(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2016-6632
(P2016-6632A)
(43)公開日　平成28年1月14日(2016.1.14)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	***9/32***	***(2006.01)***	G06F	9/32	320F	
G06F	***9/318***	***(2006.01)***	G06F	9/30	320B	

審査請求 未請求　請求項の数 15　ＯＬ　外国語出願　（全 19 頁）

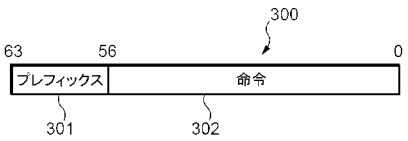
(21)出願番号　特願2015-84940(P2015-84940)
(22)出願日　平成27年4月17日(2015.4.17)
(31)優先権主張番号　1454511
(32)優先日　平成26年5月20日(2014.5.20)
(33)優先権主張国　フランス(FR)

(71)出願人　512033279
ブル・エス・アー・エス
フランス国、エフ－７８３４０・レ・クレイ・スー・ボア、リユ・ジヤン・ジヨレ
(74)代理人　100126572
弁理士　村越　智史
(74)代理人　100125195
弁理士　尾畑　雄一
(72)発明者　シェハイバー，ガッサーン
フランス国　ギュイヤンクール　７８２８０　サン・ポル・ルー通り　１１

(54)【発明の名称】条件付き命令を有するプロセッサ

(57)【要約】　（修正有）
【課題】プロセッサによって機械命令を処理する方法を提供する。
【解決手段】機械命令は、実行する少なくとも１つの第１演算の識別子と、少なくとも１つの第１演算を実行するために検証する条件を表す条件プレフィックス、とを含む。方法は、機械命令を受信する工程と、プレフィックスを評価する工程と、条件が実証されたか否かに従って、機械命令の中で識別された少なくとも１つの第１演算を実行する又は実行しない工程と、を備える。
【選択図】図３Ａ

Fig. 3A



【特許請求の範囲】
【請求項１】
プロセッサによって機械命令を処理する方法であって、
実行する機械命令（３００）を受信する工程（１０４、１０５）であって、前記機械命令は実行する少なくとも１つの第１演算（３０２）の識別子と、前記少なくとも１つの第１演算を実行するために検証する条件を表す条件プレフィックス（３０１）とを含む、工程と、
前記プレフィックスを評価する工程（１０８）と、
前記条件が実証されたか否かに従って、前記機械命令の中で識別された前記少なくとも１つの第１演算を実行する（又は実行しない）工程（１０９）と、
を備える方法。
【請求項２】
前記プレフィックスを評価する工程は述語レジスタの値を検証する工程を含む、請求項１に記載の方法。
【請求項３】
前記条件プレフィックスが、
前記述語レジスタの前記値の識別子と、
前記検証のために前記値に対して実行する第２演算の識別子と、
を有する、請求項２に記載の方法。
【請求項４】
前記第２演算は論理演算である、請求項３に記載の方法。
【請求項５】
前記第２演算は前記値を実現するための期待値である、請求項３に記載の方法。
【請求項６】
前記第２演算は内容のない演算であり、よって前記条件は常に実証される、請求項３に記載の方法。
【請求項７】
前記少なくとも１つの第１演算は、前記プロセッサによって実行するコードの他の機械命令への分岐処理である、請求項１ないし６のうちのいずれか１項に記載の方法。
【請求項８】
前記分岐処理が述語ベクトルの前記評価によって条件付けられ、当該述語ベクトルは一又は複数の述語レジスタの複数の値を含む、請求項７に記載の方法。
【請求項９】
前記述語ベクトルは部分的に評価される、請求項８に記載の方法。
【請求項１０】
前記少なくとも１つの第１演算は述語計算を表し、当該計算は、
計算レジスタの１ビットを決定する工程と、
前記決定されたビットを所定の値と比較する工程と、
前記比較の結果を述語レジスタ内に書き込む工程と、
を有する、請求項１ないし９のうちのいずれか１項に記載の方法。
【請求項１１】
前記機械命令は並行して実行する２つの第１演算の識別子を含む、請求項１ないし１０のうちのいずれか１項に記載の方法。
【請求項１２】
前記プロセッサによる機械命令の実行は処理連鎖モジュールによって管理され、前記機械命令は前記処理連鎖の前記実行ステージにおいて実行される、請求項１ないし１１のうちのいずれか１項に記載の方法。
【請求項１３】
各機械命令は１つのプロセッササイクルにおいて実行される、請求項１２に記載の方法。

【請求項１４】
前記機械命令は書式が事前定義されたコード語によって表される、請求項１ないし１３のうちのいずれか１項に記載の方法。
【請求項１５】
実行する少なくとも１つの機械命令を記憶するように構成されたメモリ（１００）であって、前記機械命令は実行する少なくとも１つの第１演算の識別子と、前記少なくとも１つの第１演算を実行するために検証する条件を表す条件プレフィックスとを含む、メモリ（１００）と、
前記プレフィックスを評価し、前記条件が実証されたか否かに従って、前記機械命令内で識別される前記少なくとも１つの第１演算を実行させる又は実行させないように構成された管理モジュール（１０１）と、
前記識別された少なくとも１つの第１演算を実行するように構成された処理装置（１０２）と、
を備えるプロセッサ。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は計算装置のプロセッサの分野に関する。特に、リスト中の項目検索を実装するための専用プロセッサに関する。
【背景技術】
【０００２】
ＭＰＩ（メッセージ・パッシング・インターフェース）に対応するネットワーク・インターフェース・カードなどの、ある種の計算装置はリスト中の項目検索の大規模な実装を必要とする。
【０００３】
この検索の処理には、例えば、通信ネットワークノードが予想するメッセージのリストを、各々の記憶スペースの表示とともに作成する処理や、当該ノードが受信するすべてのメッセージを上記リスト内のメッセージと比較する処理が含まれる。よって、メッセージが届くと、当該メッセージの記憶スペースに送信して処理することができる。
【０００４】
伝統的に、受信したメッセージは各々がラベルを有しており、これをリスト中のメッセージのラベルと比較しなければならない。リスト中のメッセージのラベルにマスキングを行い、より少ない数のビットでラベルの比較を実行してもよい。
【０００５】
メッセージが届くと、そのラベルはリスト中の最初の項目のラベルと比較され、続いて第２、第３の項目というように、一致するラベルが見つかるまで比較が繰り返される。
【０００６】
このとき、受信したメッセージは記憶スペースに送信され、リスト中の一致した項目は削除される。その後リストは更新される。
【０００７】
従って、予想されるメッセージのリストは、（対応するメッセージを受信したときに）除去される項目又は（新しいメッセージを求めるときに）追加される項目によって動的に変更されるリストである。
【０００８】
この種の検索の実装には、複雑なパス及びリスト管理アルゴリズムの実行が必要である。さらに、これらのアルゴリズムは通常管理対象となる多数のオプションとともに実装される。
【０００９】
その結果、ＭＰＩ型のインターフェースをはじめとする計算装置において、この種の演算の専用プロセッサが必要となる。専用のプロセッサを用いれば、リスト中の項目の検索

