＢにループ終了アドレス（後方分岐命令自身のアドレス）を登録する。ここで、ＲＷへの登録を行った際にＲＷが溢れた場合には、最も古いＲＷエントリを削除し（Ｓ２８）、それに対応するＲＢを無効化する。その後、Ｓ２９において、プログラムカウンタを条件分岐先へ進め、処理を終了する。

【０１００】

一方、前記したＳ２４においてＹＥＳとなった場合、次ループは再利用可能であることになるので、ＲＢから出力値を求め、この値をレジスタおよび主記憶３に書き込む（Ｓ３０）。ここで、登録中の関数／ループがＲＷに登録されているか否かが判定され（Ｓ３１）、登録されている場合、再利用を行ったループのＲＢエントリの内容のうち必要なもの

をＲＷに登録されているエントリに追加する（Ｓ３２）。このとき、ＲＷのＴＯＰから順に登録し、途中でＲＢが溢れた場合、以降、ＲＷのＢＯＴＴＯＭまでに対するＲＢを無効化し、ＲＷから削除する。

【０１０１】

その後、プログラムカウンタは、次ループ先頭ではなく、該ＲＢ中のtakenの値に応じて、taken=1の場合は自命令、taken=0の場合は、ＲＢ中に記憶しておいたループ終了アドレスの次へ進める。その後、処理を終了する。

【０１０２】

（後方分岐不成立である場合）

命令がデコードされた結果、後方分岐不成立である場合の処理を図６に示すフローチャートを参照しながら以下に説明する。まず、ＲＷのＴＯＰから順に検索し（Ｓ４１）、関数に対応するＲＢを検出したか否かが判定される（Ｓ４２）。Ｓ４２においてＹＥＳ、すなわち、関数に対応するＲＢを検出したと判定された場合、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１０３】

Ｓ４２においてＮＯ、すなわち、関数に対応するＲＢを検出していないと判定された場合、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスが一致するか否かが判定される（Ｓ４３）。Ｓ４３においてＮＯ、すなわち、該後方分岐命令に対応するＲＦ／ＲＢを検出していないと判定された場合、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１０４】

一方、Ｓ４３においてＹＥＳ、すなわち、該後方分岐命令に対応するＲＦ／ＲＢを検出したと判定された場合、ＲＷのＴＯＰから該ＲＢの手前までのＲＢを全て無効化し（Ｓ４４）、ＲＷから削除する。また、該ＲＢエントリを有効化し、かつtaken=0とし、ＲＷから削除する（Ｓ４５）。その後、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１０５】

（その他の命令である場合）

次に、命令がデコードされた結果、上記以外のその他の命令である場合について説明する。その他の命令である場合、レジスタＲ／Ｗ、主記憶Ｒ／Ｗが実行される。その際に、ＲＷが空でなければ、以下の手順によってレジスタＲ／Ｗ、主記憶Ｒ／ＷをＲＷに登録されているＲＢに対して登録する。以下では、（１）汎用レジスタＲＥＡＤの場合、（２）汎用レジスタＷＲＩＴＥの場合、（３）浮動小数点レジスタＲＥＡＤの場合、（４）浮動小数点レジスタＷＲＩＴＥの場合、（５）条件コードレジスタＩＣＣ－ＲＥＡＤの場合、（６）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合、（７）浮動小数点条件コードレジスタＦＣＣ－ＲＥＡＤの場合、（８）浮動小数点条件コードレジスタＦＣＣ－ＷＲＩＴＥの場合、（９）主記憶ＲＥＡＤの場合、（１０）主記憶ＷＲＩＴＥの場合についてそれぞれ説明する。

【０１０６】

（１）汎用レジスタＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（１－１）該ＲＢがリーフ関数かつ%o0-6の場合、または該ＲＢが非リーフ関数かつ%i0-6の場合、arg[0-5].V=0であれば、arg[0-5].V=1に変更し、arg[0-5].Valに読み出しデータを記録する。その後、さらにＲＷをたどり、該ＲＢが関数の場合、処理を終了する。一方、該ＲＢが関数ではない（ループである）場合、arg[0-5].V=0であれば、arg[0-5].V=1に変更し、arg[0-5].Valに読み出しデータを記録し、処理を終了する。

【０１０７】

一方、（１－２）該ＲＢがループの場合、（ａ）%g0-7でgrr[0-7].V=0であれば、grr[0-7].V=1に変更し、grr[0-7].Valに読み出しデータを記録し、処理を終了する。（ｂ）%o0-7でarg[0-7].V=0であれば、arg[0-7].V=1に変更し、arg[0-7].Valに読み出しデータを記

録し、処理を終了する。（ｃ）%l0-7でlrr[0-7].V=0であれば、lrr[0-7].V=1に変更し、lrr[0-7].Valに読み出しデータを記録し、処理を終了する。（ｄ）%i0-7でirr[0-7].V=0であれば、irr[0-7].V=1に変更し、irr[0-7].Valに読み出しデータを記録し、次のＲＷエントリに進む。

【０１０８】

（２）汎用レジスタＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（２－１）該ＲＢがリーフ関数かつ%o0-5の場合、または該ＲＢが非リーフ関数かつ%i0-5の場合、arg[0-5].V=0であれば、以降の読み出しは入力ではないことを示すために、arg[0-5].V=2に変更する。さらに、%o0-1/%i0-1について、rti[0-1].V=1に変更し、rti[0-1].Valに書き込みデータを記録する。その後、さらにＲＷをたどり、該ＲＢが関数の場合、処理を終了する。一方、該ＲＢが関数ではない（ループである）場合、arg[0-1].V=0であれば、以降の読み出しは入力ではないことを示すために、arg[0-1].V=2に変更し、rti[0-1].V=1に変更し、rti[0-1].Valに書き込みデータを記録し、処理を終了する。

【０１０９】

一方、（２－２）該ＲＢがループの場合、（ａ）%g0-7でgrr[0-7].V=0であれば、grr[0-7].V=2に変更し、grr[0-7].Valに書き込みデータを記録し、処理を終了する。（ｂ）%o0-7でarg[0-7].V=0であれば、arg[0-7].V=2に変更し、arg[0-7].Valに書き込みデータを記録し、処理を終了する。（ｃ）%l0-7でlrr[0-7].V=0であれば、lrr[0-7].V=2に変更し、lrr[0-7].Valに書き込みデータを記録し、処理を終了する。（ｄ）%i0-7でirr[0-7].V=0であれば、irr[0-7].V=2に変更し、irr[0-7].Valに書き込みデータを記録し、次のＲＷエントリに進む。

【０１１０】

（３）浮動小数点レジスタＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（３－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（３－２）該ＲＢがループの場合、frr[0-31].V=0であれば、frr[0-31].V=1に変更し、frr[0-31].Valに読み出しデータを記録し、処理を終了する。

【０１１１】

（４）浮動小数点レジスタＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（４－１）該ＲＢが関数かつ%f0-1の場合、rtf[0-1].V=1に変更し、rtf[0-1].Valに書き込みデータを記録する。さらにＲＷをたどり、frr[0-1].V=0であれば、以降の読み出しは入力ではないことを示すために、frr[0-1].V=2に変更し、rtf[0-1].V=1に変更し、rtf[0-1].Valに書き込みデータを記録し、処理を終了する。

【０１１２】

一方、（４－２）該ＲＢがループの場合、frr[0-31].V=0であれば、frr[0-31].V=2に変更し、frw[0-31].V=1に変更し、frw[0-7].Valに書き込みデータを記録し、処理を終了する。

【０１１３】

（５）条件コードレジスタＩＣＣ－ＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（５－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（５－２）該ＲＢがループの場合、icr.V=0であれば、icr.V=1に変更し、icr.Valに読み出しデータを記録し、処理を終了する。

【０１１４】

（６）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（６－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（６－２）該ＲＢがループの場合、icr.V=0であれば、icr.V=2、icw.V=1に変更し、icw.Valに書き込みデータを記録し、処理を終了する。

【０１１５】
（７）浮動小数点条件コードレジスタＦＣＣ－ＲＥＡＤの場合
まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（７－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（７－２）該ＲＢがループの場合、fcr.V=0であれば、fcr.V=1に変更し、fcr.Valに読み出しデータを記録し、処理を終了する。
【０１１６】
（８）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合
まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（８－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（８－２）該ＲＢがループの場合、fcr.V=0であれば、fcr.V=2、fcw.V=1に変更し、fcw.Valに書き込みデータを記録し、処理を終了する。
【０１１７】
（９）主記憶ＲＥＡＤの場合
まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、ＲＢにＷＲＩＴＥデータとして登録済である場合は、その値を使用する。一方、上記の場合ではなく、ＲＢにＲＥＡＤデータとして登録済である場合には、その値を使用する。さらに、いずれにも登録済でない場合は、キャッシュを経由して主記憶３から読み込む。
【０１１８】
その後、再度ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（ａ）アドレスが、ＲＢに登録されているsp+64の場合、構造体ポインタの読み出しであるので、arg0.V=0であれば、arg0.V=1に変更し、arg0.Valに読み出しデータを記録する。（ｂ）上記の（ａ）の場合でなく、アドレスが、LIMIT以上sp+92未満であれば、登録不要領域であるので、何もしない。（ｃ）上記の（ｂ）の場合でない場合、ＷＲＩＴＥデータとして登録済であるかどうかを検査し、そうであれば、すでに上書きされたあとのＲＥＡＤであるので登録不要であり、何もしない。（ｄ）上記の（ｃ）でない場合、ＲＥＡＤデータとして登録済であるかどうかを検査し、そうであれば、すでに登録済であるので登録不要であり、何もしない。（ｅ）上記の（ｄ）でない場合、ＲＥＡＤデータとしての登録が必要であるので、ＲＦに主記憶ＲＥＡＤアドレスを確保し、ＲＥＡＤデータとして登録する。ＲＦに主記憶アドレスを確保できなかった場合には、登録不能であるため、そのＲＷエントリからＢＯＴＴＯＭまでに対応するＲＢエントリを全て無効化する。
【０１１９】
（１０）主記憶ＷＲＩＴＥの場合
まず、キャッシュを経由して、主記憶３に書き込む。そして、ベースレジスタが１４（％ｓｐ）かつオフセットが９２以上である場合、引数の第７ワードを検出したことを記憶する。
【０１２０】
その後、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（ａ）アドレスが、ＲＢに登録されているsp+64の場合、構造体ポインタの読み出しであるので、arg0.V=0であれば、arg0.V=2に変更する。（ｂ）上記の（ａ）の場合ではなく、アドレスがLIMIT以上sp+92未満であれば、登録不要領域であるので、何もしない。（ｃ）上記の（ｂ）の場合でない場合、ＷＲＩＴＥデータとして登録済であるかどうかを検査し、そうであれば、すでにアドレスは登録済であるので、内容を新しいＷＲＩＴＥデータに更新する。（ｄ）上記の（ｃ）でない場合、ＷＲＩＴＥデータとしての登録が必要であるので、ＲＦに主記憶ＷＲＩＴＥアドレスを確保し、ＷＲＩＴＥデータとして登録する。ＲＦに主記憶アドレスを確保できなかった場合には、登録不能であるため、そのＲＷエントリからＢＯＴＴＯＭまでに対応するＲＢエントリを全て無効化する。
【０１２１】
（ループを含む多重再利用）
１レベルで上記のような再利用機構を用いた場合、図１７（ａ）に示した例で言えば、リーフ関数としての関数Ｂや、関数Ｂの内部にあるループＣなどをそれぞれ再利用するこ

