(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2017-33342
(P2017-33342A)
(43)公開日　平成29年2月9日(2017.2.9)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/45	(2006.01)	G06F	9/44	322F	5B081
		G06F	9/44	322E	

審査請求 未請求　請求項の数 7　ＯＬ　（全 35 頁）

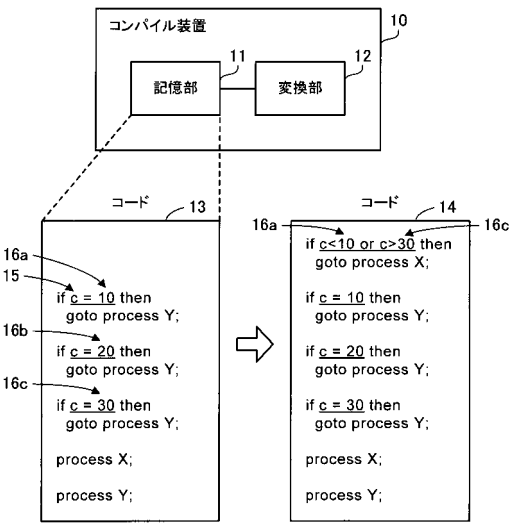
(21)出願番号　特願2015-153450 (P2015-153450)
(22)出願日　平成27年8月3日 (2015.8.3)

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100092152
弁理士　服部　毅巖
(72)発明者　三好　貴洋
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　千葉　修一
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

Ｆターム(参考)　5B081 CC21　CC41

(54)【発明の名称】コンパイル装置、コンパイル方法およびコンパイルプログラム

(57)【要約】
【課題】複数の比較値についての比較処理の負荷を軽減する。
【解決手段】記憶部１１は、変数１５の値と３以上の比較値（比較値１６ａ，１６ｂ，１６ｃ）それぞれとを比較し、比較結果に応じて分岐制御を行うコード１３を記憶する。変換部１２は、比較値１６ａ，１６ｂ，１６ｃの中から最小の比較値１６ａと最大の比較値１６ｃとを判定する。変換部１２は、コード１３から、変数１５の値と比較値１６ａ，１６ｃとを比較し、変数１５の値が比較値１６ａより小さいかまたは比較値１６ｃより大きい場合には、他の比較値との比較を迂回して分岐制御を行うコード１４に変換する。
【選択図】図１



【特許請求の範囲】
【請求項１】
変数の値と３以上の比較値それぞれとを比較し、比較結果に応じて分岐制御を行う第１のコードを記憶する記憶部と、
前記３以上の比較値の中から最小の比較値と最大の比較値とを判定し、前記第１のコードから、前記変数の値と前記最小の比較値および前記最大の比較値とを比較し、前記変数の値が前記最小の比較値より小さいかまたは前記最大の比較値より大きい場合には、前記３以上の比較値のうちの他の比較値との比較を迂回して前記分岐制御を行う第２のコードに変換する変換部と、
を有するコンパイル装置。
【請求項２】
前記変換部は、前記３以上の比較値をソートし、前記３以上の比較値の中から第１の比較値と前記第１の比較値の次に大きい第２の比較値とを選択し、
前記第２のコードは、前記変数の値が前記最小の比較値以上かつ前記最大の比較値以下である場合、前記変数の値と前記第１の比較値および前記第２の比較値とを比較し、前記変数の値が前記第１の比較値より大きくかつ前記第２の比較値より小さい場合には、前記３以上の比較値のうちの他の比較値との比較を迂回して前記分岐制御を行う、
請求項１記載のコンパイル装置。
【請求項３】
前記変換部は、ソートされた前記３以上の比較値の中から、間隔が最も離れた隣接する２つの比較値を前記第１の比較値および前記第２の比較値として選択する、
請求項２記載のコンパイル装置。
【請求項４】
前記変換部は、前記変数の値域と前記３以上の比較値とに基づいて、前記変数の値が前記第１の比較値より大きくかつ前記第２の比較値より小さいか判定した場合の前記分岐制御の効率を評価し、評価結果が所定条件を満たす場合に前記第２のコードを出力する、
請求項２記載のコンパイル装置。
【請求項５】
前記変換部は、前記３以上の比較値の中から一の比較値を選択し、また、前記第２のコードにおける前記３以上の比較値の比較順序をソートし、
前記第２のコードは、前記変数の値が前記最小の比較値以上かつ前記最大の比較値以下である場合、前記変数の値と前記一の比較値とを比較し、前記変数の値と前記一の比較値との大小に応じて、前記３以上の比較値のうちの一部の比較値との比較を迂回して前記分岐制御を行う、
請求項１記載のコンパイル装置。
【請求項６】
コンピュータが実行するコンパイル方法であって、
第１のコードの中から、変数の値と３以上の比較値それぞれとを比較し、比較結果に応じて分岐制御を行う命令群を検出し、
前記３以上の比較値の中から最小の比較値と最大の比較値とを判定し、
前記第１のコードから、前記変数の値と前記最小の比較値および前記最大の比較値とを比較し、前記変数の値が前記最小の比較値より小さいかまたは前記最大の比較値より大きい場合には、前記３以上の比較値のうちの他の比較値との比較を迂回して前記分岐制御を行う第２のコードに変換する、
コンパイル方法。
【請求項７】
コンピュータに、
第１のコードの中から、変数の値と３以上の比較値それぞれとを比較し、比較結果に応じて分岐制御を行う命令群を検出し、
前記３以上の比較値の中から最小の比較値と最大の比較値とを判定し、

前記第１のコードから、前記変数の値と前記最小の比較値および前記最大の比較値とを比較し、前記変数の値が前記最小の比較値より小さいかまたは前記最大の比較値より大きい場合には、前記３以上の比較値のうちの他の比較値との比較を迂回して前記分岐制御を行う第２のコードに変換する、

処理を実行させるコンパイルプログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明はコンパイル装置、コンパイル方法およびコンパイルプログラムに関する。

【背景技術】

【０００２】

ソフトウェア開発においては、人間が理解容易な高級言語を用いてソースコードを作成し、コンパイラを用いてソースコードを機械可読なオブジェクトコードに変換することが多い。コンパイラは、ソースコードをオブジェクトコードに変換する（コンパイルする）過程で、オブジェクトコードの実行効率が向上するように各種の最適化を行う。最適化には、例えば、冗長な命令の削除、命令の実行順序の入れ替え、特殊な命令を使用することによる命令数の削減、並列処理プロセッサを想定した並列化などが含まれる。