（又はマッチングとも称する）はハードウェアではなくソフトウェアを用いて管理できる。これによって、プロセッサに指示を出す計算機コード（マイクロコード又はファームウェアともいう）を改善してインターフェースの仕様の変更を反映することができる。

【００１０】

プロセッサの最高の性能を得るためには、その実行時間、よって演算サイクルを短縮する必要がある。プロセッサ内の処理実行時間は、インターフェースによって管理されるメッセージのフローに影響する。

【００１１】

また、開発者によるファームウェアの記述も容易になる。ファームウェアはアセンブリ言語で記述されており、よって他の種類の言語によって提供される高レベルの制御構造を経ていない。エラーを記述したアセンブリコードはプロセッサに対する深刻で直接的な結果をもたらし、このエラーを制御することはできない。

【００１２】

また、プロセッサによって実行される機械命令を合理的なサイズに保つことが望ましい。

【００１３】

Ｈｅｍｍｅｒｔらによる文献”Ａｎ　ａｒｃｈｉｔｅｃｔｕｒｅ　ｔｏ　ｐｅｒｆｏｒｍ　ＮＩＣ　Ｂａｓｅｄ　ＭＰＩ　Ｍａｔｃｈｉｎｇ”には、実行される機械命令のフローを制御するための述語に基づいたプロセッサが開示されている。機械命令は、（ビット単位の）比較結果の（ＡＮＤ－ＯＲ型の）論理的な組合せを記憶する述語レジスタに記憶された値に従って実行される。述語レジスタは実行される命令が満たすべき条件を表している。

【００１４】

この文献において、述語レジスタの１ビットの値に従う分岐命令によってフローが制御される。分岐とは、コード中の次の命令を実行するのではなく、コード中の前又は後の命令に直接遷移することによって、一連の命令の一部を実行しないことである、と理解されている。よって分岐は、計算機コード内で前方又は後方に実行することができる。

【００１５】

命令から実行オプションを抽出するためには、三項比較装置（ＮＡＬＵ）によって比較が行われる。三項比較装置は２つの値を比較マスクと比較する。

【００１６】

しかしながら、この種のプロセッサには多くの欠点がある。

【００１７】

例えば、コードを実行するために必要なサイクル数が多い。これは主に制御の手段として分岐を広汎に用いているからである。この文献では、分岐を行うのに多数の２サイクルを必要とする。しかしながら、この場合は、（例えばＥＣＣ型の）エラー修正コードを用いずに単一のサイクルでメモリにアクセスできる試作プロセッサである。このようなプロセッサは現実的には産業用途に用いることはできない。産業用途においては、分岐を実行するためには通常多数の５サイクルが必要である。

【００１８】

さらに、図示されたプロセッサは典型的な演算装置（ＡＬＵ）及び三項演算装置（ＴＡＬＵ）を用いる。従って、同時計算を行うことはできない。なお、同時計算では命令サイズを最適化しないが、命令サイズは通常同時計算を実行可能な１６４ビットである。

【００１９】

よって、リスト中の項目をマッチングするための検索専用のプロセッサをはじめとする、従来技術のプロセッサを改善する必要がある。

【発明の概要】

【００２０】

この発明もそうしたプロセッサの１つである。

【００２１】

本発明の第１の態様は、プロセッサによって実行される機械命令の方法に関し、この方法は、実行する機械命令を受信する工程であって、前記機械命令は実行する少なくとも１つの第１演算の識別子と、前記少なくとも１つの第１演算を実行するために検証する条件を表す条件プレフィックスとを含む、工程と、前記プレフィックスを評価する工程と、前記条件が実証されたか否かに従って、前記機械命令の中で識別された前記少なくとも１つの第１演算を実行する（又は実行しない）工程と、を備える。

【００２２】

第１の態様に従う方法は命令実行の制御を最適化する。分岐処理のためのリソースを低減することは特に有益である。

【００２３】

第１の態様に従うプロセスは、命令を実行するために必要なサイクルの数を低減ことによって処理を高速化する。

【００２４】

例えば、第１の態様に従うプロセスは、項目をマッチングするための検索を、コンピュータシステム内のより少ない数のサイクルで実行することを可能にする。

【００２５】

現実化（ｒｅａｌｉｚａｔｉｏｎ）の方法によれば、前記プレフィックスを評価する工程は述語レジスタの値を検証する工程を含む。

【００２６】

前記条件プレフィックスは、前記述語レジスタの値の識別子と、前記検証のために前記値に対して実行する第２演算の識別子と、を含む。

【００２７】

例えば、前記第２演算は論理演算である。

【００２８】

例えば、前記演算は前記値を実現するための期待値である。

【００２９】

前記第２演算は内容のない演算であってもよく、よって前記条件は常に実証される。

【００３０】

現実化の方法によれば、前記少なくとも１つの第１演算は、前記プロセッサによって実行するコードの他の機械命令への分岐処理である。

【００３１】

例えば、前記分岐処理が述語ベクトルの前記評価によって条件付けられ、当該述語ベクトルは一又は複数の述語レジスタの複数の値を含む。

【００３２】

前記述語ベクトルは部分的に評価することもできる。

【００３３】

現実化の方法によれば、前記少なくとも１つの第１演算は述語計算を表し、当該計算は、計算レジスタの１ビットを決定する工程と、前記決定されたビットを所定の値と比較する工程と、前記比較の結果を述語レジスタ内に書き込む工程と、を含む。