とが可能となる。これに対して、ある関数を一度実行しただけで、その関数の内部に含まれる関数やループを含む全ての命令区間が再利用可能となるように登録を行う仕組みが多重再利用である。例えば上記の例で言えば、多重再利用によれば、関数Ａを一度実行しただけで、入れ子関係にあるＡ，Ｂ，Ｃの全ての命令区間が再利用可能となる。以下に、多重再利用を実現する上で必要とされる機能拡張について説明する。

【０１２２】

図７に、一例として、関数Ａおよび関数Ｄの概念的な構造を示す。同図に示す例では、関数Ａの内部にループＢが存在しており、ループＢの内部にループＣが存在しており、ループＣにおいて関数Ｄが呼び出されるようになっている。そして、関数Ｄの内部にループＥが存在しており、ループＥの内部にループＦが存在している。

【０１２３】

図８は、図７に示す関数Ａ，ＤおよびループＢ，Ｃ，Ｅ，Ｆの入れ子構造において、内側の構造のレジスタ入出力（太枠セル領域）が、外側の構造のレジスタ入出力となる影響範囲（矢印）について示している。例えば、ループＦの内部において入力として参照された％ｉ０～５は、ループＥおよび関数Ｄに対する入力でもあり、さらに、関数Ｄを呼び出したループＣおよびループＢに対する入力（ただし％ｏ０～５に読み替える）でもある。一方、関数Ａにとって％ｏ０～５は局所変数に相当するので、％ｉ０～５（％ｏ０～５）は、関数Ａに対してのレジスタ入力とはならない。すなわち、％ｉ０～５（％ｏ０～５）の影響範囲はループＢまでとなる。別の見方をすれば、関数Ｄの内部で％ｉ０～５が参照された場合には、ループＢが直接的に％ｏ０～５を参照しなくても、％ｏ０～５をループＢの入力値として登録する必要がある。ループＦ内部において出力された％ｉ０～１についても同様である。

【０１２４】

浮動小数点レジスタはレジスタウィンドウに含まれないので、出力された％ｆ０～１は、関数Ａを含む全階層の出力となる。一方、その他のレジスタ入出力は、関数を超えて影響がおよぶことはない。すなわち、ループＦ内部における入出力、すなわち、レジスタ入力としての％ｉ６～７、％ｇ,ｌ,ｏ、％ｆ０～３１、％ｉｃｃ、％ｆｃｃ、およびレジスタ出力としての％Ｉ２～７、％ｇ,ｌ,ｏ、％ｆ２～３１、％ｉｃｃ、％ｆｃｃの影響範囲はループＥまでとなる。主記憶３に対する入出力については、前述した、関数呼び出し直前の％ｓｐ(SP)と比較する方法を入れ子の全階層に対して適用することにより、影響範囲を特定することができる。

【０１２５】

以上のことから、多重再利用を実現するには、前述したＲＦおよびＲＢを関数やループの入れ子構造と関連づける機構が必要である。図９に示すように、再利用ウィンドウ（ＲＷ）を装備することによって、現在実行中かつ登録中であるＲＦおよびＲＢの各エントリ（図中ではＡ、Ｂ、Ｃと示す）をスタック構造として保持する。関数やループの実行中は、ＲＷに登録されている全てのエントリについて、これまでに述べた方法に基づいて、レジスタおよび主記憶参照を登録していく。

【０１２６】

この際に、あるエントリに関して、（１）登録可能項目数の超過、（２）引数の第７ワードの検出、（３）システムコールの検出、によって再利用不可能であると判断した場合には、ＲＷを用いて、そのエントリに対応するＲＢおよび上位のＲＢを特定し、登録を中止することができる。

【０１２７】

なお、ＲＷの深さは有限であるものの、一度に登録可能な多重度を超えて関数やループを検出した場合には、外側の命令区間から順次登録を中止し、より内側の命令区間を登録対象に加えることによって、入れ子関係の動的変化に追随することができる。また、実行および登録中（例えばＡ）に、再利用可能な命令区間（例えばＤ）に遭遇した場合には、登録済の入出力をそのまま登録中エントリに追加することによって、ＲＷの深さを超えるＡの多重再利用も可能となる。

【０１２８】
（並列事前実行）
以上に述べた、関数やループの多重再利用では、ＲＢエントリの生存時間よりも同一パラメータが出現する間隔が長い場合や、パラメータが単調に変化し続ける場合には全く効果がないことになる。すなわち、ＲＢエントリの生存時間よりも同一パラメータが出現する間隔が長い場合には、ある関数またはループがＲＢに登録されたとしても、その登録された関数またはループに関して同一パラメータが次に出現した際には、すでにその関数またはループがＲＢエントリから消えていることになり、再利用できないことになる。また、パラメータが単調に変化し続ける場合には、該当する関数やループがＲＢに登録されていても、パラメータが異なることによって再利用できないことになる。
【０１２９】
これに対して、多重再利用を行うプロセッサとしてのＭＳＰ１Ａとは別に、命令区間の事前実行によってＲＢエントリを有効にするプロセッサとしてのＳＳＰ１Ｂを複数個設けることによって、さらなる高速化を図ることができる。
【０１３０】
並列事前実行機構を行うためのハードウェア構成は、前記した図２に示すような構成となる。同図に示すように、ＲＷ４Ａ・４Ｂ、演算器５Ａ・５Ｂ、レジスタ６Ａ・６Ｂ、キャッシュ７Ａ・７Ｂは、各プロセッサごとに独立して設けられている一方、ＲＦ／ＲＢ２、および主記憶３は全てのプロセッサが共有するようになっている。同図において、破線は、ＭＳＰ１ＡおよびＳＳＰ１ＢがＲＦ／ＲＢ２に対して入出力を登録するパスを示している。
【０１３１】
ここで、並列事前実行を実現する上での課題は、（１）どのように主記憶一貫性を保つか、（２）どのように入力を予測するかが挙げられる。以下に、これらの課題に対する解決手法について説明する。
【０１３２】
（主記憶一貫性に関する課題の解決方法）
まず、上記の課題（１）どのように主記憶一貫性を保つかについて説明する。特に予測した入力パラメータに基づいて命令区間を実行する場合、主記憶３に書き込む値がＭＳＰ１ＡとＳＳＰ１Ｂとで異なることになる。これを解決するために、図２に示すように、ＳＳＰ１Ｂは、ＲＢへの登録対象となる主記憶参照にはＲＦ／ＲＢ２、また、その他の局所的な参照にはＳＳＰ１Ｂごとに設けた局所メモリとしてのＬｏｃａｌ７Ｂを使用することとし、Ｃａｃｈｅ７Ｂおよび主記憶３への書き込みを不要としている。なお、ＭＳＰ１Ａが主記憶３に対して書き込みを行った場合には、対応するＳＳＰ１Ｂのキャッシュラインが無効化される。
【０１３３】
具体的には、ＲＢへの登録対象のうち、読み出しが先行するアドレスについては主記憶３を参照し、ＭＳＰ１Ａと同様にアドレスおよび値をＲＢへ登録する。以後、主記憶３ではなくＲＢを参照することによって、他のプロセッサからの上書きによる矛盾の発生を避けることができる。局所的な参照については、読み出しが先行するということは、変数を初期化せずに使うことに相当し、値は不定でよいことになるので、主記憶３を参照する必要はない。
【０１３４】
なお、局所メモリとしてのＬｏｃａｌ７Ｂの容量は有限であり、関数フレームの大きさがＬｏｃａｌ７Ｂの容量を超えた場合など、実行を継続できない場合は、事前実行を打ち切るようにする。また、事前実行の結果は主記憶３に書き込まれないので、事前実行結果を使って、さらに次の事前実行を行うことはできない。
【０１３５】
（予測機構）
次に、上記の課題（２）どのように入力を予測するかについて説明する。事前実行に際

しては、ＲＢの使用履歴に基づいて将来の入力を予測し、ＳＳＰ１Ｂへ渡す必要がある。このために、ＲＦ／ＲＢ２には、予測処理部２Ｂが設けられている。この予測処理部２Ｂは、ＲＦの各エントリごとに設けた小さなプロセッサによって構成され、ＭＳＰ１ＡやＳＳＰ１Ｂとは独立して入力予測値を求めるものである。

【０１３６】

前記したように、従来の入力予測では、ＲＢにおける入力側に登録された全てのアドレスが一律に扱われたことによって、予測の的中率を下げる結果となっている。この問題を解決するためには、予測が的中する可能性が高いアドレスと、予想が外れる可能性が高いアドレスを区別するとともに、値の変化にも着目して必要最小限のアドレスのみを予測対象とすることが必要である。

【０１３７】

予測が的中することが期待できるアドレスとは、アドレスが固定しており、かつ、値が単調変化するアドレスである。このようなアドレスには、ラベルによって参照される帯域変数、および、スタックポインタやフレームポインタをベースレジスタとして参照される局所変数（フレーム内変数）などがある。

【０１３８】

これらのアドレスを識別するために、ロード命令実行時のアドレス計算が参照するレジスタに定数フラグ（Const-FLAG）が設けられる。スタックポインタやフレームポインタとして用いるレジスタについては無条件に定数フラグがセットされるものとする。その他のレジスタについては、定数をセットする命令が実行された時に定数フラグ（Const-FLAG）がセットされるものとする。

【０１３９】

次に、過去に参照したアドレスのうち、一度も書き込みが行われないアドレスについては、内容が変化していないことが保証されることになり、このようなアドレスについては予測する必要がないことになる。よって、このようなアドレスを区別するために、書き込みが行われたことを示す変更フラグ（C-FLAG）が設けられる。入力要素としてのアドレスをＲＦ／ＲＢに新規に記録する時には、該アドレスに対応する変更フラグ（C-FLAG）がリセットされ、登録後に該アドレスに対してストア命令が実行された時に、変更フラグ（C-FLAG）がセットされる。

【０１４０】

また、入力要素としてのアドレスを履歴保存対象とするか否かを示す履歴マスク（P-Mask）が設けられる。入力要素としてのアドレスをＲＦ／ＲＢに新規に記録する時には、該アドレスに対応する履歴マスク（P-Mask）（履歴フラグ）がリセットされる。そして、ロード命令実行時に、該アドレスを生成したレジスタに対応する定数フラグ（Const-FLAG）がセットされている場合には、履歴マスク（P-Mask）のうちロード対象となったバイト位置がセットされる。