【０００３】

ところで、ソースコードには、連続する複数の比較処理が記述されることがある。例えば、ある変数の値が複数の比較値のうちの何れか１つと一致するか判定し、何れか１つの比較値と一致する場合に所定の処理を開始するソースコードが記述される場合がある。このようなソースコードを単純にコンパイルすると、変数の値と１つの比較値とを比較する比較命令と、比較結果が真の場合に所定の命令にジャンプする分岐命令とが、比較値の数だけ繰り返されたオブジェクトコードが生成される可能性がある。このように、複数の比較処理に応じて分岐制御が行われる場合、プロセッサの負荷が高くなりやすい。このため、コンパイラが、比較処理や分岐制御を最適化することがある。

【０００４】

例えば、連続する複数の比較処理を記述したソースコードから、比較命令の少ないオブジェクトコードを生成するコンパイラが提案されている。提案のコンパイラは、ソースコードから変換された中間コードの中から、データＡとデータＣとを比較する比較命令＃１を検出する。また、コンパイラは、データＡと連続したメモリ領域に格納されることになるデータＢと、データＣと連続したメモリ領域に格納されることになるデータＤとを比較する比較命令＃２を検出する。コンパイラは、１回に比較するデータの範囲を拡張することで、検出した比較命令＃１，＃２を１つの比較命令にマージする。

【０００５】

また、並列処理プロセッサを利用して分岐命令のジャンプ先の命令を先行して実行し、条件式の値が偽になった場合は先行の実行結果を破棄するようにしたオブジェクトコードを生成するコンパイラが提案されている。提案のコンパイラは、オブジェクトコードをテスト実行させて条件式の値が真になる確率を算出する。コンパイラは、条件式の値が偽になる確率の方が高い場合、条件式の真偽を反転させて分岐命令のジャンプ先を入れ替える。これにより、条件式の値が本来偽の場合に実行されることになる命令が並列処理プロセッサによって先行して実行され、先行の実行結果が破棄される確率を低減できる。

【０００６】

また、オブジェクトコードに含まれる分岐命令を削減するコンパイラが提案されている。提案のコンパイラは、ソースコードから変換された中間コードの中から、比較命令＃１と、比較命令＃１の結果が偽の場合に所定の命令にジャンプする分岐命令＃１を検出する。また、コンパイラは、比較命令＃２と、比較命令＃２の結果が偽の場合に分岐命令＃１と同じ所定の命令にジャンプする分岐命令＃２を検出する。コンパイラは、分岐命令＃１，＃２を、比較命令＃１の結果と比較命令＃２の結果の論理積を算出する論理演算命令と、論理積が偽の場合に上記所定の命令にジャンプする分岐命令＃３とに置換する。

【０００７】
また、複数の比較命令に続く命令を実行するか無効化するかを、条件フラグレジスタに格納された真偽値に基づいて選択するプロセッサが提案されている。提案のプロセッサは、比較命令＃１が入力されると、比較命令＃１の比較結果を条件フラグレジスタに格納する。プロセッサは、比較命令＃１と連続して比較命令＃２が入力されると、条件フラグレジスタに格納された真偽値と比較命令＃２の比較結果の論理積を算出し、論理積を条件フラグレジスタに格納する。その後、プロセッサは、比較命令以外の命令が入力されると、条件フラグレジスタを参照し、条件フラグレジスタが偽を示している場合に当該入力された命令を無効化する。また、上記プロセッサの機能を利用して、分岐命令を含まないオブジェクトコードを生成するコンパイラが提案されている。

【先行技術文献】
【特許文献】
【０００８】
【特許文献１】特開平２－１６３８３１号公報
【特許文献２】特開２００１－１１７８９０号公報
【特許文献３】特開２００１－２６５６０５号公報
【特許文献４】特開２００４－２１５５３号公報
【発明の概要】
【発明が解決しようとする課題】
【０００９】
上記のように、ソースコードには、１つの変数の値と複数の比較値それぞれとを比較し、比較結果に応じて分岐制御を行うことが記述される場合がある。ある比較値について比較結果が真になれば、それ以降の比較値については比較を行わなくてよいこともある。そこで、比較処理に対するコンパイラ最適化の１つとして、複数の比較値の比較順序を入れ替えて、真になる確率が高い比較値から先に比較する方法が考えられる。各比較値の真になる確率は、例えば、オブジェクトコードをテスト実行することで算出される。
【００１０】
しかし、複数の比較値の比較順序を入れ替える方法には、比較処理の負荷を軽減する点で改善の余地がある。例えば、変数の値が複数の比較値の何れとも一致しない可能性がある場合、それら複数の比較値の全部について比較を行うことが発生し得る。１つの変数の値に対する実行時の平均の比較回数が削減されることが好ましい。
【００１１】
１つの側面では、本発明は、複数の比較値についての比較処理の負荷を軽減できるコンパイル装置、コンパイル方法およびコンパイルプログラムを提供することを目的とする。
【課題を解決するための手段】
【００１２】
１つの態様では、記憶部と変換部とを有するコンパイル装置が提供される。記憶部は、変数の値と３以上の比較値それぞれとを比較し、比較結果に応じて分岐制御を行う第１のコードを記憶する。変換部は、３以上の比較値の中から最小の比較値と最大の比較値とを判定し、第１のコードから、変数の値と最小の比較値および最大の比較値とを比較し、変数の値が最小の比較値より小さいかまたは最大の比較値より大きい場合には、３以上の比較値のうちの他の比較値との比較を迂回して分岐制御を行う第２のコードに変換する。
【００１３】
また、１つの態様では、コンピュータが実行するコンパイル方法が提供される。また、１つの態様では、コンピュータに実行させるコンパイルプログラムが提供される。
【発明の効果】
【００１４】
１つの側面では、複数の比較値についての比較処理の負荷を軽減できる。
【図面の簡単な説明】
【００１５】