【００３４】

例えば、前記機会命令は並行して実行する２つの第１演算の識別子を含む。

【００３５】

例えば、前記プロセッサによる機械命令の実行は処理連鎖モジュールによって管理され、前記機械命令は前記処理連鎖の前記実行ステージにおいて実行される。

【００３６】

各機械命令は１つのプロセッササイクルで実行できる。

【００３７】

前記機械命令は書式が事前定義されたコード語によって表される。

【００３８】

本発明の第２の態様は、本発明の第１の態様に従う方法を実装するように構成されたプ

ロセッサに関する。
【００３９】
例えば、このようなプロセッサは、実行する少なくとも１つの機械命令を記憶するように構成されたメモリであって、前記機械命令は実行する少なくとも１つの第１演算の識別子と、前記少なくとも１つの第１演算を実行するために検証する条件を表す条件プレフィックスとを含む、メモリと、前記プレフィックスを評価し、前記条件が実証されたか否かに従って、前記機械命令内で識別される前記少なくとも１つの第１演算を実行する（又は実行しない）ように構成された管理モジュールと、前記識別された少なくとも１つの第１演算を実行するように構成された処理装置と、を含む。
【００４０】
本発明の第３の態様は、リスト中の項目をマッチングするための検索装置であり、この検索装置は第２の態様に従うプロセッサを含む。例えば、この装置は連想リスト処理装置、すなわちＡＬＰＵである。
【００４１】
本発明の他の特性及び利点は、非限定的な実施例及び添付の図面として提供される以下の詳細な説明を通して明らかになるであろう。
【図面の簡単な説明】
【００４２】
【図１Ａ】現実化の方法に従うプロセッサを模式的に示す。
【００４３】
【図１Ｂ】現実化の方法に従う命令フローを処理する工程を示す。
【００４４】
【図２】分岐処理を示す。
【００４５】
【図３Ａ】現実化の方法に従う命令を示す。
【図３Ｂ】現実化の方法に従う命令を示す。
【図４Ａ】現実化の方法に従う命令を示す。
【図４Ｂ】現実化の方法に従う命令を示す。
【図４Ｃ】現実化の方法に従う命令を示す。
【図４Ｄ】現実化の方法に従う命令を示す。
【図４Ｅ】現実化の方法に従う命令を示す。
【図４Ｆ】現実化の方法に従う命令を示す。
【図４Ｇ】現実化の方法に従う命令を示す。
【発明を実施するための形態】
【００４６】
以下、現実化の方法に従うプロセッサアーキテクチャを説明する。図１Ａは大幅に簡略化した図である。この図は説明の後に述べる構成要素を提示することを目的としている。当業者であれば、他の構成要素がこのプロセッサの完全な動作に有用であることを理解できるであろう。これらの構成要素は簡潔さを得るために提示されるわけではない。
【００４７】
メモリ１００はプロセッサが実行する機械命令を記憶している。
【００４８】
これらの命令の実行の管理は処理連鎖モジュール１０１（又はパイプライン）によって行われる。
【００４９】
命令の実行に必要な演算の実行はＡＬＵ（算術論理演算装置）型の処理装置１０２によって行われる。
【００５０】
命令の実行に有用なデータは、処理装置による読込み及び／又は書込みの際に利用可能な一又は複数のレジスタ１０３に記憶される。

【００５１】
処理連鎖モジュールは命令フローを実行の７つの「ステージ」（又はステップ）で管理し、これらは図１Ｂに示される。
１）１０４では実行する命令のアドレスを命令メモリに送信する。
２）１０５はメモリ内の命令の読込みレイテンシを考慮した無処理のクロックサイクルである。
３）１０６では受信した命令の検証を行う。これは例えば、エラー修正コード（ＥＣＣ）の専用ビットを検証することによって行う。
４）１０７では命令の復号化を行う。
５）１０８では命令オペランドの読込みを行う。
６）１０９では演算を実行する。
７）１１０では命令の結果を一又は複数のレジスタに書き込む。
【００５２】
図１Ａに戻ると、処理連鎖モジュールによって命令の実行を制御するために、プロセッサは一又は複数の述語レジスタ１１１を備える。このレジスタ内のビット値によって、所定の命令が実行可能又は実行不能になる。
【００５３】
さらに、プロセッサによって実行されるコードの命令における分岐が、専用の分岐装置１１２によって発生する。
【００５４】
従来技術においては、分岐処理は命令の実行フローを制御するために一般的に用いられる技術である。しかしながら、分岐処理はプロセッサのサイクルの大きな部分を占有してしまう。ここでは分岐処理の使用を減らすことを提案する。
【００５５】
以下ではまず図２を参照しつつ分岐処理について述べる。
【００５６】
コード中のＡからＺで示す命令のセットが順番に実行されるとする。つまり、このコードは命令Ａ、命令Ｂ、命令Ｃ・・・という順で構成される。また、このコードの実行中、ある条件で特定の命令が実行される。
【００５７】
例えば、ＣＯＮＤ条件が満たされた場合、命令Ｄの直後の命令Ｅは実行されず、代わりに命令Ｏが実行される。よってＤからＮまでの命令の実行は「ジャンプ」される。つまり、コードは命令Ｏに「分岐」する。
【００５８】
よって分岐処理により、実行のステージ６（図１Ｂのステップ１０９）において命令Ｅに進まないことになる。しかしながら、命令Ｏのエントリのために、処理連鎖中の命令Ｅの後のすべての命令を無駄にしてしまう。
【００５９】
図２において、処理連鎖中の各ステージの内容は、Ｔ１からＴ１６の異なるサイクルとなるように示される。
【００６０】
第１のサイクルＴ１において、処理連鎖モジュールによって命令Ａが要求される。サイクルＴ２において、命令メモリモジュールからの応答があるものとする。よって、命令Ａはステージ１からステージ２に進み、ステージ１は空きとなる。サイクルＴ２において、命令メモリモジュールに対して命令Ｂが要求される。
【００６１】
サイクルＴ３において、命令Ａが受信され、ステージ３に進んで検証が行われる。命令Ｂはステージ２で待機状態となり、ステージ１が空きとなって命令Ｃが要求される。
【００６２】
各サイクルで命令は処理連鎖モジュールにエントリし、各ステージを連続的に通過する