【０１４１】

以上の定数フラグ（Const-FLAG）、変更フラグ（C-FLAG）、および履歴マスク（P-Mask）の設定の制御は、ＲＦ／ＲＢ２に設けられているＲＢ登録処理部２Ａによって行われる。このＲＢ登録処理部２Ａは、小さなプロセッサによって構成され、上記のような判断を行うことによって定数フラグ（Const-FLAG）、変更フラグ（C-FLAG）、および履歴マスク（P-Mask）の設定を行う。

【０１４２】

（命令区間の実行例）

ここで、命令区間の一例として、図２０に示す命令区間が、図１に示したＲＦおよびＲＢの構成によって実行された場合の例について説明する。同図において、ＰＣは、該命令区間が開始された際のＰＣ値を示している。すなわち、命令区間の先頭が１０００番地となっている。また、図１０は、図２０に示す命令区間が実行された場合のＲＢにおける実際の登録状況を示している。

【０１４３】

第１の命令において、アドレス定数Ａ１がレジスタＲ０にセットされる。この命令は、定数をセットする命令であるので、レジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされる。

【０１４４】

第２の命令において、レジスタＲ０の内容をアドレスとする主記憶３からロードされた４バイトデータ（00110000）がレジスタＲ１に格納される。この場合、アドレスＡ１、マスク（FFFFFFFF）、データ（00110000）は、入力としてＲＢにおけるInput側の第１列に登録され、レジスタ番号Ｒ１、マスク（FFFFFFFF）、およびデータ（00110000）は出力としてＲＢにおけるOutput側の第１列に登録される。

【０１４５】

また、アドレスとして用いたレジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされているので、アドレスＡ１に対応する履歴マスク（P-Mask）がセットされる。ここで、対象となるデータは（00110000）の４バイトデータであるので、これに対応して、アドレスＡ１に対応する履歴マスク（P-Mask）には（FFFFFFFF）がセットされる。そして、レジスタＲ１は、定数がセットされるものではないことになるので、レジスタＲ１に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１４６】

第３の命令において、アドレス定数Ａ２がレジスタＲ０にセットされる。この命令は、定数をセットする命令であるので、レジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされる。

【０１４７】

第４の命令において、レジスタＲ０の内容をアドレスとする主記憶３からロードされた１バイトデータ（02）がレジスタＲ２に格納される。この場合、アドレスＡ２、マスク（FF000000）、およびデータ（02）は入力としてＲＢにおけるInput側の第２列に登録される。この際、アドレスＡ２の残り３バイトについては、Don't Careを意味する「－」が格納される。レジスタ番号Ｒ２、マスク（FFFFFFFF）およびデータ（00000002）は出力としてＲＢにおけるOutput側の第２列に登録される。

【０１４８】

また、アドレスとして用いたレジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされているので、アドレスＡ２に対応する履歴マスク（P-Mask）がセットされる。ここで、対象となるデータは（02）の１バイトデータであるので、これに対応して、アドレスＡ２に対応する履歴マスク（P-Mask）には（FF000000）がセットされる。そして、レジスタＲ２は、定数がセットされるものではないことになるので、レジスタＲ２に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１４９】

第５の命令において、アドレス（Ａ２＋Ｒ２）からロードされた１バイトデータ（22）がレジスタＲ２に格納されている。アドレスＲ２の値は（02）であったので、アドレス（Ａ２＋02）、およびデータ（22）が、入力としてＲＢにおけるInput側の第２列に追加登録される。この際、アドレス（Ａ２＋02）の部分に登録が行われ、アドレス（Ａ２＋01）および（Ａ２＋03）に対応する部分は、Don't Careを意味する「－」のままとなる。すなわち、アドレスＡ２に対応するマスクは（FF00FF00）となる。レジスタ番号Ｒ２、マスク（FFFFFFFF）、およびデータ（00000022）は、出力としてＲＢにおけるOutput側の第２列に上書きされる。

【０１５０】

また、アドレスとして用いたレジスタＲ２に対応する定数フラグ（Const-FLAG）がリセットされているので、アドレス（Ａ２＋02）に対応する履歴マスク（P-Mask）はセットされない。すなわち、アドレスＡ２に対応する履歴マスク（P-Mask）は（FF000000）のままとなる。そして、レジスタＲ２は、定数がセットされるものではないことになるので、レジスタＲ２に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１５１】

第６の命令において、アドレス定数Ａ３がレジスタＲ０にセットされる。この命令は、定数をセットする命令であるので、レジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされる。

【０１５２】

第７の命令において、レジスタＲ０の内容をアドレスとする主記憶３からロードされた１バイトデータ（33）がレジスタＲ３に格納される。この場合、アドレスＡ３、マスク（00FF0000）、およびデータ（33）は入力としてＲＢにおけるInput側の第３列に登録される。レジスタ番号Ｒ３、マスク（FFFFFFFF）、およびデータ（00000033）は出力としてＲＢにおけるOutput側の第３列に登録される。

【０１５３】

また、アドレスとして用いたレジスタＲ０に対応する定数フラグ（Const-FLAG）がセットされているので、アドレスＡ３に対応する履歴マスク（P-Mask）がセットされる。ここで、対象となるデータは（33）の１バイトデータであるので、これに対応して、アドレスＡ３に対応する履歴マスク（P-Mask）には（00FF0000）がセットされる。そして、レジスタＲ３は、定数がセットされるものではないことになるので、レジスタＲ３に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１５４】

第８の命令において、アドレス（Ｒ１＋Ｒ２）からロードされた１バイトデータ（44）がレジスタＲ４に格納される。この場合、アドレスＲ１とアドレスＲ２は命令区間の内部にて上書きされたレジスタのアドレスとなるので、アドレスＲ１およびアドレスＲ２は命令区間の入力とはならない。一方、アドレス（Ｒ１＋Ｒ２）によって生成されたアドレスＡ４は命令区間の入力であるので、アドレスＡ４、マスク（00FF0000）、およびデータ（44）は入力としてＲＢにおけるInput側の第４列に登録される。レジスタ番号Ｒ４、マスク（FFFFFFFF）、およびデータ（00000044）は出力としてＲＢにおけるOutput側の第４列に登録される。

【０１５５】

また、アドレスとして用いたレジスタＲ１およびレジスタＲ２に対応する定数フラグ（Const-FLAG）がリセットされているので、アドレスＡ４に対応する履歴マスク（P-Mask）はセットされない。すなわち、アドレスＡ４に対応する履歴マスク（P-Mask）は（00000000）となる。そして、レジスタＲ４は、定数がセットされるものではないことになるので、レジスタＲ４に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１５６】

第９の命令において、レジスタＲ５から値が読み出され、読み出された値に１が加えられた結果が再びレジスタＲ５に格納される。この場合、レジスタＲ５、マスク（FFFFFFFF）、およびデータ（00000100）は入力としてＲＢにおけるInput側の第５列に登録される。また、レジスタ番号Ｒ５、マスク（FFFFFFFF）、およびデータ（00000101）は出力としてＲＢにおけるOutput側の第５列に登録される。この時、レジスタＲ５は、定数がセットされるものではないことになるので、レジスタＲ５に対応する定数フラグ（Const-FLAG）はリセットされる。

【０１５７】

その後、アドレスＡ２、およびアドレスＡ３に対してストア命令が実行され、アドレスＡ２、およびアドレスＡ３に対して変更フラグ（C-FLAG）がセットされたとする。

【０１５８】

以上の結果、変更フラグ（C-FLAG）がセットされ、かつ、履歴マスク（P-Mask）がセットされたマスク位置は、アドレスＡ２の第１バイト、アドレスＡ３の第２バイトのみとなる。このマスク位置のみに対応するアドレス、マスク、および値が、予測対象として、命令区間ごとに過去の入力履歴を保持する履歴情報として、ＲＢのエントリに記録される。また、ＲＢの入力パターンに登録されたレジスタについては無条件に予測対象として履歴として記録される。

【０１５９】

図１１は、図２０に示す命令区間が繰り返し実行された場合における、履歴としてＲＢに登録された例を示している。同図に示すように、ＲＢには、アドレスＡ２の列に履歴マスク（P-Mask）として（FF000000）、アドレスＡ３の列に履歴マスク（P-Mask）として（00FF0000）、およびアドレスＲ５の列に履歴マスク（P-Mask）として（FFFFFFFF）が記憶される。そして、Timeが１～４に変化する間に、各アドレスにおける履歴マスク（P-Mask）に対応する値が変化することになる。各履歴の間に示されるdiffは、対応する入力要素の値の変化量（差分）を示している。このdiffは、予測処理部２Ｂによって算出される。

【０１６０】

同図に示す例では、アドレスＡ２およびアドレスＲ５の列に関しては、Timeが１～４に変化する間におけるdiffが全て01となっている。よって、これらのアドレスに対応する値は、単位時間あたりに01ずつ増加していくことが予想される。一方、アドレスＡ３の列に関しては、Timeが１～４に変化する間に、diffは00であったり02であったりしている。したがって、アドレスＡ３に関しては、予測することが困難であることがわかる。

【０１６１】

以上より、予測処理部２Ｂは、履歴において、差分が一定となっているアドレスに関して、該差分がその後も継続するものと仮定して予測を行うとともに、差分が一定でない、または差分が０となっているアドレスに関しては予測を行わないようにする。

【０１６２】

図１２は、上記の予測に基づいて、予測処理部２ＢがアドレスＡ２およびアドレスＲ５の値に関して予測を行った場合の、予測エントリとしてＲＢに記録される入力要素の状態を示している。同図において、アドレス（Ａ２＋4）およびアドレスＡ３に関しては、予測値を求めずに直接主記憶３を参照することによって得られたものとなっている。

【０１６３】

このように入力要素の予測値が算出されると、ＳＳＰ１Ｂが、この予測入力要素に基づいて命令区間を実行することによって出力要素が算出され、この予測出力要素が予測エントリとしてＲＢに記憶される。その後、ＭＳＰ１Ａによって命令区間が実行され、予測エントリとしてＲＢに記憶されている予測入力要素と同じ入力値が入力された場合に、それに対応する予測出力要素を出力することによって再利用が実現されることになる。

【０１６４】

（ＲＦ／ＲＢの第２の構成例）

次に、ＲＦ／ＲＢ２の第２の構成例について、図１３を参照しながら以下に説明する。同図に示すように、ＲＦ／ＲＢ２は、ＲＢ、ＲＦ、ＲＯ１（第２出力パターン記憶手段）、およびＲＯ２（第１出力パターン記憶手段）を備えた構成となっている。

【０１６５】

ＲＢは、比較すべき値であるレジスタ値または主記憶入力値を格納するValue（値格納領域）、およびキー番号を格納するKey（キー格納領域）を備えており、ValueおよびKeyの組み合わせのラインを複数備えている。