【図１】第１の実施の形態のコンパイル装置を示す図である。
【図２】コンパイル装置のハードウェア例を示すブロック図である。
【図３】コンパイル装置の機能例を示すブロック図である。
【図４】最適化部の機能例を示すブロック図である。
【図５】プロセッサのレジスタ構成例を示すブロック図である。
【図６】プロセッサ情報の例を示す図である。
【図７】連続する分岐命令を含むプログラムの例を示す図である。
【図８】ブロック情報の例を示す図である。
【図９】比較値ベクトルと間隔マップと命令テーブルの例を示す図である。
【図１０】数値判定の最適化例を示す図である。
【図１１】最適化されたプログラムの例を示す図である。
【図１２】コンパイルの手順例を示すフローチャートである。
【図１３】マシン依存最適化の手順例を示すフローチャートである。
【図１４】ブロック情報生成の手順例を示すフローチャートである。
【図１５】ベクトル・マップ生成の手順例を示すフローチャートである。
【図１６】効率評価の手順例を示すフローチャートである。
【図１７】先行判定追加の手順例を示すフローチャートである。
【図１８】先行判定追加の手順例を示すフローチャート（続き）である。
【図１９】ブロック情報の第１の更新例を示す図である。
【図２０】探索木変換の手順例を示すフローチャートである。
【図２１】ブロック情報の第２の更新例を示す図である。
【発明を実施するための形態】
【００１６】
以下、本実施の形態を図面を参照して説明する。
［第１の実施の形態］
第１の実施の形態を説明する。
【００１７】
図１は、第１の実施の形態のコンパイル装置を示す図である。
第１の実施の形態のコンパイル装置１０は、高級言語で記述されたソースコードを機械可読なオブジェクトコードに変換（コンパイル）する。コンパイルの中で、コンパイル装置１０は、オブジェクトコードの実行効率を向上させる最適化を行う。
【００１８】
コンパイル装置１０は、後述するように、コード１３をコード１４に変換する。コード１３，１４は、プロセッサに実行させる命令を記述したものと言うこともでき、プログラムと言うこともできる。コード１３，１４は、ソースコードでもよいし、ソースコードから変換された中間コードでもよい。また、コード１３，１４は、アセンブリ言語で記述されたアセンブリコードでもよいし、オブジェクトコードでもよい。また、コンパイル装置１０は、ユーザが操作する端末装置でもよいし、端末装置からアクセスされるサーバ装置でもよい。コンパイル装置１０を、コンピュータを用いて実装してもよい。
【００１９】
コンパイル装置１０は、記憶部１１および変換部１２を有する。記憶部１１は、コード１３を記憶する。記憶部１１は、ＲＡＭ（Random Access Memory）などの揮発性の記憶装置でもよいし、ＨＤＤ（Hard Disk Drive）やフラッシュメモリなどの不揮発性の記憶装置でもよい。変換部１２は、記憶部１１に記憶されたコード１３をコード１４に変換する。変換部１２は、コード１４を記憶部１１に保存してもよい。変換部１２は、ＣＰＵ（Central Processing Unit）やＤＳＰ（Digital Signal Processor）などのプロセッサでもよいし、ＡＳＩＣ（Application Specific Integrated Circuit）やＦＰＧＡ（Field Programmable Gate Array）などの特定用途の電子回路を含んでもよい。プロセッサは、例えば、記憶部１１または他の記憶装置に記憶されたコンパイルプログラムを実行する。なお、複数のプロセッサの集合（マルチプロセッサ）を「プロセッサ」と呼んでもよい。

【００２０】
コード１３には、変数１５の値と比較値１６ａ，１６ｂ，１６ｃを含む３以上の比較値それぞれとを比較する比較処理が記述されている。また、コード１３には、比較結果に応じた分岐制御が記述されている。例えば、変数１５が整数型であり、比較値１６ａが「１０」、比較値１６ｂが「２０」、比較値１６ｃが「３０」である。ただし、文字は文字コードとしての整数で表現され得るため、変数１５が文字型であってもよい。
【００２１】
例えば、コード１３は、変数１５の値と比較値１６ａとを比較する１番目の比較命令と、１番目の比較結果が真になる（値が一致する）場合に処理Ｙにジャンプする１番目の分岐命令を含む。また、例えば、コード１３は、変数１５の値と比較値１６ｂとを比較する２番目の比較命令と、２番目の比較結果が真になる（値が一致する）場合に処理Ｙにジャンプする２番目の分岐命令を含む。また、例えば、コード１３は、変数１５の値と比較値１６ｃとを比較する３番目の比較命令と、３番目の比較結果が真になる（値が一致する）場合に処理Ｙにジャンプする３番目の分岐命令を含む。
【００２２】
上記の３つの比較命令は、比較する一方の値（変数１５の値）が同じであり、他方の値（比較値１６ａ，１６ｂ，１６ｃ）が異なる。また、上記の３つの分岐命令は、比較結果が真の場合のジャンプ先（処理Ｙ）が同じである。３つの比較結果が全て偽である場合、処理Ｘが実行される。これは、変数１５の値が「１０」，「２０」，「３０」の何れかである場合は処理Ｙが実行され、それ以外の場合は処理Ｘが実行されることを示す。
【００２３】
変換部１２は、コード１３の中から、変数１５の値と３以上の比較値それぞれとを比較して分岐制御を行う命令群を検出する。変換部１２は、３以上の比較値の中から、最小の比較値と最大の比較値を判定する。最小の比較値は、変数１５が取り得る値の範囲（変数１５の値域）の下限よりも大きい可能性がある。また、最大の比較値は、変数１５の値域の上限よりも小さい可能性がある。例えば、比較値１６ａ，１６ｂ，１６ｃのうち、比較値１６ａが最小の比較値であり、比較値１６ｃが最大の比較値である。変換部１２は、判定した最小の比較値と最大の比較値に基づいて、コード１３をコード１４に変換する。
【００２４】
コード１４には、変数１５の値と最小の比較値および最大の比較値とを比較する比較処理が記述されている。また、コード１４には、変数１５の値が最小の比較値より小さいかまたは最大の比較値より大きい場合、少なくとも最小の比較値および最大の比較値以外の比較値との比較を迂回する分岐制御が記述されている。変数１５の値が最小の比較値より小さいかまたは最大の比較値より大きいことの判定は、例えば、コード１３に記述された変数１５の値と各比較値との間の個別の比較処理よりも前に挿入される。
【００２５】
例えば、コード１４は、変数１５の値と最小の比較値１６ａとを比較する１番目の追加の比較命令と、変数１５の値が比較値１６ａより小さい場合に処理Ｘにジャンプする１番目の追加の分岐命令を含む。また、例えば、コード１４は、変数１５の値と最大の比較値１６ｃとを比較する２番目の追加の比較命令と、変数１５の値が比較値１６ｃより大きい場合に処理Ｘにジャンプする２番目の追加の分岐命令を含む。また、例えば、コード１４は、これらの追加の比較命令および追加の分岐命令の後に、コード１３と同様の比較値１６ａ，１６ｂ，１６ｃに関する３つの比較命令および３つの分岐命令を含む。
【００２６】
変数１５の値が最小の比較値より小さい場合、変数１５の値は３以上の比較値の何れとも一致しないことが明らかである。よって、この場合には、変数１５の値と３以上の比較値それぞれとの個別の比較を実行せず、処理Ｘにジャンプすることが可能である。また、変数１５の値が最大の比較値より大きい場合、変数１５の値は３以上の比較値の何れとも一致しないことが明らかである。よって、この場合にも、変数１５の値と３以上の比較値それぞれとの個別の比較を実行せず、処理Ｘにジャンプすることが可能である。