。
【００６３】
サイクルＴ７において、処理連鎖の全体が満たされる。サイクルＴ８において、命令Ａは処理連鎖から離脱する。
【００６４】
サイクルＴ９において、例えば、述語レジスタ１１１内の分岐モジュール１１２による条件の検証の後、分岐処理が要求される。このように、例えば、命令Ｅは実行されず、コードは命令Ｏから続行される。
【００６５】
よって命令Ｅはステージ６に送られず、サイクル１０において一般的に「ＮＯＰ」と示される内容のない命令と置き換えられる。さらに、ステージ２、３、４、及び５に進むはずの命令Ｉ、Ｈ、Ｇ、及びＦが内容のない命令と置き換えられる。サイクルＴ１０においてステージ１にエントリするはずの命令Ｊは、命令Ｏに置き換えられる（処理連鎖モジュールは命令Ｊを要求する代わりに命令Ｏを要求する）。
【００６６】
このサイクルから始まり、処理連鎖モジュールの各ステージの一連の命令が通常通り継続される。
【００６７】
命令Ｏが実行のステージ６に到達するのがサイクルＴ１５のみであることが見て取れる。よってこのコードの分岐処理は５つのサイクルを要する。
【００６８】
分岐処理がいかに多くのプロセッササイクルと実行時間を占有するかが容易に理解できる。現実化の方法に従い、これらを回避する必要があるのはこのためである。
【００６９】
よって、分岐処理への依存度を低減した、命令実行フローを制御する新構造を提案する。
【００７０】
しかしながら、現実化の方法は分岐処理の使用を排除しない。
【００７１】
命令実行条件を試験するための分岐処理を、プロセッサ命令内の条件プレフィックスで置き換えることを提案する。
【００７２】
このように、図３Ａに示すような新しい命令フォーマットを提案する。現実化の方法に従い、命令３００は２つの部分からなる。第１の部分３０１は「プレフィックス」と呼ばれ、部分３０２に含まれる演算を実行する（又はしない）ことを検証するための条件を含む。例えば、命令は６４ビットでコード化され、そのうち１バイト（８ビット）を用いてプレフィックスをコード化する。
【００７３】
数種類のプレフィックスを想定することができる。図３Ｂに示すように、プレフィックスには２つの下位部分が含まれる。下位部分３０２はプレフィックスを識別し（例えば、２ビット値の０から３でコード化される）、下位部分３０３は検証する条件を含む。
【００７４】
プレフィックスはオペランド読込みステージで評価される。得られた結果により、部分３０２に含まれる一又は複数の演算が実行される。
【００７５】
数種類のプレフィックスを想定することができる。
【００７６】
例えば、コード「０」は内容のない条件を識別することができる。この場合、部分３０２に含まれる一又は複数の演算は無条件で実行される。アセンブリ言語では、命令は直接記述されてもよい。命令に優先する条件がないということは、内容のない条件と等価であ

る。
【００７７】
例えば、コード「１」は待機条件を識別することができる。この場合、部分３０２に含まれる一又は複数の演算は、部分３０３に含まれる条件が満たされるまで実行されない。よってこのプレフィックスは現在の命令アドレスにおける分岐処理と等価である。このプレフィックスはアセンブリ言語における「ｗａｉｔ＿ｆｏｒ」によって識別される。
【００７８】
例えば、コード「２」は部分３０２に含まれる一又は複数の演算を実行する又は演算を実行しないための条件が満たされていることの検証を行うことを識別する。よって、条件３０３が成立していれば、一又は複数の演算が実行され、当該条件が成立していなければ、演算は実行されない（一般的に「ＮＯＰ」で示される内容のない命令で命令が置き換えられる）。このプレフィックスはアセンブリ言語における「ｄｏ＿ｉｆ（）」によって示される。
【００７９】
コード「３」は部分３０２に含まれる一又は複数の演算を実行する又は演算を実行しないための条件が満たされていないことの検証を行うことを識別する。よって、条件３０３が成立しない場合、一又は複数の命令が実行され、当該条件が成立していれば、演算は実行されない（一般的に「ＮＯＰ」で示される内容のない命令で命令が置き換えられる）。このプレフィックスはアセンブリ言語における「ｄｏ＿ｉｆ＿ｎｏｔ（）」によって示される。
【００８０】
部分３０３内の条件は、一又は複数の述語レジスタビットのアドレス、又は述語レジスタビットと固定値との比較の結果を記憶するレジスタのアドレスによって表されてもよい。
【００８１】
図４Ａから４Ｇは、図３Ａに示す構造の部分３０２に含まれる命令の例を示す。
【００８２】
各命令は、フィールド４００内の命令コードによって識別される。
【００８３】
図４Ａは内容のない「ＮＯＰ」演算を示す。これは内容のない命令であるため、フィールド４００によって識別された場合、残りの部分４０１は特に情報を含まない。よってこの部分は使用されない。
【００８４】
図４Ｂは「ＳＴＯＰ」命令を示す。この命令はプログラムの終了を示す。フィールド４００によって識別された場合、残りの部分４０２は特に情報を含まない。よってこの部分は使用されない。
【００８５】
図４Ｃは即値オペランドを有する命令を示す。これは、オペランドの１つがその値を記憶しているメモリアドレスによって表されるのではなく、値そのものにより直接表される命令であり、例えば、フィールド４０３において４バイトでコード化される。この命令はまた、第２のオペランドのメモリアドレスを表すフィールド４０４、命令の結果を記憶するメモリアドレスを表すフィールド４０５、及び２つのオペランドに対してＡＬＵ処理装置によって実行される演算のコードを含むフィールド４０６を含む。
【００８６】
オペランドの直接コード化は多くのスペースを必要とする。図４Ｃの命令は多数の演算を並行して実行することは考慮していない。
【００８７】
図４Ｄは単一演算命令を示す。これはＡＬＵ処理装置による単一の演算のみを指図する命令である。ここでこの２つのオペランドはそれらを記憶するメモリアドレスによって表される。よってこの命令は、第２のオペランドのメモリアドレスを表すフィールド４０７

、第１のオペランドのメモリアドレスを表すフィールド４０８、命令の結果を記憶するメモリアドレスを表すフィールド４０９、及び２つのオペランドに対してＡＬＵ処理装置によって実行される演算のコードを含むフィールド４１０を含む。フィールド４１１は使用されない。