【０１６６】

ＲＦは、次に比較すべきレジスタ番号または主記憶アドレスがないことを示す終端フラグＥ、次に比較すべきレジスタ番号または主記憶アドレスの内容が更新されたことを示す比較要フラグ、次に比較すべき対象がレジスタか主記憶かを示すＲ／Ｍ、次に比較すべきレジスタ番号または主記憶アドレスを示すAdr.（検索項目指定領域）、直前に参照したライン番号を示すUP（親ノード格納領域）、次に比較すべきレジスタ番号または主記憶アドレスよりも優先して比較すべきレジスタ番号または主記憶アドレスを示すAlt.（比較要項目指定領域）、および、優先して比較する際に必要なキーを示すDN（比較要キー指定領域）を備えており、これらはＲＢにおける各ラインに対応して設けられている。

【０１６７】

ＲＯ１およびＲＯ２は、ＲＢおよびＲＦによる検索結果により、再利用が可能であると判定された場合に、主記憶および／またはレジスタに出力する出力値を格納するものである。ＲＯ１は、ＲＦの各ラインに１対１で対応して出力値および出力すべきアドレスを格

納している。ＲＯ２は、ＲＯ１のみでは出力値を格納しきれない場合に、格納しきれない分の出力値および出力すべきアドレスを格納している。ＲＯ２からも出力値を読み出す必要がある場合には、ＲＯ１における該当ラインに、ＲＯ２における出力値が格納されているポインタが示されており、このポインタを用いてＲＯ２から出力値の読み出しが行われる。また、ＲＢおよびＲＦは、それぞれＣＡＭおよびＲＡＭによって構成されている。

【０１６８】

（第２の構成例における連想検索動作）

次に、第２の構成例における連想検索動作について説明する。図１に示した構成では、ＲＢにおける各エントリとしての横の行は、一致比較を行うべき入力値の項目を全て含んだものとなっている。すなわち、全ての入力パターンをそれぞれ１つの行としてＲＢに登録するようになっている。

【０１６９】

これに対して、第２の構成例では、一致比較を行うべき入力値の項目を短い単位に区切り、それぞれの比較単位をノードとしてとらえ、入力パターンを木構造としてＲＦおよびＲＢに登録するようになっている。そして、再利用を行う際には、一致するノードを順次選択することによって、最終的に再利用可能かを判断するようになっている。別の言い方をすれば、複数の入力パターンに共通する部分を１つにまとめて、ＲＦおよびＲＢの１行に対応づけるようになっている。

【０１７０】

これにより、冗長性をなくし、ＲＦ／ＲＢ２を構成するメモリの利用効率を向上させることが可能となる。また、入力パターンを木構造としているので、１つの入力パターンをＲＢにおける１つの行としてのエントリに対応付ける必要がないことになる。よって、一致比較を行うべき入力値の項目の数を可変にすることが可能となっている。

【０１７１】

また、ＲＦおよびＲＢは、入力パターンを木構造として登録しているので、一致比較を行う際には、マルチマッチが行われないことになる。つまり、命令区間記憶部２としては、シングルマッチ機構を有する連想検索メモリであれば実現可能となる。ここで、シングルマッチ機構のみを有する連想検索メモリは一般的に市販されている一方、マルチマッチをシングルマッチと同一性能によって報告可能な連想検索メモリは一般的には市販されていない。すなわち、第２の構成例によれば、市販の連想検索メモリを利用することができるので、より短期間かつ低コストで、本実施形態に係るデータ処理装置を実現することが可能となる。

【０１７２】

次に、図１４を参照しながら、ＲＦ／ＲＢ２における連想検索動作の具体例について説明する。まず、命令区間の実行が検出されると、プログラムカウンタ（ＰＣ）およびレジスタの内容（Reg.）がＲＢに入力される。そして、ＲＢにおいて、連想検索により、入力されたこれらの値と、ＲＢのValueの列に登録されている命令区間先頭アドレスおよびレジスタ値とが比較され、値が一致する唯一の行（ライン）が候補（マッチライン）として選択される。この例では、ＲＢにおける「０１」のラインがマッチラインとして選択される。

【０１７３】

次に、マッチラインとして選択されたラインのＲＢにおける番地である「０１」が、エンコード結果としてＲＦに伝達され、キー０１に対応するＲＦにおけるラインが参照される。キー０１に対応するＲＦにおけるラインでは、比較要フラグが「０」であり、比較すべき主記憶アドレスがＡ１となっている。すなわち、主記憶アドレスＡ１に関しては、一致比較を行う必要はないことになる。

【０１７４】

次に、キー０１を用いて、ＲＢにおけるKeyの列に対して検索が行われる。この例では、ＲＢにおける「０３」のラインがマッチラインとして選択される。そして、エンコード結果としてキー０３がＲＦに伝達され、キー０３に対応するＲＦにおけるラインが参照さ

れる。キー０３に対応するＲＦにおけるラインでは、比較要フラグが「１」であり、比較すべき主記憶アドレスがＡ２となっている。すなわち、主記憶アドレスＡ２に関しては、一致比較を行う必要があることになる。ここで、主記憶３における主記憶アドレスＡ２の値がＣａｃｈｅ７Ａを介して読み出され、ＲＢにおいて、Valueが主記憶３から読み出された値であり、かつ、Keyが「０３」となっているラインが検索される。図１４に示す例では、Keyが「０３」となっているラインは「０４」および「０５」の２つあるが、主記憶３から読み出された値が「００」であるので、「０５」のラインがマッチラインとして選択され、ＲＦに対して、エンコード結果としてキー０５が伝達される。

【０１７５】

以上のような処理が繰り返され、ＲＦにおいて、次に比較すべきレジスタ番号または主記憶アドレスがないことを示す終端フラグＥが検出された場合、入力パターンが全て一致したと判定され、該当命令区間は再利用可能と判断される。そして、終端フラグＥが検出されたラインから「Select Output」信号が出力され、ＲＯ１およびＲＯ２に格納されている、該ラインに対応する出力値がレジスタ６Ａおよび主記憶３に対して出力される。

【０１７６】

以上のように、第２の構成例による連想検索動作は、次のような特徴を有している。まず、内容が一致したことを示すマッチラインは、ＲＢにおいて１つのラインのみとなるので、検索動作を次列へ伝搬する際にエンコードした結果を１つ伝送すればよいことになる。したがって、ＲＢとＲＦとの間を接続する信号線は、アドレスのエンコード結果である１組（Ｎ本）でよいことになる。これに対して、上記した図１に示す例では、ＲＢにおいてマルチマッチが許容されているので、ＲＢにおける各列同士を接続する信号線は、各ラインごとに設ける（2^N本）必要があることになる。すなわち、第２の構成例によれば、ＲＦ／ＲＢ２を構成する連想検索メモリにおける信号線の数を大幅に低減することが可能となる。

【０１７７】

また、検索途中ではシングルマッチのみが許容されるようになっているので、比較すべき項目の比較順番は、木構造における参照順に限定されることになる。すなわち、レジスタ値とメモリ内容とは、参照順に混在させながら比較する必要がある。

【０１７８】

入力パターンは、各項目を参照すべきKeyという形でリンクさせることにより、木構造によってＲＢおよびＲＦに登録されている。また、入力パターンの項目は、終端フラグによってその終端が示されるようになっている。よって、入力パターンの項目数を可変とすることができるので、再利用表に登録すべき命令区間の状態に応じて、柔軟に入力パターンの項目数を設定することが可能となる。また、入力パターンの項目数が固定でないことによって、利用しない項目が無駄にメモリ領域を占有することがなくなるので、メモリ領域の利用効率を向上させることができる。

【０１７９】

また、木構造によって入力パターンが登録されるので、項目の内容が重複する部分については、複数の入力パターンで１つのラインを共有することが可能となっている。よって、メモリ領域の利用効率をさらに向上させることができる。

【０１８０】

なお、以上のような構成の場合、ＲＦおよびＲＢを構成するメモリとしては、構造が縦長のものとなる。例えばこのメモリ容量を２Ｍｂｙｔｅとした場合、横が８ｗｏｒｄ、縦を６５５３６ラインとすることになる。

【０１８１】

（連想検索動作の別の例）

上記の例では、図１３に示したＲＦにおいて、UP、Alt.、およびDNの項目は利用していないことになる。すなわち、上記の例では、ＲＦにおいて、これらの項目を設ける必要はないことになる。これに対して、UP、Alt.、およびDNの項目を利用することによって、連想検索動作をさらに高速化する構成および動作について以下に説明する。

【０１８２】
まず、図１５（ｂ）に、プログラムカウンタ（ＰＣ）およびレジスタの内容（Reg.）のみを比較し、これらが一致した場合は、主記憶値を比較することなく、区間の再利用が可能であると判断できる場合の状態を示す。この状態では、まず、ＲＢの「０１」のラインにおいて、ＰＣおよびReg.がValueに登録されており、ＲＦの「０１」のラインにおいて、終端フラグが「Ｅ」、比較要フラグが「０」、比較すべき主記憶アドレスが「Ａ１」、親ノード番号を示すUPが「ＦＦ」となっている。また、ＲＢの「０３」のラインでは、Value値なしで、Keyが「０１」となっており、ＲＦの「０３」のラインでは、終端フラグが「Ｅ」、比較要フラグが「０」、比較すべき主記憶アドレスが「Ａ２」、親ノード番号を示すUPが「ＦＦ」となっている。以降、同様に、ＲＢおよびＲＦにおける「０５」のラインおよび「０７」のラインが登録されており、それぞれ終端フラグが「Ｅ」、比較要フラグが「０」となっている。

【０１８３】
この状態で、ある命令区間の実行が検出されると、ＰＣおよびReg.がＲＢに入力され、マッチラインとして、ＲＢにおける「０１」のラインが選択される。そして、マッチラインとして選択されたラインのＲＢにおける番地である「０１」が、エンコード結果としてＲＦに伝達され、キー０１に対応するＲＦにおけるラインが参照される。キー０１に対応するＲＦにおけるラインでは、終端フラグが「Ｅ」となっているので、次に比較すべき主記憶アドレスがないことがわかる。また、比較要フラグ「０」となっているので、主記憶アドレスＡ１について比較を行う必要はないことがわかる。

【０１８４】
したがって、図１５（ａ）の木構造に示すように、ＰＣおよびReg.の一致がＳ１において確認されると、Ｔｒ１に示すノードのように、主記憶アドレスＡ１、Ａ２、Ａ３における比較を行うことなく、対応する出力値が出力されることになる。

【０１８５】
ＲＦおよびＲＢがこの状態である場合に、主記憶アドレスＡ２に対して書き込みが行われたとする。この場合、ＲＦおよびＲＢにおける入力パターンの登録時には主記憶アドレスＡ２の一致比較を行う必要はない状態であったが、主記憶アドレスＡ２が変更されることによって、主記憶アドレスＡ２の一致比較を行う必要が生じることになる。したがって、この場合には、図１６（ｂ）に示すようにＲＦおよびＲＢが変更されることになる。