【００２７】
コード１４には、３以上の比較値についての個別の比較の前に、それら個別の比較を迂回して（実行せずに）処理Ｘにジャンプするパス（実行経路）が挿入されていると言うこともできる。一方、変数１５の値が最小の比較値以上かつ最大の比較値以下である場合、変数１５の値は３以上の比較値の何れかと一致する可能性がある。よって、この場合には、例えば、３以上の比較値についての個別の比較の少なくとも一部が実行され得る。
【００２８】
第１の実施の形態のコンパイル装置１０によれば、コード１３から、同じ変数１５の値と比較される３以上の比較値の中から最小の比較値と最大の比較値が判定される。そして、変数１５の値が最小の比較値より小さいかまたは最大の比較値より大きい場合には他の比較値との比較が迂回されるように、コード１３がコード１４に変換される。
【００２９】
これにより、変数１５の値が最小の比較値より小さいかまたは最大の比較値より大きい場合にはそれ以降の比較命令が実行されず、変数１５の値１つ当たりの実行時の平均の比較回数が削減される。よって、３以上の比較値についての比較処理の負荷を軽減することができる。これは、比較値の個数と比べて変数１５の値域が広く、変数１５の値が何れの比較値とも一致しない可能性が高い場合に特に有効である。また、比較値が多く、変数１５の値と全ての比較値とを比較すると比較処理の負荷が大きい場合に特に有効である。
【００３０】
例えば、比較値を比較結果が真になる確率が高い順にソートするだけでは、変数１５の値が何れの比較値とも一致しない場合、変数１５の値と全ての比較値とを比較することになってしまう。また、比較値の間で真になる確率の差が小さい（ばらつきが小さい）場合には、比較回数を削減することが難しい。これに対し、コード１４では、変数１５の値が何れの比較値とも一致しない場合でも比較回数を削減することができる。また、コンパイル装置１０は、比較結果が真になる確率を収集しなくてもよく、オブジェクトコードをテスト実行するコストやコンパイルのコストを削減することができる。
【００３１】
また、変数１５の値１つ当たりの比較回数が削減されることで、分岐命令の実行回数が削減されることがある。分岐命令の実行回数を削減することで、オブジェクトコードを実行するプロセッサのパイプライン処理において、分岐予測の失敗により生じる再実行コスト（ペナルティ）を削減できる。よって、オブジェクトコードの実行効率を向上させることができる。これは、分岐予測の精度が高くないプロセッサでは特に有効である。
【００３２】
［第２の実施の形態］
次に、第２の実施の形態を説明する。
図２は、コンパイル装置のハードウェア例を示すブロック図である。
【００３３】
コンパイル装置１００は、ＣＰＵ１０１、ＲＡＭ１０２、ＨＤＤ１０３、画像信号処理部１０４、入力信号処理部１０５、媒体リーダ１０６および通信インタフェース１０７を有する。ＣＰＵ１０１、ＲＡＭ１０２、ＨＤＤ１０３、画像信号処理部１０４、入力信号処理部１０５、媒体リーダ１０６および通信インタフェース１０７は、バス１０８に接続されている。なお、コンパイル装置１００は、第１の実施の形態のコンパイル装置１０に対応する。ＣＰＵ１０１は、第１の実施の形態の変換部１２に対応する。ＲＡＭ１０２またはＨＤＤ１０３は、第１の実施の形態の記憶部１１に対応する。
【００３４】
ＣＰＵ１０１は、プログラムの命令を実行する演算回路を含むプロセッサである。ＣＰＵ１０１は、ＨＤＤ１０３に記憶されたプログラムやデータの少なくとも一部をＲＡＭ１０２にロードし、プログラムを実行する。なお、ＣＰＵ１０１は複数のプロセッサコアを備えてもよく、コンパイル装置１００は複数のプロセッサを備えてもよく、以下で説明する処理を複数のプロセッサまたはプロセッサコアを用いて並列に実行してもよい。また、

複数のプロセッサの集合（マルチプロセッサ）を「プロセッサ」と呼んでもよい。

【００３５】

ＲＡＭ１０２は、ＣＰＵ１０１が実行するプログラムやＣＰＵ１０１が演算に用いるデータを一時的に記憶する揮発性の半導体メモリである。なお、コンパイル装置１００は、ＲＡＭ以外の種類のメモリを備えてもよく、複数個のメモリを備えてもよい。

【００３６】

ＨＤＤ１０３は、ＯＳ（Operating System）やミドルウェアやアプリケーションソフトウェアなどのソフトウェアのプログラム、および、データを記憶する不揮発性の記憶装置である。プログラムには、コンパイルプログラムが含まれる。なお、コンパイル装置１００は、フラッシュメモリやＳＳＤ（Solid State Drive）などの他の種類の記憶装置を備えてもよく、複数の不揮発性の記憶装置を備えてもよい。

【００３７】

画像信号処理部１０４は、ＣＰＵ１０１からの命令に従って、コンパイル装置１００に接続されたディスプレイ１１１に画像を出力する。ディスプレイ１１１としては、ＣＲＴ（Cathode Ray Tube）ディスプレイ、液晶ディスプレイ（ＬＣＤ：Liquid Crystal Display）、プラズマディスプレイ、有機ＥＬ（ＯＥＬ：Organic Electro-Luminescence）ディスプレイなどを用いることができる。

【００３８】

入力信号処理部１０５は、コンパイル装置１００に接続された入力デバイス１１２から入力信号を取得し、ＣＰＵ１０１に出力する。入力デバイス１１２としては、マウスやタッチパネルやタッチパッドやトラックボールなどのポインティングデバイス、キーボード、リモートコントローラ、ボタンスイッチなどを用いることができる。また、コンパイル装置１００に、複数の種類の入力デバイスが接続されていてもよい。

【００３９】

媒体リーダ１０６は、記録媒体１１３に記録されたプログラムやデータを読み取る読み取り装置である。記録媒体１１３として、例えば、フレキシブルディスク（ＦＤ：Flexible Disk）やＨＤＤなどの磁気ディスク、ＣＤ（Compact Disc）やＤＶＤ（Digital Versatile Disc）などの光ディスク、光磁気ディスク（ＭＯ：Magneto-Optical disk）、半導体メモリなどを使用できる。媒体リーダ１０６は、例えば、記録媒体１１３から読み取ったプログラムやデータをＲＡＭ１０２またはＨＤＤ１０３に格納する。

【００４０】

通信インタフェース１０７は、ネットワーク１１４に接続され、ネットワーク１１４を介して他の装置と通信を行うインタフェースである。通信インタフェース１０７は、スイッチなどの通信装置とケーブルで接続される有線通信インタフェースでもよいし、基地局と無線リンクで接続される無線通信インタフェースでもよい。

【００４１】

なお、コンパイル装置１００は、ユーザが操作する端末装置でもよいし、端末装置または他のサーバ装置からネットワーク１１４を介してアクセスされるサーバ装置でもよい。コンパイル装置１００は、媒体リーダ１０６を備えていなくてもよく、ユーザが操作する端末装置から制御可能である場合には画像信号処理部１０４や入力信号処理部１０５を備えていなくてもよい。また、ディスプレイ１１１や入力デバイス１１２が、コンパイル装置１００の筐体と一体に形成されていてもよい。