【００８８】

図４Ｅは二演算命令を示す。これはＡＬＵ処理装置による２つの演算の実行を指図する命令である。各演算はフィールド４１２及び４１３によって表される。フィールド４１２（及び４１３）内において、フィールド４１４（及び４１８）は第２のオペランドのメモリアドレスを表し、フィールド４１５（及び４１９）は第１のオペランドのメモリアドレスを表し、フィールド４１６（及び４２０）は命令の結果を記憶するメモリアドレスを表し、フィールド４１７（及び４２１）は２つのオペランドに対してＡＬＵ処理装置によって実行される演算のコードを含む。２つの演算は並行して実行することができるが、それは命令３００のサイズがそれを考慮しているからである。

【００８９】

図４Ｆは分岐処理及び演算を有する命令を示す。これは処理連鎖モジュールによる分岐処理の実行及びＡＬＵ処理装置による１つの演算の実行を指図する命令である。分岐処理はフィールド４２２によって表され、演算はフィールド４２３によって表される。

【００９０】

フィールド４２３内において、フィールド４２４は第２のオペランドのメモリアドレスを表し、フィールド４２５は第１のオペランドのメモリアドレスを表し、フィールド４２６は命令の結果を記憶するメモリアドレスを表し、フィールド４２７は２つのオペランドに対してＡＬＵ処理装置によって実行される演算のコードを含む。

【００９１】

フィールド４２２内において、フィールド４２８は分岐処理が指し示す命令のメモリアドレス（「ジャンプ」先のアドレス）を表し、フィールド４２９は検証を行う述語レジスタの述語ベクトルを表し（現実化の方法に従って、いくつかの述語が単一の演算で検証できることが望ましく、よって述語ベクトルは以下に述べるように言われている）、フィールド４３０は述語ベクトルのターゲット値（分岐の条件が満たされていると考えられる値を含む）を含み、フィールド４３１は分岐処理を表すコードを含む。

【００９２】

分岐と演算とは並行して実行することができるが、それは命令３００のサイズがそれを考慮しているからである。

【００９３】

図４Ｇは単一分岐命令を示す。これは処理連鎖モジュールによる分岐処理のみを指図する命令である。

【００９４】

フィールド４３２は分岐処理が指し示す命令のメモリアドレス（「ジャンプ」先のアドレス）を表し、フィールド４３３は検証を行う述語レジスタの述語ベクトルを表し（現実化の方法に従って、いくつかの述語が単一の演算で検証できることが望ましく、よって述語ベクトルは以下に述べるように言われている）、フィールド４３４は述語ベクトルのターゲット値（分岐の条件が満たされていると考えられる値を含む）を含み、フィールド４３５は分岐処理を表すコードを含む。フィールド４３６は使用されない。

【００９５】

他の種類の命令を想定することができる。例えば、ＡＬＵ処理装置の単元的な演算の組合せの実現を可能にする複合演算を含む命令を想定することが可能である。また、従来の演算又は分岐処理を有する複合演算を組み合わせること、又は２つの複合演算を同一の命令内で並行して起動することが想定できる。

【００９６】

上述の条件プレフィックスの使用により、プロセッサのサイクルを向上させることができる。以下の非常に簡素化した例によってこれを説明する。実行された分岐処理（満たさ

れた条件）が５サイクルを消費することはすでに述べたが、実行されない分岐処理（満たされない条件）は１サイクルを消費するのみである。

【００９７】

分岐処理を実装した以下のコードを見てみよう。

ＣＯＤＥ＿Ａ／／

１：　ｏｐ０

／／　演算ｏｐ０の実行

２：　ｂｒａｎｃｈ＿ｉｆ＿ｎｏｔ（ｐ０）　Ｌ０

／／　述語ｐ０が成立しない場合、コードをＬ０に分岐させる。

３：　ｏｐ１

／／　演算ｏｐ１の実行この行はｐ０が成立する場合に実行される。

４：　ｂｒａｎｃｈ　Ｌ１

／／　次の行がＬ０であり、ｐ０が成立する場合は実効されないため、Ｌ１への無条件分岐となる。

５：Ｌ０：　ｏｐ２

／／　演算ｏｐ２の実行

６：Ｌ１：　ｏｐ３

／／　演算ｏｐ３の実行

【００９８】

上記のコードでは、ｏｐ０を実行し、その後ｐ０が成立するか否かによって、ｏｐ２に続いてｏｐ３を実行する（ｐ０が成立しない場合）、又はｏｐ１に続いてｏｐ３を実行する（ｐ０が成立する場合）。

サイクルについては、ｐ０が成立しない場合、８サイクルが消費される。すなわち、

－　ｏｐ０に１サイクル、

－　行２で発生する分岐処理に５サイクル、

－　ｏｐ２に１サイクル、

－　ｏｐ３に１サイクル、である。

サイクルについては、ｐ０が成立する場合、９サイクルが消費される。すなわち、

－　ｏｐ０に１サイクル、

－　行２で発生しない分岐処理に１サイクル、

－　ｏｐ１に１サイクル、

－　行４で発生する分岐処理に５サイクル、

－　ｏｐ３に１サイクル、である。

【００９９】

次のコードを見てみよう。このコードは同じプログラムを実行するが、上述のプレフィックスを有する条件付き命令を用いる。

ＣＯＤＥ＿Ｂ／／

１：　ｏｐ０

／／　演算ｏｐ０の実行

２：　ｄｏ＿ｉｆ＿（ｐ０），　ｏｐ１

／／　ｐ０が成立するという条件で演算ｏｐ１を実行する。図３Ａに示す例において、フィールド３０１は述語ｐ０のアドレス及びＡＬＵ処理装置のＩＦ演算のコードを含む。フィールド３０２は演算ｏｐ１の表現を含む。プロセッサが複数の演算を並行して処理する大きな命令を実装できるのであれば、図４Ｄ又は４Ｃのモデルを使用することも可能である。