【０１８６】
まず、内容が変更された主記憶アドレスであるＡ２をキーにして、ＲＦにおけるAdr.の列に対して検索がかけられる。これによって、ＲＦにおける「０３」のラインが選択される。そして、選択された「０３」のラインにおいて、比較要フラグが「１」に設定されるとともに、終端フラグ「Ｅ」が削除される。

【０１８７】
次に、「０３」のラインにおけるUPを参照することによって、親ノードとしての「０１」のラインが認識される。そして、「０１」のラインにおいて、次に比較すべき主記憶アドレスよりも優先して比較すべき主記憶アドレスを示すAlt.に、内容が変更された主記憶アドレスであるＡ２を書き込まれるとともに、終端フラグ「Ｅ」が削除される。さらに、「０１」のラインにおいて、優先して比較する際に必要なキーを示すDNに「０３」が書き込まれる。

【０１８８】
以上のようにＲＦおよびＲＢが書き換えられた場合の連想検索動作は次のようになる。ある命令区間が検出された際に、まず、ＰＣおよびReg.がＲＢに入力される。そして、ＲＢにおいて、連想検索により、入力されたこれらの値と、ＲＢのValueの列に登録されている命令区間先頭アドレスおよびレジスタ値とが比較され、ＲＢにおける「０１」のラインがマッチラインとして選択される。

【０１８９】
次に、マッチラインとして選択されたラインのＲＢにおける番地である「０１」が、エ

ンコード結果としてＲＦに伝達され、キー０１に対応するＲＦにおけるラインが参照される。キー０１に対応するＲＦにおけるラインでは、比較要フラグが「０」であり、比較すべき主記憶アドレスがＡ１となっている。すなわち、主記憶アドレスＡ１に関しては、一致比較を行う必要はないことがわかる。

【０１９０】

また、次に比較すべき主記憶アドレスよりも優先して比較すべき主記憶アドレスを示すAlt.に、主記憶アドレスＡ２が登録されており、優先して比較する際に必要なキーを示すDNに「０３」が登録されていることが確認される。この場合、主記憶３における主記憶アドレスＡ２の値がＣａｃｈｅ７Ａを介して読み出され、ＲＢにおいて、Valueが主記憶３から読み出された値であり、かつ、Keyが、DNに示されている「０３」となっているラインが検索される。

【０１９１】

図１６（ｂ）に示す例では、Keyが「０３」となっているラインは「０４」および「０５」の２つあるが、主記憶３から読み出された値が「００」であるので、「０５」のラインがマッチラインとして選択され、ＲＦに対して、エンコード結果としてキー０５が伝達される。キー０５に対応するＲＦにおけるラインでは、終端フラグが「Ｅ」となっているので、入力パターンが全て一致したと判定され、該当命令区間は再利用可能と判断される。そして、終端フラグＥが検出されたラインから「Select Output」信号が出力され、ＲＯ１およびＲＯ２に格納されている、該ラインに対応する出力値がレジスタ６Ａおよび主記憶３に対して出力される。

【０１９２】

以上のような連想検索動作によれば、ＲＦにおいて、次に比較すべき主記憶アドレスよりも優先して比較すべき主記憶アドレスを示すAlt.、および、優先して比較する際に必要なキーを示すDNが設けられているので、主記憶アドレスＡ１の内容とキー０１による検索をスキップして、主記憶アドレスＡ２の内容とキー０３による検索が可能となる。したがって、検索動作の処理ステップを低減することができるので、処理の高速化を図ることができる。

【０１９３】

（出力値の格納手段）

上記では、命令区間の入力パターンをＲＦおよびＲＢに登録し、連想検索動作を行うことについて説明したが、以下では、入力パターンの一致が確認された後に、再利用として出力される出力値を格納する手段について説明する。上記において図１３を参照しながら説明したように、命令区間記憶部２には、再利用が可能であると判定された場合に、主記憶および／またはレジスタに出力する出力値を格納する出力値格納手段として、ＲＯ１およびＲＯ２が設けられている。

【０１９４】

出力値は、ＲＦおよびＲＢから出力されるアドレスに基づいて、出力値を記憶するＲＡＭなどの記憶手段を参照することによって得ることが可能である。しかしながら、入力パターンと同様に、出力パターンについても、出力値の項目数を可変とすることが好ましいので、出力値の格納方法に関して工夫が必要である。

【０１９５】

入力パターンに関しては、ＲＦおよびＲＢにおいて木構造によって登録されている。そして、木構造の末端となっているライン、すなわち、終端フラグＥが登録されているラインにおいて、再利用が可能であると判定されることになる。したがって、終端フラグＥが登録されている各ラインに、出力すべき出力値を格納する出力値格納手段におけるポインタを登録しておくことによって、再利用の際の出力動作を行うことが可能となる。

【０１９６】

しかしながら、入力パターンが全て一致したことが確認された時点で、出力値が格納されているポインタに基づいて出力値格納手段における格納位置が特定される場合、ポインタに基づいて格納位置を特定するという変換処理が必要となり、処理速度を低下させる要

因となる。

【０１９７】

そこで、本実施形態では、出力値格納手段として、ＲＯ１およびＲＯ２の２つの記憶手段を設けている。そして、ＲＯ１は、ＲＦの各ラインに１対１で対応して出力値および出力すべきアドレスを格納している。すなわち、終端フラグＥが登録されているＲＦのラインにおいて再利用が可能であると判定された場合には、そのラインに対応するＲＯ１のラインが選択され、出力値が出力される。

【０１９８】

しかしながら、このように、出力値格納手段を、ＲＦの各ラインに１対１で対応して出力値および出力すべきアドレスを格納している場合、ＲＦにおける、終端フラグＥが登録されていないＲＦのラインに対しても、ＲＯ１においてメモリ領域が確保されることになる。また、終端フラグＥが登録されているＲＦの全てのラインに対応して、ＲＯ１において出力値を格納するので、同じ内容が複数箇所で記憶されている、というような冗長性が存在することになる。したがって、ＲＯ１は、高速に処理を行うという面では優れているが、メモリの利用効率としてはよくないことになる。

【０１９９】

この問題を解消するために、ＲＯ１に登録可能な項目数、すなわち出力値と出力アドレスとの組の数を少なめに設定する（図１３の例では２つ）とともに、ＲＯ１に登録しきれない出力値および出力アドレスの組については、ポインタを用いて格納領域が指示される構成のＲＯ２に登録するようにしている。

【０２００】

ＲＯ２においては、ポインタによって格納領域が指示されるので、使用されないメモリ領域はほとんど生じないことになる。また、複数の出力値および出力アドレスの組を登録する場合には、順次ポインタを用いてつなげていくことができるので、登録可能な出力値および出力アドレスの組の数を可変にすることが可能である。さらに、ＲＯ１における複数のラインから、ＲＯ２における同じ格納位置を示すポインタを指示することも可能となるので、ＲＯ２における格納情報を、ＲＯ１における複数のラインで共有することも可能となる。よって、ＲＯ２においては、格納内容の冗長性を低くすることができる。

【０２０１】

以上のように、出力値格納手段としてＲＯ１およびＲＯ２の２つを設けることによって、出力値の項目が少ない場合にはＲＯ１のみの利用により処理の高速性を実現するとともに、出力値の項目が多い場合には、項目の数を可変とすることが可能なＲＯ２を用いることによって対応している。よって、上記の構成によれば、処理の高速性とメモリ利用効率の向上とを実現することができる。

【０２０２】

（命令区間記憶部に対する登録処理）

上記では、ある命令区間の実行に際して再利用を行う場合の動作について説明した。以下では、ある命令区間の実行に際して、再利用が行えないと判断された場合に、該命令区間による入出力をＲＦ、ＲＢ、ＲＯ１、およびＲＯ２に登録する際の動作について説明する。

【０２０３】

まず、ある命令区間の実行が検出されると、ＰＣおよびReg.の値がＲＢに入力される。そして、ＲＢにおいて、連想検索により、入力されたこれらの値と、ＲＢのValueの列に登録されている命令区間先頭アドレスおよびレジスタ値とが比較される。ここで、ＲＢのValueの列に、入力された値と一致するものがないと判定された場合、該命令区間は、再利用が不可能であると判定され、演算器５Ａによる演算処理が行われる。そして、該当命令区間の演算処理が終了するまでに用いられるレジスタ入力値、主記憶入力値、主記憶出力値、およびレジスタ出力値が、ＲＢ、ＲＦ、ＲＯ１、必要に応じてＲＯ２に登録される。ここで、ＲＢおよびＲＦに登録を行う際には、上記で示したような木構造となるように、各項目が１つのラインに対応するように登録が行われる。そして、登録すべき入力パタ

ーンの最後の項目が登録されたラインにおいて、ＲＦの終端フラグを「Ｅ」とし、入力パターンの登録を終了する。

【０２０４】

一方、入力されたＰＣおよびReg.の値に一致するものが、ＲＢのValueの列に登録されている場合には、上記した連想検索動作と同様にして、次の一致比較すべき項目についての一致比較が行われる。このようにして、ＲＢおよびＲＦに登録されている入力パターンと、該当命令区間における入力パターンとの一致比較を継続していき、一致しない項目が生じた時点で、新たにノードを追加する形で、その一致しない項目についてＲＢおよびＲＦに登録が行われる。そして、登録すべき入力パターンの最後の項目が登録されたラインにおいて、ＲＦの終端フラグを「Ｅ」とし、入力パターンの登録を終了する。

【０２０５】

入力パターンの登録が終了すると、終端フラグを「Ｅ」としたＲＦにおけるラインに対応する、ＲＯ１におけるラインに、出力値および出力アドレスの登録を行う。そして、出力値として登録すべき項目がＲＯ１に登録しきれない場合には、ポインタを用いてＲＯ２に対して登録が行われる。以上により、命令区間の登録処理が完了する。

【０２０６】

（第２の構成例における予測機構）

第２の構成例では、命令区間の実行時における入出力パターンを一時的に格納する場所は、ＲＷ４Ａ・４Ｂとなる。ここで、前記した第１の構成例では、命令区間の実行時における入出力パターンはＲＢに直接登録されていたので、ＲＷ４Ａ・４ＢはＲＢの各行に対するポインタによって実現されていた。これに対して、第２の構成例では、ＲＦおよびＲＢが木構造によって構成されているので、ＲＷ４Ａ・４Ｂが直接ＲＢの行をポイントすることができない。すなわち、第２の構成例では、ＲＷ４Ａ・４Ｂは、ＲＢの各行に対するポインタとして機能するものではなく、命令区間の実行時における入出力パターンを一時的に格納する実質的なメモリとして機能することになる。