【００４２】

図３は、コンパイル装置の機能例を示すブロック図である。

コンパイル装置１００は、ソースコード記憶部１２１、中間コード記憶部１２２、オブジェクトコード記憶部１２３、実行ファイル記憶部１２４、制御情報記憶部１２５、コンパイラ１３１およびリンカ１３７を有する。ソースコード記憶部１２１、中間コード記憶部１２２、オブジェクトコード記憶部１２３、実行ファイル記憶部１２４および制御情報記憶部１２５は、ＲＡＭ１０２またはＨＤＤ１０３に確保した記憶領域を用いて実現できる。コンパイラ１３１およびリンカ１３７は、ＣＰＵ１０１が実行するプログラム（コン

パイルプログラムおよびリンクプログラム）を用いて実現できる。ただし、コンパイラ１３１およびリンカ１３７を、特定用途の電子回路を用いて実現してもよい。

【００４３】

ソースコード記憶部１２１は、ソースコードを記憶する。ソースコードは、Ｃ言語などの高級言語で記述されたプログラムである。ユーザは、入力デバイス１１２などを用いてコンパイル装置１００に対してソースコードを直接入力してもよいし、ネットワーク１１４を介してコンパイル装置１００にソースコードを送信してもよい。中間コード記憶部１２２は、中間コードを記憶する。中間コードは、コンパイル装置１００の内部で利用される中間言語で記述されたプログラムであり、ソースコードに対応する。

【００４４】

オブジェクトコード記憶部１２３は、オブジェクトコードを記憶する。オブジェクトコードは、プロセッサが解釈可能な機械語で記述されたプログラムであり、ソースコードや中間コードに対応する。実行ファイル記憶部１２４は、実行ファイルを記憶する。実行ファイルは、ＣＰＵが実行できるプログラムであり、オブジェクトコードとライブラリなどへのリンクとを含む。実行ファイルは、ＣＰＵ１０１が実行してもよいし、コンパイル装置１００が備える他のＣＰＵまたはコンパイル装置１００以外のコンピュータのＣＰＵが実行してもよい。コンパイラ１３１は、ターゲットのＣＰＵアーキテクチャに応じて、オブジェクトコードに使用する命令を変更する。制御情報記憶部１２５は、コンパイラ１３１が最適化に使用する各種の制御情報を記憶する。制御情報の詳細は後述する。

【００４５】

コンパイラ１３１は、ソースコードをオブジェクトコードに変換する。コンパイラ１３１は、ソースコード入力部１３２、中間コード生成部１３３、最適化部１３４、アセンブリコード生成部１３５およびオブジェクトコード出力部１３６を有する。

【００４６】

ソースコード入力部１３２は、ユーザからコンパイルコマンドを受け付け、コンパイルコマンドで指定されたソースコードをソースコード記憶部１２１から読み出す。中間コード生成部１３３は、ソースコード入力部１３２が読み出したソースコードを解析し、ソースコードを中間コードに変換する。ソースコードの解析には、字句解析、構文解析、意味解析などのいわゆるフロントエンド処理が含まれる。中間コード生成部１３３は、生成した中間コードを中間コード記憶部１２２に格納する。

【００４７】

最適化部１３４は、中間コード記憶部１２２に記憶された中間コードに対して、実行速度が向上するように各種の最適化を行う。最適化部１３４は、中間コードを解析し、所定の規則に従って中間コードを書き換える。最適化には、ターゲットのＣＰＵアーキテクチャ（オブジェクトコードを実行させる予定のＣＰＵの種類）に応じた命令変換も含まれる。ターゲットのＣＰＵアーキテクチャは、予め決まっていてもよいし、コンパイルコマンドのオプションとしてユーザから指定されてもよい。最適化の詳細は後述する。

【００４８】

アセンブリコード生成部１３５は、中間コード記憶部１２２から最適化後の中間コードを読み出し、低級言語であるアセンブリ言語で記述されたアセンブリコードに変換する。オブジェクトコード出力部１３６は、アセンブリコード生成部１３５が生成したアセンブリコードをオブジェクトコードに変換する。アセンブリコードの命令とオブジェクトコードの命令とは、通常は１対１に対応する。オブジェクトコード出力部１３６は、生成したオブジェクトコードをオブジェクトコード記憶部１２３に格納する。

【００４９】

リンカ１３７は、オブジェクトコード記憶部１２３からオブジェクトコードを読み出し、オブジェクトコードから参照されている他のオブジェクトコードやライブラリを検出する。リンカ１３７は、読み出したオブジェクトコードと検出した他のオブジェクトコードやライブラリとをリンクし、実行ファイルを生成する。リンカ１３７は、生成した実行ファイルを実行ファイル記憶部１２４に格納する。なお、コンパイラ１３１のコンパイル機

能とリンカ１３７のリンク機能とが、１つのモジュールに統合されてもよい。

【００５０】

図４は、最適化部の機能例を示すブロック図である。

最適化部１３４は、汎用最適化部１４１、ループ最適化部１４２、ＳＩＭＤ（Single Instruction Multiple Data）最適化部１４３、命令変換部１４４、マシン依存最適化部１４５および命令スケジューリング部１４８を有する。

【００５１】

汎用最適化部１４１は、中間コードに対して汎用的な最適化を実行する。汎用的な最適化には、例えば、使用されていない変数の削除、静的に決定できる値のみに依存する計算式の定数化、複数の計算式に共通に含まれる部分式の計算結果の再利用などが含まれる。

【００５２】

ループ最適化部１４２は、中間コードの中からループを検出し、ループに対して最適化を実行する。ループの最適化には、例えば、ループ内で値が変わらない変数に関する演算をループ外に移動することが含まれる。また、ループの最適化には、例えば、ループ内の命令を展開（アンローリング）して、ｉ回目（ｉは正の整数）の処理を示す命令とｉ＋１回目の処理を示す命令に分解し、ループの繰り返し数を減らすことなどが含まれる。

【００５３】

ＳＩＭＤ最適化部１４３は、ターゲットのＣＰＵアーキテクチャがＳＩＭＤ命令をサポートしている場合、中間コードの中からＳＩＭＤ命令に変換可能な２以上の命令の組を検出する。ＳＩＭＤ命令に変換可能な命令の組は、依存関係がなく演算の種類が同じスカラ命令（非ＳＩＭＤ命令）の組である。ＳＩＭＤ命令は、異なるデータに対する同じ種類の演算を並列に実行する命令である。ＳＩＭＤ最適化部１４３は、検出した２以上の命令の組をＳＩＭＤ命令にマージして、中間コードの命令数を削減する。