３：　ｄｏ＿ｉｆ＿ｎｏｔ（ｐ０），　ｏｐ２

／／　ｐ０が成立しないという条件で演算ｏｐ１を実行する。

４：　ｏｐ３

／／　演算ｏｐ３の実行

【０１００】

コードの記述が簡素化されていることがすでに明確である。このコードは４行以下であり、先ほどのコードは６行であった。また、このコードは分岐処理を用いていない。
サイクルについては、ｐ０が成立しない場合、４サイクルが消費される。すなわち、
－　ｏｐ０に１サイクル、
－　行２の条件付き命令に１サイクル、
－　行３の条件付き命令に１サイクル、
－　ｏｐ３に１サイクル、である。
【０１０１】
サイクルについては、ｐ０が成立する場合も、４サイクルが消費される。
【０１０２】
よって本発明に従うプロセッサを用いれば、同じプログラムがはるかに高速に実行できる。
【０１０３】
現実化の方法によれば、さらにプログラムの実行を高速化することができる。
【０１０４】
そのためには、ビット単位の比較演算を導入し、レジスタの１ビットを試験してその結果を述語レジスタに書き出すことを可能にする。
【０１０５】
実際、試験（上記の例におけるｐ０）を行うための条件の値を得るためにこの種の比較が頻繁に実装されているという事実を踏まえると、プロセッサが自由に用いることができる専用の演算を用意することが有利である。
【０１０６】
このような演算を以下の通り記述できる：ｃｍｐ＿ｂｉｔ＿１＿ｔｏ＿ｒｅｇ２［２８］，　ｐ０。この演算はレジスタｒｅｇ２の２８番目の位置にあるビットを値「１」と比較し、結果（一致するときは「１」、一致しないときは「０」）を述語ｐ０に書き込む。よって、この演算は試験を行うビットの位置の値（２８）、レジスタｒｅｇ２のアドレス、及び述語ｐ０のアドレスをオペランドとして用いる。
【０１０７】
従来のプロセッサでは、２つのサイクルが同じ結果を達成することが求められる。
１：　ａｎｄ　ｒｅｇ２，　０ｘ１００００００，　ｒｅｇ０
／／　ここではレジスタｒｅｇ２の内容と値２８との間でＡＮＤ論理が（１６進法で）実行され、結果をレジスタｒｅｇ０に記憶する。
２：　ｃｍｐ＿ｎｅｑ＿ｔｏ　ｒｅｇ０，　０，　ｐ０
／／　この命令はレジスタｒｅｇ０の内容と０とを比較し、結果を述語ｐ０に記憶する。このように、ｒｅｇ０が０でない場合、ｐ０は成立し、ｒｅｇ０が０である場合、ｐ０は成立しない。
【０１０８】
上記のコード例ＣＯＤＥ＿Ａにおいて、ｐ０及びｐ１という２つの述語が用いられている。よって、このコードにおいてそれらの値を得るためには４つのサイクルが必要になる。
【０１０９】
（ｃｍｐ＿ｂｉｔ＿１＿ｔｏ＿ｒｅｇ２［２８］，　ｐ０の種類の）専用の演算を用いれば、この数は２に減らすことができる。
【０１１０】
さらに、並行して実行される演算を有する命令にプロセッサが対応していれば、この数は１サイクルにまで減らすことができる。
【０１１１】
よって、述語ｐ０及びｐ１を得る処理を以下のように記述する必要はない。
１：　ａｎｄ　ｒｅｇ２，　０ｘ１００００００，　ｒｅｇ０
／／　レジスタｒｅｇ２の２８番目の位置のビットの値を取得してｒｅｇ０に記憶する

。
２：　ａｎｄ　ｒｅｇ２，　０ｘ１０００００００００００００００，　ｒｅｇ１
／／　レジスタｒｅｇ２の５６番目の位置のビットの値を取得してｒｅｇ０に記憶する
。
３：　ｃｍｐ＿ｎｅｑ＿ｔｏ　ｒｅｇ０，　０，　ｐ０
／／　この命令はレジスタｒｅｇ０の内容と０とを比較し、結果を述語ｐ０に記憶する。このように、ｒｅｇ０が０でない場合、ｐ０は成立し、ｒｅｇ０が０である場合、ｐ０は成立しない。
４：　ｃｍｐ＿ｅｑ＿ｔｏ　ｒｅｇ１，　０，　ｐ１
／／　この命令はレジスタｒｅｇ１の内容と０とを比較し、結果を述語ｐ１に記憶する。このように、ｒｅｇ１が０である場合、ｐ１は成立し、ｒｅｇ１が０でない場合、ｐ１は成立しない。

【０１１２】

これは１行で記述することができる。
１：　ｃｍｐ＿ｂｉｔ＿１＿ｔｏ＿ｒｅｇ２［２８］，　ｐ０　｜｜　ｃｍｐ＿ｂｉｔ＿０＿ｔｏ＿ｒｅｇ２［５６］，　ｐ１
／／　縦の二重線はこれらの演算が並行して実行されることを意味する。

【０１１３】

レジスタ内の試験する位置の直値が操作されないため、演算の並行実行が可能となる（従来技術にはこのような構成がない）。直値を使用すると、（図４Ｃを参照すれば理解できるように）多数のビットを必要とするため、並行演算ができなくなるものと理解される。

【０１１４】

すでに述べたように、プレフィックスを有する条件付き命令の使用は分岐処理の使用を排除するものではない。ただし、現実化の方法は既知の分岐処理に改善を行う。

【０１１５】

やはりプロセッサのサイクルを向上させるという同じ目標のために、述語ベクトルに分岐処理を作成していくつかの条件を同時に試験することが提案される。

【０１１６】

このような分岐命令は、ｂｒａｎｃｈ＿ｉｆ＿ｖｅｑ（ａｂｃｄ），　Ｌ０と記述することができる。ここでは、述語ベクトル｛ｐ３，ｐ２，ｐ１，ｐ０｝が｛ａ，ｂ，ｃ，ｄ｝と等しい場合に行Ｌ０で示される命令に分岐処理が行われる。パラルータａ、ｂ、ｃ、及びｄは０又は１の値を取り得るが、述語の１つに対して試験を行う必要がなければｘとしてもよい。例えば、条件が述語ｐ０及びｐ１にのみ適用される場合、以下の種類の命令を記述できる：ｂｒａｎｃｈ＿ｉｆ＿ｖｅｑ（ｘｘｃｄ），　Ｌ０。

【０１１７】

以下、上述の異なる特性を同一のコードに組み合わせてプロセッサのサイクルを向上させる。最初に、コードが従来技術に従って記述及び実行されるのと同じように提示される。その後最適化されたコードが提示される。このコードは同じ演算を実行しつつも必要なサイクル数を低減している。

【０１１８】

以下のコードを見てみよう。このコードは従来技術に従う命令を用いて記述されている。

ＣＯＤＥ＿Ｃ　／／
１：　ｏｐ１　｜｜　ｏｐ２
／／　演算ｏｐ１及びｏｐ２は並行して実行される。
２：　ａｎｄ　ｒｅｇ２，　０ｘ１０００００００，　ｒｅｇ０
／／　レジスタｒｅｇ２の２８番目の位置のビットの値（１６進数）を取得してｒｅｇ０に記憶する。
３：　ａｎｄ　ｒｅｇ２，　０ｘ１０００００００００００００００，　ｒｅｇ１