【０２０７】

また、図１３においては図示していないが、第２の構成例においても、所定の命令区間が繰り返し実行された場合における入力パターンの履歴エントリを格納する一時格納メモリ領域として、図１に示すようなＲＦおよびＲＢが設けられている。ただし、この場合には、ＲＢにおけるエントリの行は、履歴エントリを格納する履歴格納行としての数行によって構成されることになる。

【０２０８】

命令区間が実行されると、その入力要素がＲＷ４Ａ・４Ｂに順次格納され、全ての入力要素が揃い、演算が行われることによって出力要素が確定すると、この入出力パターンが、上記履歴格納行に格納されるとともに、上記のような木構造の入出力パターン格納機構に格納されることになる。

【０２０９】

また、所定の命令区間が繰り返し実行された場合には、履歴格納行に順次格納され、所定の数の履歴が格納された時点で、上記のように予測処理部２Ｂによって予測が行われ、予測に基づいてＳＳＰ１Ｂによって実行された結果は、上記のような木構造の入出力パターン格納機構に格納されることになる。

【０２１０】

（本発明の適用例）

「LIMIT」などによって大域変数領域とスタック領域とを区別できるプログラム実行環境があるとした上で、本発明に係るデータ処理装置を他の命令セットアーキテクチャにも適用するためには、スタックフレーム上の変数が、上位／下位関数のどちらの局所変数であるかを区別する手段が必要である。特に、引数を格納するレジスタが不足し、引数をスタックに格納する場合、呼ばれた関数側ではこの区別をすることができないことになる。

【０２１１】

本実施の形態で取り上げたＳＰＡＲＣプロセッサでは、引数の先頭６ワードを汎用レジ

スタに格納しており、６ワード以上の引数を扱う関数は出現頻度が高くないことと、引数がスタックに溢れた時点で再利用ができなくなることの両方を利用することによって、関数／ループの再利用を実現している。ＳＰＡＲＣプロセッサ同様に、３２本以上の汎用レジスタを有する多くのＲＩＳＣプロセッサでも、同様の判断をすることによって、本発明のような関数／ループの再利用を実現することが可能である。

【産業上の利用可能性】

【０２１２】

本発明に係るデータ処理装置は、上記したようにＳＰＡＲＣプロセッサに適用することが可能である。また、ＳＰＡＲＣプロセッサと同様に、３２本以上の汎用レジスタを有する多くのＲＩＳＣプロセッサにも適用することが可能である。また、このようなプロセッサを備えたゲーム機器、携帯型電話機、および情報家電などに適用することができる。

【図面の簡単な説明】

【０２１３】

【図１】本発明の一実施形態に係るデータ処理装置が備えるＲＦ／ＲＢによって実現される再利用表を示す図である。

【図２】上記データ処理装置の概略構成を示すブロック図である。

【図３】命令がデコードされた結果、関数呼び出し命令である場合の処理の流れを示すフローチャートである。

【図４】命令がデコードされた結果、関数復帰命令である場合の処理の流れを示すフローチャートである。

【図５】命令がデコードされた結果、後方分岐成立である場合の処理の流れを示すフローチャートである。

【図６】命令がデコードされた結果、後方分岐不成立である場合の処理の流れを示すフローチャートである。

【図７】関数およびループが入れ子構造となっている状態の一例を示す図である。

【図８】関数の入れ子構造において、内側の構造のレジスタ入出力が、外側の構造のレジスタ入出力となる影響範囲を示す図である。

【図９】ＲＷと、ＲＦ・ＲＢとの関係を示す図である。

【図１０】ある命令区間が実行された場合のＲＢにおける実際の登録状況を示す図である。

【図１１】ある命令区間が繰り返し実行された場合における、履歴としてＲＢに登録された例を示す図である。

【図１２】予測に基づいて、予測処理部がアドレスＡ２およびアドレスＲ５の値に関して予測を行った場合の、予測エントリとしてＲＢに記録される入力要素の状態を示す図である。

【図１３】ＲＦ／ＲＢの第２の構成例の概略を示す図である。

【図１４】図１３に示すＲＦ／ＲＢにおける連想検索動作の具体例を示す図である。

【図１５】同図（ｂ）は、図１３に示すＲＦ／ＲＢにおける連想検索動作の他の具体例を示す図であり、同図（ａ）は、同図（ｂ）における連想検索動作を木構造として示す図である。

【図１６】同図（ｂ）は、図１３に示すＲＦ／ＲＢにおける連想検索動作のさらに他の具体例を示す図であり、同図（ａ）は、同図（ｂ）における連想検索動作を木構造として示す図である。

【図１７】同図（ａ）は、関数Ａが関数Ｂを呼び出す構造を概念的に示す概念図であり、同図（ｂ）は、同図（ａ）に示すプログラム構造を実行する際の主記憶におけるメモリマップを示す図である。

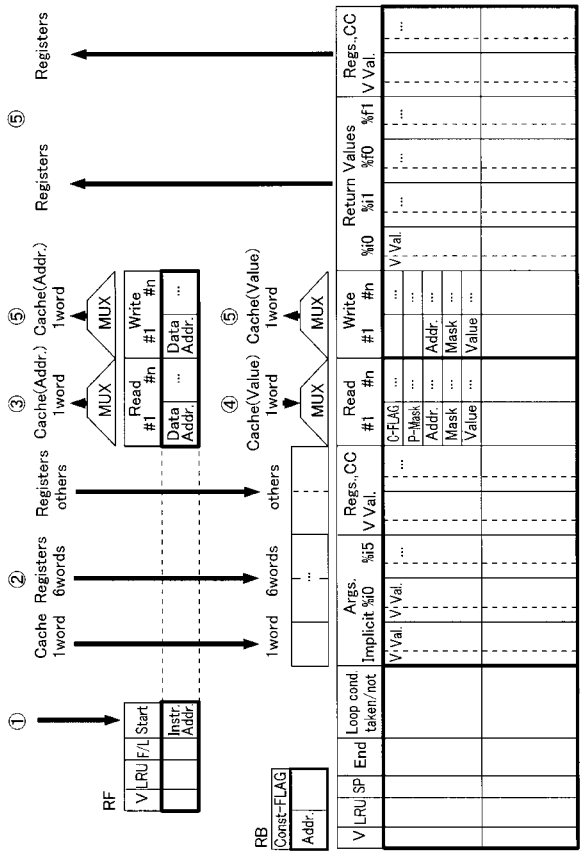
【図１８】関数Ａが関数Ｂを呼び出す場合の、メモリマップにおける引数およびフレームの概要を示す図である。

【図１９】１つの関数を再利用するための従来の再利用表を示す図である。

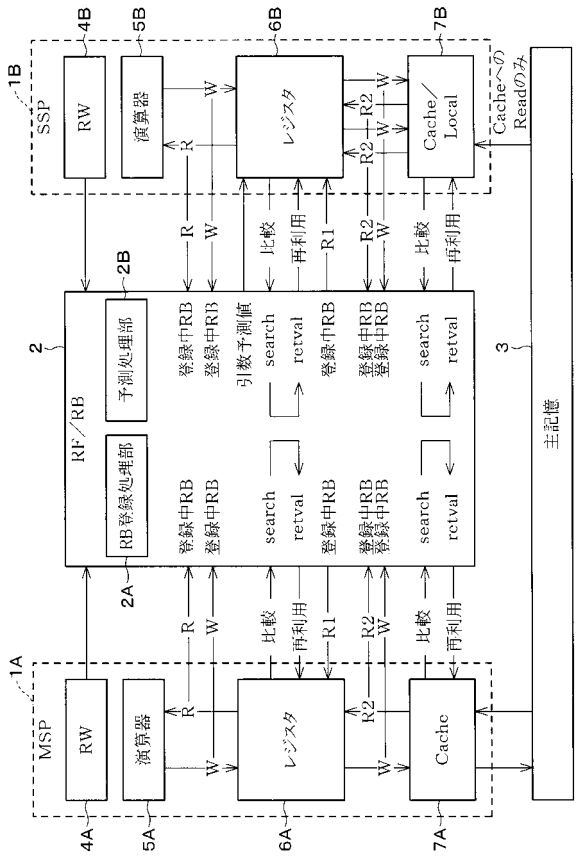
【図２０】命令区間の一例を示す図である。

【図２１】図２０に示す命令区間が実行された場合に、ＲＢに登録される入力アドレスおよび入力データ、並びに出力アドレスおよび出力データを簡略化して示す図である。
【図２２】ＲＢにおける実際の登録状況を示す図である。
【図２３】図２０に示す命令区間が繰り返し実行された場合における、ＲＢの入力側に登録される履歴の例を示す図である。
【図２４】従来の入力予測による予測結果を示す図である。
【符号の説明】
【０２１４】
１Ａ　　ＭＳＰ
１Ｂ　　ＳＳＰ
２　　ＲＦ／ＲＢ（入出力記憶手段）
２Ａ　　ＲＢ登録処理部（区別処理手段）
２Ｂ　　予測処理部（予測処理手段）
３　　主記憶（主記憶手段）
４Ａ・４Ｂ　　ＲＷ
５Ａ・５Ｂ　　演算器（第１・第２の演算手段）
６Ａ・６Ｂ　　レジスタ
７Ａ・７Ｂ　　Ｃａｃｈｅ