【００５４】

命令変換部１４４は、中間コードで使用されている命令形式を、ターゲットのＣＰＵアーキテクチャがサポートする命令セットの命令形式に変換する。すなわち、命令変換部１４４は、ＣＰＵ非依存の中間コードをＣＰＵ依存の中間コードに変換する。命令変換部１４４は、１つのＣＰＵアーキテクチャのみに対応していてもよいし、複数のＣＰＵアーキテクチャに対応していてもよい。後者の場合、例えば、ターゲットのＣＰＵアーキテクチャがユーザから指定される。コンパイラ１３１は、各ＣＰＵアーキテクチャがサポートする命令セットを示すプロセッサ情報を保持していてもよい。

【００５５】

マシン依存最適化部１４５は、ＣＰＵ依存の命令の種類（ターゲットのＣＰＵアーキテクチャに特有の命令の種類など）を活用した最適化を行う。ＣＰＵ依存の命令の種類には、後述するような各種の分岐命令が含まれる。第２の実施の形態では特に、１つの変数の値と複数の比較値とを比較し比較結果に応じて処理が分岐する制御構造を最適化することを考える。マシン依存最適化部１４５は、解析部１４６および変換部１４７を有する。

【００５６】

解析部１４６は、中間コードの中から、複数の比較値それぞれについて、ある変数の値と当該比較値とを比較する比較命令と、両者が一致する場合に所定の命令にジャンプする分岐命令とを検出する。解析部１４６は、変数の値域と比較値の個数とに基づいて、比較パターン数が最小となる制御構造を判定する。変換部１４７は、解析部１４６の解析結果に基づいて、変数の値１つ当たりの平均の比較回数が減少するように中間コードを書き換える。このとき、変換部１４７は、既存の比較命令および分岐命令を並べ替える。また、変換部１４７は、新たな比較命令および分岐命令を挿入することで、既存の比較命令および分岐命令の少なくとも一部を迂回する実行経路を作成する。

【００５７】

命令スケジューリング部１４８は、中間コードに含まれる命令の順序を変更するスケジューリングを行う。スケジューリングには、例えば、依存関係のない２以上の命令を検出し、検出した２以上の命令を異なるＣＰＵまたは異なるＣＰＵコアに割り当てる並列化が

含まれる。また、スケジューリングには、例えば、パイプラインが効率的に動作するように、依存関係のない２以上の命令の間で実行順序を入れ替えることが含まれる。

【００５８】

次に、ターゲットのＣＰＵについて説明する。

図５は、プロセッサのレジスタ構成例を示すブロック図である。

ＣＰＵ２０は、コンパイル装置１００によって生成された実行ファイルを実行可能なプロセッサの１つである。ＣＰＵ２０は、コンパイル装置１００が備えてもよいし、他のコンピュータが備えてもよい。ＣＰＵ２０は、汎用レジスタ２１ａ，２１ｂ，２１ｃ（ｒ１，ｒ２，ｒ３）を含む複数の汎用レジスタと、ステータスレジスタ２２を有する。

【００５９】

汎用レジスタ２１ａ，２１ｂ，２１ｃは、演算に使用するデータを一時的に記憶する揮発性の記憶素子（レジスタ）である。汎用レジスタ２１ａ，２１ｂ，２１ｃは、実行ファイルに含まれるユーザ命令から明示的に指定することができる。ステータスレジスタ２２は、演算の実行状況を示す制御フラグなどを記憶するレジスタである。ステータスレジスタ２２は、実行ファイルに含まれるユーザ命令からは明示的に指定されず、演算の進行に応じてＣＰＵ２０によって自動的に更新される。制御フラグには、ゼロフラグ（Ｚ）、ネガティブフラグ（Ｎ）およびオーバフローフラグ（Ｖ）が含まれる。

【００６０】

ゼロフラグは、演算結果が０か否かを示す。ゼロフラグは、例えば、１ビットで表現できる。数値演算が実行される毎にゼロフラグが更新される。演算結果が０の場合はゼロフラグがＯＮ（１）に更新され、演算結果が０以外の場合はゼロフラグがＯＦＦ（０）に更新される。２つの数値を比較する比較演算は、ＣＰＵ２０の内部では除算として実装される。このため、比較演算において２つの数値が一致する場合はゼロフラグがＯＮに更新され、２つの数値が一致しない場合はゼロフラグがＯＦＦに更新される。

【００６１】

ネガティブフラグは、演算結果が負の値か否かを示す。ネガティブフラグは、例えば、１ビットで表現できる。数値演算が実行される毎にネガティブフラグが更新される。演算結果が負の値の場合はネガティブフラグがＯＮ（１）に更新され、演算結果が０以上（非負の値）の場合はネガティブフラグがＯＦＦ（０）に更新される。

【００６２】

オーバフローフラグは、演算結果がオーバフローしたか否かを示す。オーバフローは、変数の桁数（ビット数）が不足し、演算結果が変数の型の上限値を超えたかまたは下限値を下回ったことである。例えば、ある変数の型が４ビットの符号付き整数である場合、当該変数の値域は－８～７である。この場合、演算結果が－８より小さいかまたは７より大きくなると、オーバフローが発生する。オーバフローが発生すると、変数が演算結果を正しく表現しないことになる。オーバフローフラグは、例えば、１ビットで表現できる。数値演算が実行される毎にオーバフローフラグが更新される。オーバフローが発生した場合はオーバフローフラグがＯＮ（１）に更新され、オーバフローが発生していない場合はオーバフローフラグがＯＦＦ（０）に更新される。

【００６３】

図６は、プロセッサ情報の例を示す図である。

プロセッサ情報１５１は、あるＣＰＵアーキテクチャがサポートする命令の種類を示す。コンパイラ１３１は、ターゲットのＣＰＵアーキテクチャに対応するオブジェクトコードを生成するために、プロセッサ情報１５１を保持していてもよい。プロセッサ情報１５１には、命令の種類として、ｃｍｐ命令、ｂｅｑ命令、ｂｎｅ命令、ｂｌ命令、ｂｇ命令、ｂｇｅ命令およびｂａ命令が記載されている。第２の実施の形態では、ターゲットのＣＰＵアーキテクチャがこれらの命令をサポートしているものとする。すなわち、上記のＣＰＵ２０がこれらの命令を実行することができる。

【００６４】

ｃｍｐ命令は、数値を示す２つのオペランド（オペランドＡ，Ｂ）の値を比較する比較

命令である。前述のように、ｃｍｐ命令は除算Ａ－Ｂとして実装される。オペランドＡ，Ｂの値が一致する場合、ゼロフラグがＯＮ（Ｚ＝１）に更新される。オペランドＡの値がオペランドＢの値よりも小さい場合、ネガティブフラグがＯＮ（Ｎ＝１）に更新される。除算Ａ－Ｂの結果が変数の型の上限値より大きいかまたは下限値より小さい場合、オーバフローフラグがＯＮ（Ｖ＝１）に更新される。