／／　レジスタｒｅｇ２の５６番目の位置のビットの値を取得してｒｅｇ０に記憶する。
４：　ｃｍｐ＿ｎｅｑ＿ｔｏ　ｒｅｇ０，　０，　ｐ０
／／　この命令はレジスタｒｅｇ０の内容と０とを比較し、結果を述語ｐ０に記憶する。このように、ｒｅｇ０が０でない場合、ｐ０は成立し、ｒｅｇ０が０である場合、ｐ０は成立しない。
５：　ｃｍｐ＿ｅｑ＿ｔｏ　ｒｅｇ１，　０，　ｐ１
／／　この命令はレジスタｒｅｇ１の内容と０とを比較し、結果を述語ｐ１に記憶する。このように、ｒｅｇ１が０である場合、ｐ１は成立し、ｒｅｇ１が０でない場合、ｐ１は成立しない。
６：　ｏｐ３　｜｜　ｏｐ４
／／　演算ｏｐ３及びｏｐ４は並行して実行される。
７：　ｂｒａｎｃｈ＿ｉｆ＿ｎｏｔ（ｐ０）　Ｌ０
／／　述語ｐ０が成立しない場合、このコードは行Ｌ０に分岐する。
８：　ｏｐ５
／／　演算ｏｐ５が実行される。この行はｐ０が成立する場合に実行される。
９：　ｂｒａｎｃｈ　Ｌ１
／／　次の行がＬ０であり、ｐ０が成立する場合は実効されないため、Ｌ１への無条件分岐となる。
１０：Ｌ０：　ｏｐ６
／／　演算ｏｐ６が実行される。
１１：Ｌ１：　ｏｐ７
／／　演算ｏｐ７が実行される。
１２：　ｂｒａｎｃｈ＿ｉｆ＿ｎｏｔ（ｐ１）　Ｌ２
／／　述語ｐ１が成立しない場合、このコードは行Ｌ２に分岐する。
１３：　ｏｐ８
／／　演算ｏｐ８が実行される。この行はｐ１が成立する場合に実行される。
１４：　ｂｒａｎｃｈ　Ｌ３
／／　次の行がＬ２であり、ｐ１が成立する場合は実効されないため、Ｌ３への無条件分岐となる。
１５：Ｌ２：　ｏｐ９
／／　演算ｏｐ９が実行される。
１６：Ｌ３：　ｏｐ１０
／／　演算ｏｐ１０が実行される。
１７：　ｏｒ，　ｐ０，　ｐ１，　ｐ２
ｐ０とｐ１のＯＲ論理によって述語ｐ２が計算される。
１８：　ｂｒａｎｃｈ＿ｉｆ＿ｎｏｔ（ｐ２）　Ｌ４
／／　述語ｐ２が成立しない場合、このコードは行Ｌ４に分岐する。
１９：　ｏｐ１１
／／　演算ｏｐ８が実行される。この行はｐ２が成立する場合に実行される。
２０：　ｓｔｏｐ
／／　プログラムの終了（Ｌ４に分岐した場合を除く）。
２１：Ｌ４：　ｏｐ１２
／／　演算ｏｐ１２が実行される。

【０１１９】
よって、コードＣＯＤＥ＿Ｃは３つの述語ｐ０、ｐ１、及びｐ２及び３つの条件分岐を含む。また、コードＣＯＤＥ＿Ｃは単純な演算実行及び（｜｜の記号で表される）並行実行を含む。

【０１２０】
実行された条件分岐が５つのサイクルを消費し、（条件が満たされないために）実行さ

れない分岐が１サイクルのみを消費するため、ｐ０が成立しｐ１が成立しない場合（及びその逆）は、このコードは２５サイクルで実行されると判断できる（ｓｔｏｐ命令も１サイクルのみ消費すると見なされる）。ｐ０とｐ１が共に成立する、又は成立しない場合、このコードは２４サイクルで実行される。

【０１２１】

上述のプレフィックスを有する条件付き命令及び比較及び分岐命令を用いたコードの記述を以下に示す。

```
ＣＯＤＥ＿Ｄ　／／
１：　ｏｐ１　｜｜　ｏｐ２
／／　演算ｏｐ１及びｏｐ２は並行して実行される。
２：　ｃｍｐ＿ｂｉｔ＿１＿ｔｏ＿ｒｅｇ２［２８］，　ｐ０　｜｜　ｃｍｐ＿ｂｉｔ＿０＿ｔｏ＿ｒｅｇ２［５６］，　ｐ１
／／　ＣＯＤＥ＿Ｃの２，３，４，及び５行はここで（２つの並行して処理される演算を有する）単一の行及び命令に短縮される。
３：　ｏｐ３　｜｜　ｏｐ４
／／　演算ｏｐ３及びｏｐ４は並行して実行される。
４：　ｄｏ＿ｉｆ＿（ｐ０），　ｏｐ５
／／　演算ｏｐ５はｐ０が成立する場合に実行される。
５：　ｄｏ＿ｉｆ＿ｎｏｔ（ｐ０），　ｏｐ６
／／　演算ｏｐ６はｐ０が成立しない場合に実行される。
６：　ｏｐ７
／／　演算ｏｐ７が実行される。
７：　ｄｏ＿ｉｆ＿（ｐ１），　ｏｐ８
／／　演算ｏｐ８はｐ１が成立する場合に実行される。
８：　ｄｏ＿ｉｆ＿ｎｏｔ（ｐ１），　ｏｐ９
／／　演算ｏｐ９はｐ１が成立する場合に実行される。
６：　ｏｐ１０
／／　演算ｏｐ１０が実行される。
７：　ｂｒａｎｃｈ＿ｉｆ＿ｖｅｑ（ｘｘ００）　ＬＡＢ０
／／　述語ｐ１及びｐ０が成立しない場合、このコードは行ＬＡＢ０に分岐する。なお、この方法ではコードＣＯＤＥ＿Ｃの１７行のＯＲを実行することを回避している。
９：　ｏｐ１１
／／　演算ｏｐ１１が実行される。この行はｐ１又はｐ２が成立する場合に実行される。
１０：　ｓｔｏｐ
／／　プログラムの終了（ＬＡＢ０に分岐した場合を除く）。
１１：ＬＡＢ０：　ｏｐ１２
／／　演算ｏｐ１２が実行される。
```