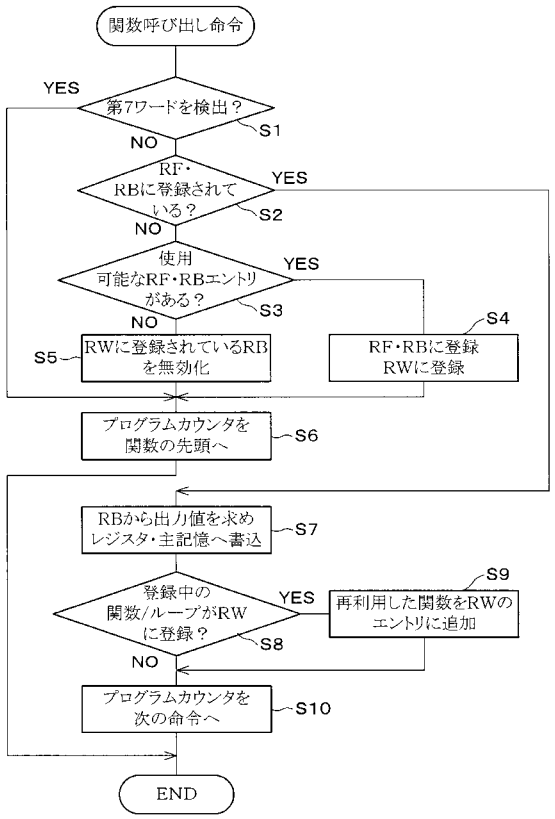
【図１】



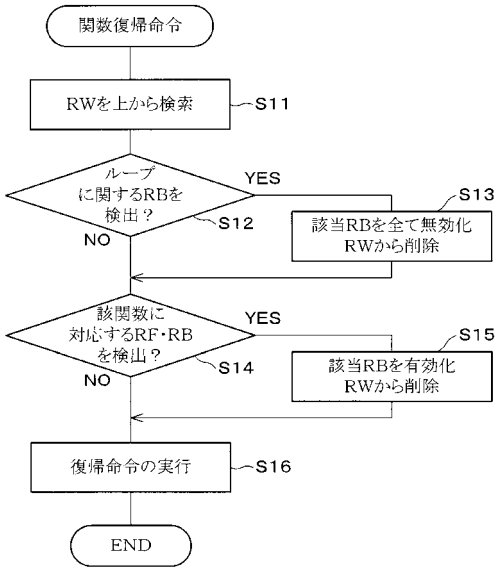
【図２】



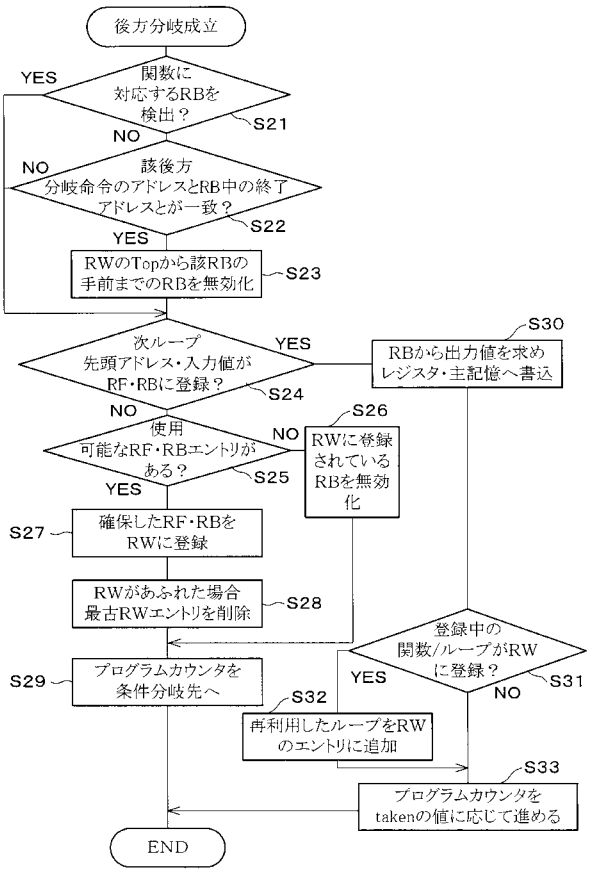
【図３】


関数呼び出し命令
第7ワードを検出？
YES
NO
S1
RF・RBに登録されている？
YES
NO
S2
使用可能なRF・RBエントリがある？
YES
NO
S3
S4
S5
RWに登録されているRBを無効化
RF・RBに登録 RWに登録
プログラムカウンタを関数の先頭へ
S6
RBから出力値を求めレジスタ・主記憶へ書込
S7
登録中の関数/ループがRWに登録？
YES
NO
S8
S9
再利用した関数をRWのエントリに追加
プログラムカウンタを次の命令へ
S10
END

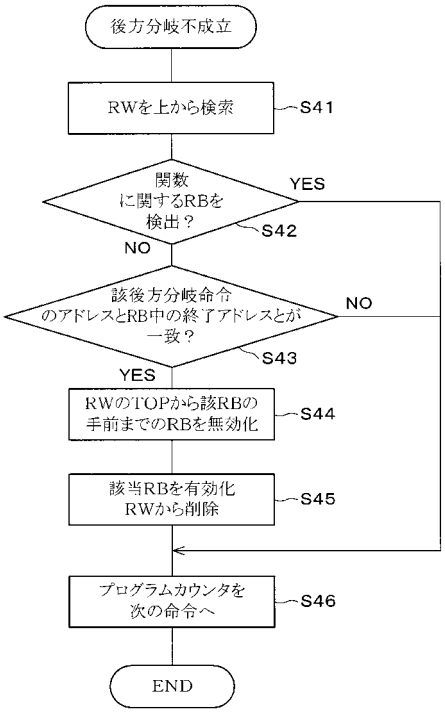
【図４】


関数復帰命令
RWを上から検索
S11
ループに関するRBを検出？
YES
NO
S12
S13
該当RBを全て無効化 RWから削除
該関数に対応するRF・RBを検出？
YES
NO
S14
S15
該当RBを有効化 RWから削除
復帰命令の実行
S16
END

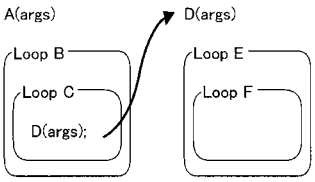
【図５】


後方分岐成立
関数に対応するRBを検出？
YES
NO
S21
該後方分岐命令のアドレスとRB中の終了アドレスとが一致？
NO
YES
S22
RWのTopから該RBの手前までのRBを無効化
S23
次ループ先頭アドレス・入力値がRF・RBに登録？
YES
NO
S24
S30
RBから出力値を求めレジスタ・主記憶へ書込
使用可能なRF・RBエントリがある？
NO
YES
S25
S26
RWに登録されているRBを無効化
S27
確保したRF・RBをRWに登録
RWがあふれた場合最古RWエントリを削除
S28
S29
プログラムカウンタを条件分岐先へ
登録中の関数/ループがRWに登録？
YES
NO
S31
S32
再利用したループをRWのエントリに追加
S33
プログラムカウンタをtakenの値に応じて進める
END

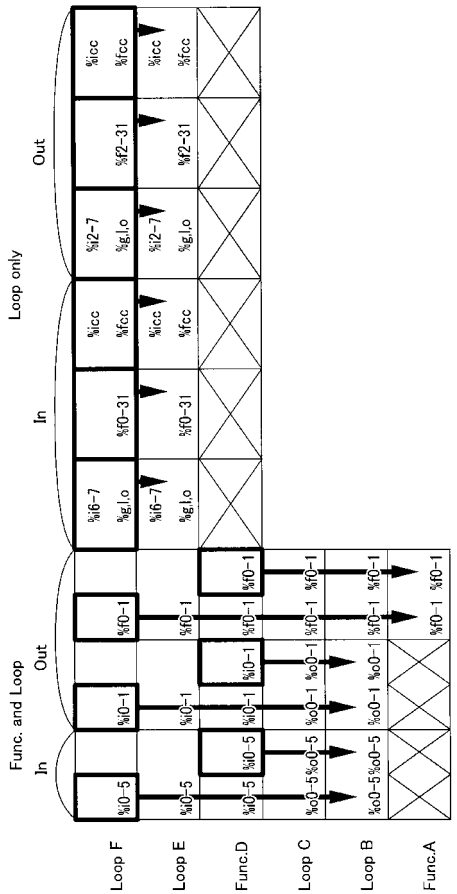
【図６】


後方分岐不成立
RWを上から検索
S41
関数に関するRBを検出？
YES
NO
S42
該後方分岐命令のアドレスとRB中の終了アドレスとが一致？
NO
YES
S43
RWのTOPから該RBの手前までのRBを無効化
S44
該当RBを有効化 RWから削除
S45
プログラムカウンタを次の命令へ
S46
END

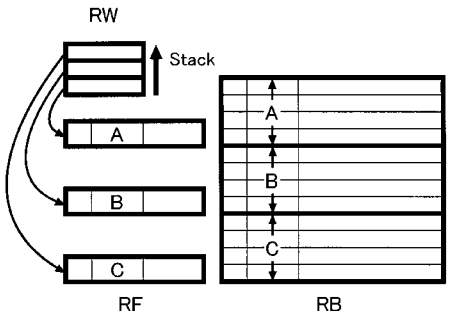
【図７】


A(args)
D(args)
Loop B
Loop C
D(args);
Loop E
Loop F

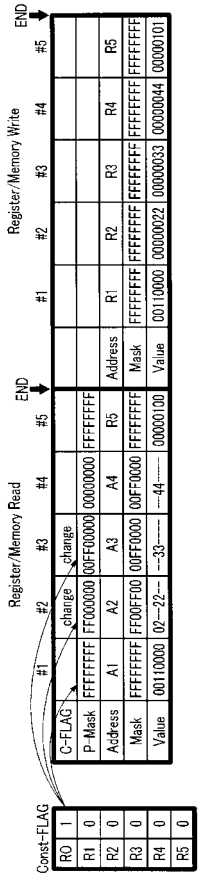
【図８】


Func. and Loop
Loop only
In
Out
%i0-5
%i0-1
%f0-1
%i6-7
%f0-31
%i2-7
%f2-31
%g,l,o
%icc
%fcc
Loop F
Loop E
Func.D
Loop C
Loop B
Func.A

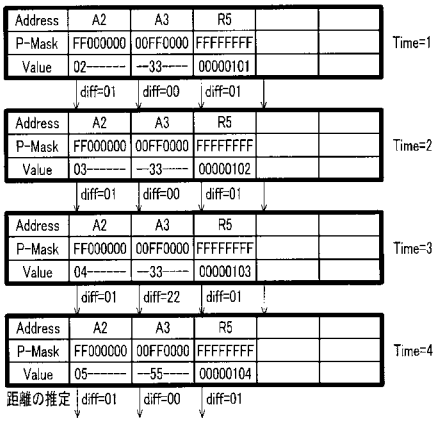
【図９】


RW
Stack
A
B
C
RF
RB

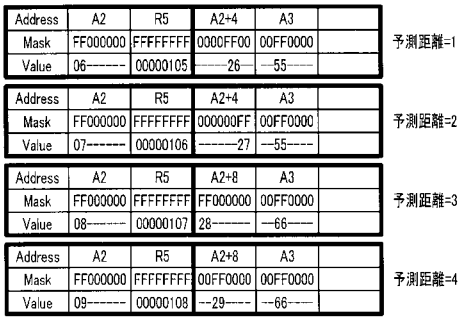
【図１０】


Register/Memory Read
Register/Memory Write
END
C-FLAG
P-Mask
Address
Mask
Value
change
Const-FLAG
R0
R1
R2
R3
R4
R5