【００６５】

ｂｅｑ命令は、ゼロフラグがＯＮの場合にオペランドｌａｂｅｌが示す命令にジャンプし、ゼロフラグがＯＦＦの場合にはジャンプしない条件分岐命令である。ｂｅｑ命令の直前にｃｍｐ命令が実行された場合、ｂｅｑ命令は、オペランドＡ，Ｂの値が一致する場合にジャンプし一致しない場合はジャンプしないことを示す。

【００６６】

ｂｎｅ命令は、ゼロフラグがＯＦＦの場合にオペランドｌａｂｅｌが示す命令にジャンプし、ゼロフラグがＯＮの場合にはジャンプしない条件分岐命令である。ｂｎｅ命令の直前にｃｍｐ命令が実行された場合、ｂｎｅ命令は、オペランドＡ，Ｂの値が一致しない場合にジャンプし一致する場合はジャンプしないことを示す。

【００６７】

ｂｌ命令は、ネガティブフラグとオーバフローフラグの排他的論理和（Ｎ　ｘｏｒ　Ｖ）が１の場合にオペランドｌａｂｅｌが示す命令にジャンプし、排他的論理和が０の場合にはジャンプしない条件分岐命令である。ｂｌ命令の直前にｃｍｐ命令が実行された場合、ｂｌ命令は、オペランドＡの値がオペランドＢの値より小さい場合（Ａ＜Ｂ）にジャンプしそれ以外の場合（Ａ≧Ｂ）にジャンプしないことを示す。

【００６８】

ｂｇ命令は、ゼロフラグとネガティブフラグとオーバフローフラグから算出される値が１の場合にオペランドｌａｂｅｌが示す命令にジャンプし、０の場合にはジャンプしない条件分岐命令である。上記の値は、ネガティブフラグとオーバフローフラグの排他的論理和を算出し、排他的論理話とゼロフラグとの論理和を算出し、論理和を否定することで算出される（ｎｏｔ（Ｚ　ｏｒ（Ｎ　ｘｏｒ　Ｖ））））。ｂｇ命令の直前にｃｍｐ命令が実行された場合、ｂｇ命令は、オペランドＡの値がオペランドＢの値より大きい場合（Ａ＞Ｂ）にジャンプしそれ以外の場合（Ａ≦Ｂ）にジャンプしないことを示す。

【００６９】

ｂｇｅ命令は、ネガティブフラグとオーバフローフラグの排他的論理和の否定（ｎｏｔ（Ｎ　ｘｏｒ　Ｖ））が１の場合にオペランドｌａｂｅｌが示す命令にジャンプし、排他的論理和の否定が０の場合にはジャンプしない条件分岐命令である。ｂｇｅ命令の直前にｃｍｐ命令が実行された場合、ｂｇｅ命令は、オペランドＡの値がオペランドＢの値以上の場合（Ａ≧Ｂ）にジャンプしそれ以外の場合（Ａ＜Ｂ）にジャンプしないことを示す。

【００７０】

ｂａ命令は、オペランドｌａｂｅｌが示す命令に常にジャンプする無条件分岐命令である。ｂａ命令の直前にｃｍｐ命令が実行されても、ｂａ命令は比較結果に依存しない。

次に、マシン依存最適化部１４５によって行われる最適化について説明する。

【００７１】

図７は、連続する分岐命令を含むプログラムの例を示す図である。

中間コード生成部１３３は、コード１６１をコード１６２に変換し、コード１６２をコード１６３に変換する。図７では、コード１６１，１６２をソースコード形式で記述しており、コード１６３を疑似アセンブラ形式で記述している。実際には、コード１６３は、コンパイル装置１００の内部で使用される中間言語を用いて記述される。マシン依存最適化部１４５は、中間コードに対して最適化を実行することになる。

【００７２】

コード１６１には、文字型の変数ｃと論理型の変数ｂを引数として受け取る関数ｆｏｏが定義されている。変数ｃは１文字を示す。ただし、文字は所定バイト数の文字コードで表現されることから、内部では変数ｃの値は整数である。変数ｂはｔｒｕｅまたはｆａｌ

ｓｅを示す。ただし、内部ではｆａｌｓｅは０で表現され、ｔｒｕｅは０以外の整数で表現される。関数ｆｏｏは、変数ｃの示す文字が所定の複数の文字の何れとも一致せず、かつ、変数ｂの値がｆａｌｓｅであるか判定する。所定の複数の文字は、改行、スペース、タブ、復帰、ダブルクォテーション（”）、バックスラッシュ（＼）、スラッシュ（／）、セミコロン（；）、左括弧（｛）および右括弧（｝）である。関数ｆｏｏは、上記条件を満たす場合は処理Ｘを実行し、上記条件を満たさない場合は処理Ｙを実行する。

【００７３】

コード１６２には、コード１６１と同様の関数ｆｏｏが定義されている。ただし、コード１６２では、変数ｃの値と比較される文字が文字コードを示す整数に置換されている。「１０」は改行を示し、「３２」はスペースを示し、「９」はタブを示し、「１３」は復帰を示す。「３４」はダブルクォテーションを示し、「３９」はバックスラッシュを示し、「４７」はスラッシュを示し、「５９」はセミコロンを示し、「１２３」は左括弧を示し、「１２５」は右括弧を示す。これら文字コードに相当する１０個の整数が、変数ｃの値と比較される比較値になる。また、コード１６２では、変数ｂの値と比較される論理値が整数に置換されている。「０」はｆａｌｓｅを示す。

【００７４】

コード１６３は、コード１６２の処理を、ＣＰＵ２０がサポートするｃｍｐ命令、ｂｅｑ命令、ｂｎｅ命令およびｌｏａｄ命令を用いて表現したものである。ｌｏａｄ命令は、メモリから汎用レジスタにデータをロードする命令である。コード１６３は、１１個のｃｍｐ命令、１０個のｂｅｑ命令、１個のｂｎｅ命令および１個のｌｏａｄ命令を含む。

【００７５】

コード１６３では、ラベルＬ１において、変数ｃの値と「１０」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ２において、変数ｃの値と「３２」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ３において、変数ｃの値と「９」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。

【００７６】

次に、ラベルＬ４において、変数ｃの値と「１３」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ５において、メモリから変数ｂの値をロードするｌｏａｄ命令が実行され、変数ｂの値と「０」とを比較するｃｍｐ命令が実行され、両者が一致しない場合にラベルＬ１３にジャンプするｂｎｅ命令が実行される。

【００７７】

次に、ラベルＬ６において、変数ｃの値と「３４」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ７において、変数ｃの値と「３９」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ８において、変数ｃの値と「４７」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。

【００７８】

次に、ラベルＬ９において、変数ｃの値と「５９」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ１０において、変数ｃの値と「１２３」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。次に、ラベルＬ７において、変数ｃの値と「１２５」とを比較するｃｍｐ命令が実行され、両者が一致する場合にラベルＬ１３にジャンプするｂｅｑ命令が実行される。