【０１２２】

なお、ここではコードがよりコンパクトになり、わずか１１行の命令からなる。コードＣの２１行と比較すると大きく低減されている。

【０１２３】

最悪の場合（ｐ１及びｐ０が共に成立しない）には、分岐処理が実行されるが、ＣＯＤＥ＿Ｄは１５サイクルで実行される。これはＣＯＤＥ＿Ｃの最善の場合の２４サイクルよりもはるかに少ない。

【０１２４】

この詳細な説明及び添付の図面によって本発明を説明した。しかしながら、本発明は説明した実施形態には限定されない。本発明の説明及び添付の図面に触れた当業者が、他の変形例及び現実化の方法を演繹及び実装することは可能である。

【０１２５】

請求項においては、「含む（ｃｏｍｐｒｉｓｅ）」の語は他の構成要素又は他の工程を排除しない。不定冠詞「ａ」は複数であることを排除しない。提示された、及び／又は権利を主張された様々な構成は、有益に組み合わせることができる。これらの構成が説明中又は異なる独立請求項中にあったとしても、組合せの可能性は排除されない。参照符号は発明の範囲を限定するものと解釈すべきではない。

【図１Ａ】

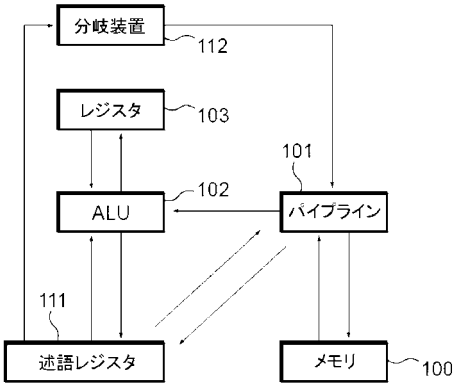


Fig. 1A

【図１Ｂ】

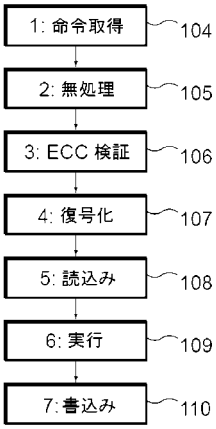


Fig. 1B

【図３Ａ】

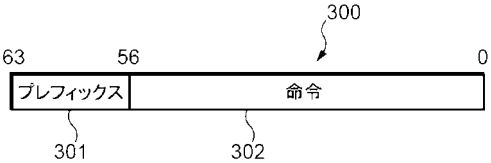
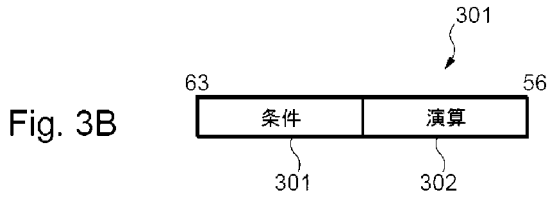
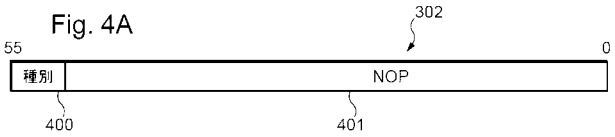


Fig. 3A

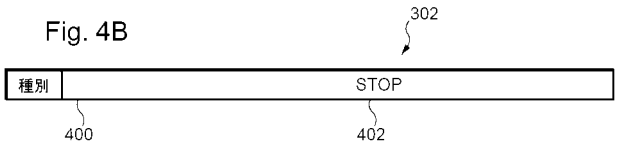
【図３Ｂ】



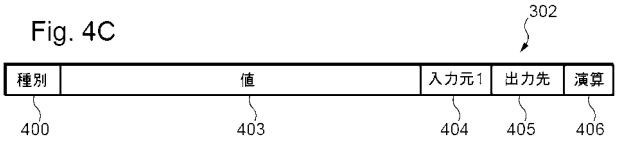
【図４Ａ】



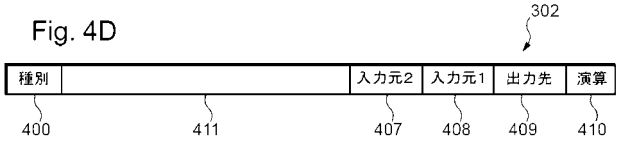
【図４Ｂ】



【図４Ｃ】



【図４Ｄ】



【図４Ｅ】

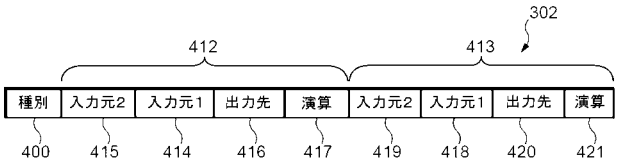


Fig. 4E

【図４Ｆ】

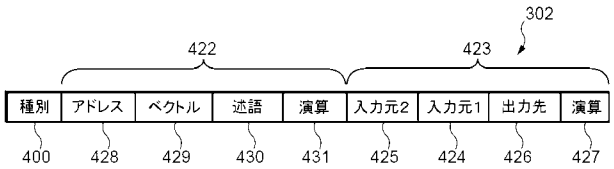


Fig. 4F

【図４Ｇ】

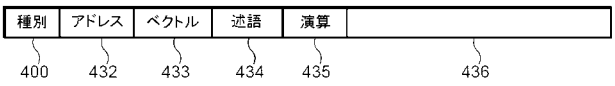


Fig. 4G

【図２】

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
1: 命令取得	A	B	C	D	E	F	G	H	I	~~J~~ O	P	Q	R	S	T	U
2: 無処理		A	B	C	D	E	F	G	H	I	O	P	Q	R	S	T
3: ECC検証			A	B	C	D	E	F	G	H		O	P	Q	R	S
4: 復号化				A	B	C	D	E	F	G			O	P	Q	R
5: 読込み					A	B	C	D	(E)	F				O	P	Q
6: 実行						A	B	C	D	~~E~~					O	P
7: 書込み							A	B	C	D						O

分岐処理

Fig. 2

【外国語明細書】
2016006632000001.pdf