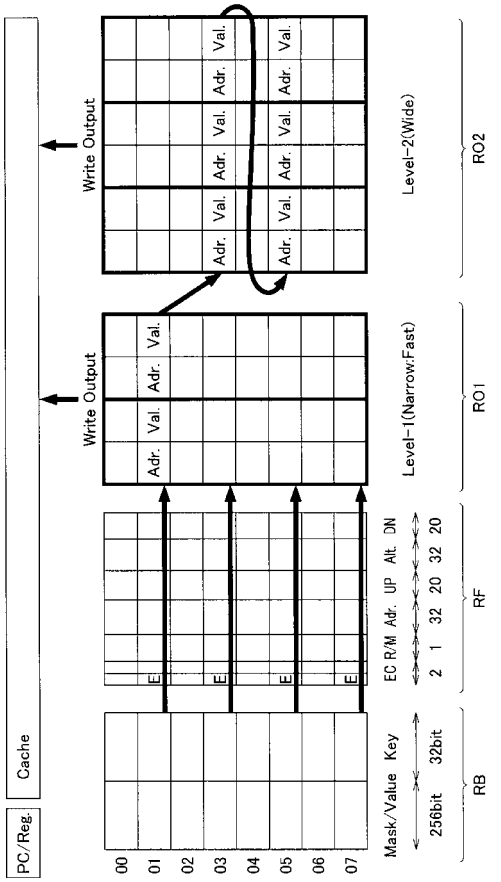
【図１１】



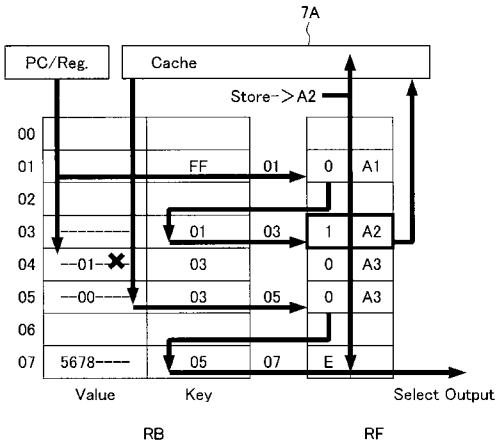
【図１２】



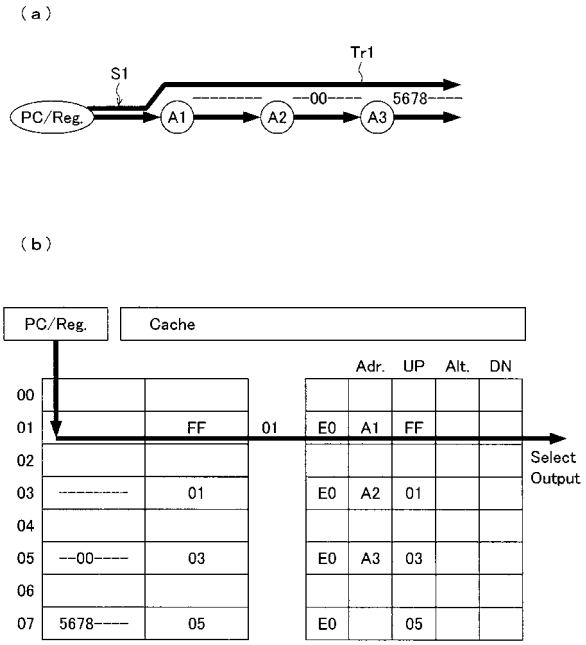
【図１３】



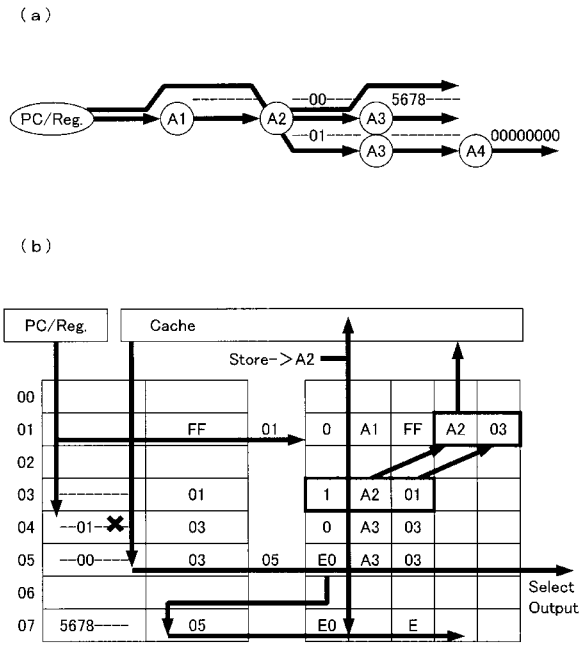
【図１４】



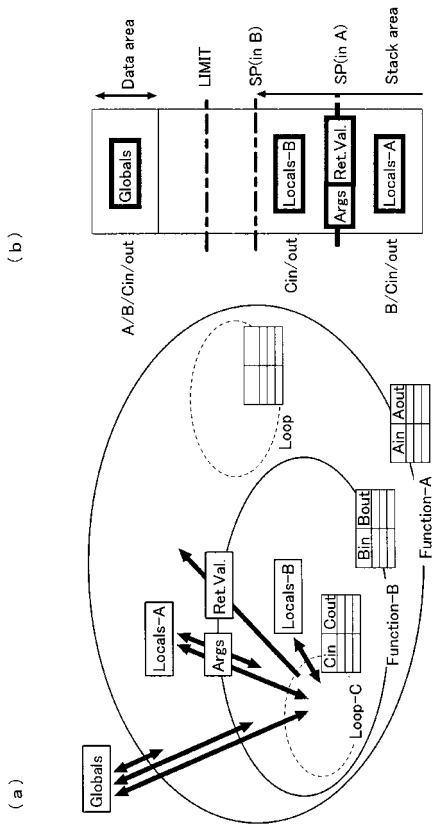
【図１５】



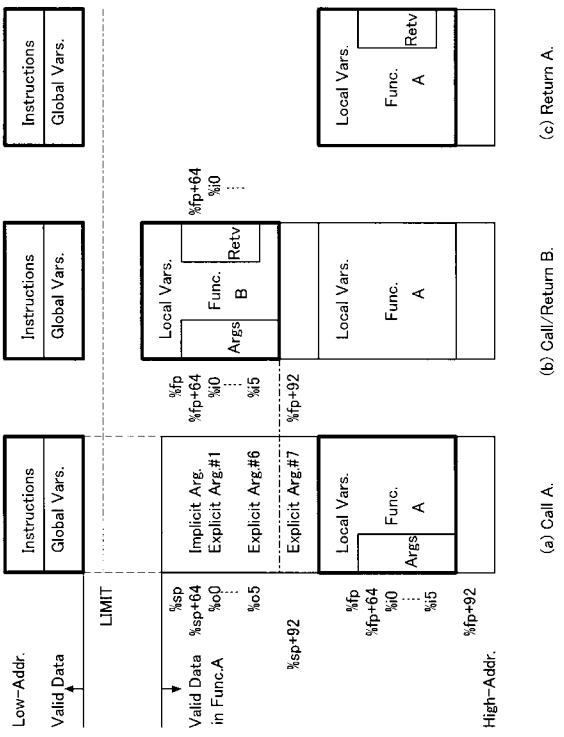
【図１６】



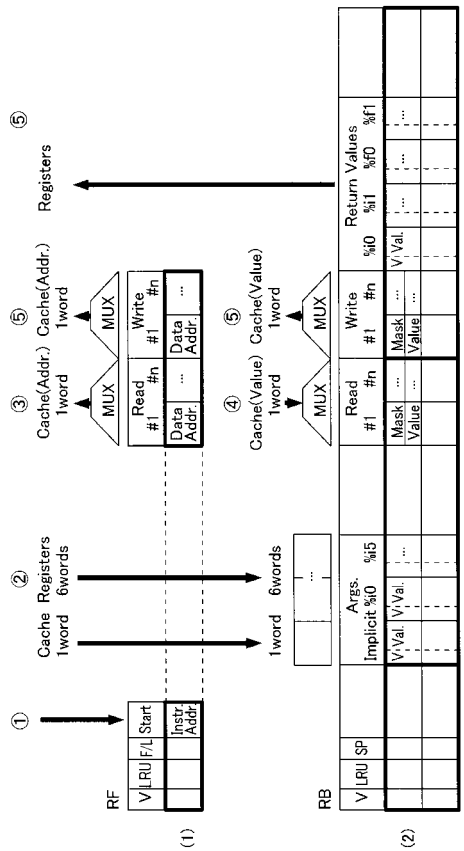
【図１７】



【図１８】



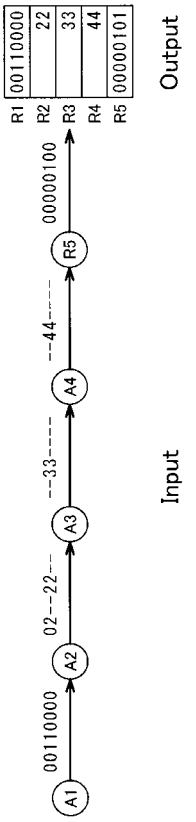
【図１９】



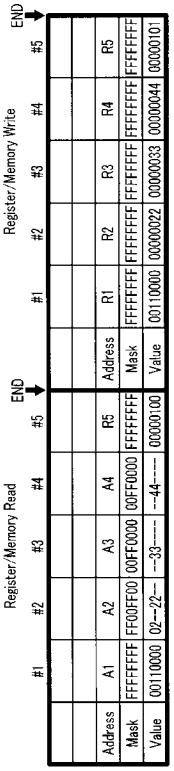
【図２０】

```
PC:1000
 set A1->R0
 ld (R0)->R1
 set A2->R0
 ldb (R0)->R2
 ldb (A2+R2)->R2
 set A3->R0
 ldb (R0)->R3
 ldb (A4=R1+R2)->R4
 add R5+1->R5
```

【図２１】



【図２２】



【図２３】

Address	A1	A2	A3	A4	R5	
Mask	FFFFFFFF	FF00FF00	00FF0000	00FF0000	FFFFFFFF	Time=1
Value	00110000	02---22---	--33----	--44----	00000101	
	diff=00	diff=??	diff=00	diff=00	diff=01	
Address	A1	A2	A3	A4	R5	
Mask	FFFFFFFF	FF0000FF	00FF0000	00FF0000	FFFFFFFF	Time=2
Value	00110000	03------23	--33----	--44----	00000102	
	diff=00	diff=??	diff=??	diff=11	diff=??	
Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	FF000000	00FF0000	00FF0000	Time=3
Value	00110000	04-------	24-------	--33----	--44----	
	diff=00	diff=01	diff=??	diff=22	diff=00	
Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	00FF0000	00FF0000	00FF0000	Time=4
Value	00110000	05-------	--25----	--55----	--44----	
距離の推定	diff=00	diff=01	diff=00	diff=22	diff=00	

【図２４】

Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	00FF0000	00FF0000	00FF0000	予測距離=1
Value	00110000	06-------	--25----	--77----	--44----	
Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	00FF0000	00FF0000	00FF0000	予測距離=2
Value	00110000	07-------	--25----	--99----	--44----	
Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	00FF0000	00FF0000	00FF0000	予測距離=3
Value	00110000	08-------	--25----	--BB----	--44----	
Address	A1	A2	A2+4	A3	A4	
Mask	FFFFFFFF	FF000000	00FF0000	00FF0000	00FF0000	予測距離=4
Value	00110000	09-------	--25----	--DD----	--44----	