【００７９】

上記の何れの分岐命令（ｂｅｑ命令およびｂｎｅ命令）によってもジャンプしなかった場合、ラベルＬ１２において処理Ｘが実行される。上記の何れかの分岐命令によってジャ

ンプした場合、ラベルＬ１３において処理Ｙが実行される。すなわち、コード１６３では、変数ｃの値が「１０」，「３２」，「９」，「１３」，「３４」，「３９」，「４７」，「５９」，「１２３」，「１２５」の何れかと一致するか、または、変数ｂの値が「０」でない場合、処理Ｙが実行される。それ以外の場合は処理Ｘが実行される。このコード１６３の処理は、コード１６２が示す処理と等価である。

【００８０】

図８は、ブロック情報の例を示す図である。

マシン依存最適化部１４５が最適化を行うにあたり、上記のコード１６３から、図８に示すようなブロック情報が生成されて制御情報記憶部１２５に格納される。ブロック情報は、中間コードのブロック毎に生成される。１つのブロックは、連続的に実行されるひとまとまりの命令の集合を示す。分岐命令は、ブロックを区切る１つの基準となる。第２の実施の形態では、上記のコード１６３の各ラベルが１つのブロックに対応する。

【００８１】

ブロック情報１５２ａはラベルＬ１に対応し、ブロック情報１５２ｂはラベルＬ２に対応し、ブロック情報１５２ｃはラベルＬ３に対応する。ブロック情報１５２ｄはラベルＬ４に対応し、ブロック情報１５２ｅはラベルＬ５に対応し、ブロック情報１５２ｆはラベルＬ６に対応し、ブロック情報１５２ｇはラベルＬ７に対応する。ブロック情報１５２ｈはラベルＬ８に対応し、ブロック情報１５２ｉはラベルＬ９に対応し、ブロック情報１５２ｊはラベルＬ１０に対応し、ブロック情報１５２ｋはラベルＬ１１に対応する。

【００８２】

各ブロック情報は、ブロック名（項目＃Ａ）、比較値（項目＃Ｂ）、境界フラグ（項目＃Ｃ）、隙間フラグ（項目＃Ｄ）、分岐点フラグ（項目＃Ｅ）、連続フラグ（項目＃Ｆ）、真ブロック（項目＃Ｇ）および偽ブロック（＃Ｈ）を含む。また、各ブロック情報は、ｂｌフラグ（項目＃Ｉ）、ｂｇフラグ（項目＃Ｊ）、ｂｇｅフラグ（項目＃Ｋ）、ｂａフラグ（項目＃Ｌ）、前ポインタ（項目＃Ｍ）および次ポインタ（項目＃Ｎ）を含む。

【００８３】

ブロック名は、ブロックを識別する識別情報である。例えば、ブロック名にはラベルが用いられる。比較値は、変数の値と比較される固定値であり、ブロックに含まれるｃｍｐ命令の第２オペランドの値である。ただし、後述する連続フラグがｆａｌｓｅのブロックについては、比較値を－１に設定する。境界フラグは、後述する境界判定のブロックであるか否かを示す。隙間フラグは、後述する隙間判定のブロックであるか否かを示す。分岐点フラグは、後述する分岐点判定のブロックであるか否かを示す。境界判定、隙間判定および分岐点判定のブロックは、最適化を通じて中間コードに挿入される。境界フラグ、隙間グラフおよび分岐点フラグの初期値はｆａｌｓｅである。

【００８４】

連続フラグは、ブロックに含まれるｃｍｐ命令が他のブロックと同じ変数を第１オペランドに指定しているか、すなわち、同じ変数に対する連続する比較処理であるかを示す。前述のコード１６３の場合、ラベルＬ１～Ｌ４，Ｌ６～Ｌ１１のブロックは同じ変数ｃを比較対象とするため、連続フラグがｔｒｕｅになる。一方、ラベルＬ５のブロックは変数ｂを比較対象とするため、連続フラグがｆａｌｓｅになる。真ブロックは、分岐命令のジャンプ先の命令が属するブロックである。偽ブロックは、分岐命令でジャンプしない場合に次に実行される命令、すなわち、分岐命令の次の命令が属するブロックである。

【００８５】

ｂｌフラグは、ブロック末尾の分岐命令としてｂｌ命令が用いられるか否かを示す。ｂｌ命令は、後述する境界判定のブロックや隙間判定のブロックで使用されることがある。ｂｇフラグは、ブロック末尾の分岐命令としてｂｇ命令が用いられるか否かを示す。ｂｇ命令は、後述する境界判定のブロックで使用されることがある。ｂｇｅフラグは、ブロック末尾の分岐命令としてｂｇｅ命令が用いられるか否かを示す。ｂｇｅ命令は、後述する分岐点判定のブロックで使用されることがある。ｂｌフラグ、ｂｇフラグおよびｂｇｅフラグの初期値はｆａｌｓｅである。ｂａフラグは、ブロック末尾にｂａ命令が挿入される

か否かを示す。ｂａフラグの初期値はｆａｌｓｅである。
【００８６】
前ポインタは、そのブロックの１つ前に配置されるブロックを示す。先頭のブロックの前ポインタはＮＵＬＬである。次ポインタは、そのブロックの１つ後ろに配置されるブロックを示す。制御情報記憶部１２５には、ブロック情報の列の末尾を示すダミーのブロック情報が格納される。末尾のブロック情報の次ポインタはＮＵＬＬである。
【００８７】
例えば、ブロック情報１５２ａの生成当初の内容は、ブロック名がＬ１、比較値が１０、境界フラグがｆａｌｓｅ、隙間フラグがｆａｌｓｅ、分岐点フラグがｆａｌｓｅ、連続フラグがｔｒｕｅ、真ブロックがＬ１３、偽ブロックがＬ２である。また、ｂｌフラグがｆａｌｓｅ、ｂｇフラグがｆａｌｓｅ、ｂｇｅフラグがｆａｌｓｅ、ｂａフラグがｆａｌｓｅ、前ポインタがＮＵＬＬ、次ポインタがＬ２である。各ブロック情報の内容は、最適化方法を検討する過程で順次更新される。なお、以下では、境界フラグ、隙間フラグ、分岐点フラグ、ｂｌフラグ、ｂｇフラグ、ｂｇｅフラグおよびｂａフラグについては、値がｆａｌｓｅ（初期値）である場合はその記載を省略することがある。
【００８８】
図９は、比較値ベクトルと間隔マップと命令テーブルの例を示す図である。
マシン依存最適化部１４５が最適化を行う過程で、比較値ベクトル１５３、間隔マップ１５４および命令テーブル１５５が生成されて制御情報記憶部１２５に格納される。
【００８９】
比較値ベクトル１５３は、インデックスと比較値とを対応付ける。インデックスは、０から始まる連続する非負整数である。比較値は、同じ変数の値（例えば、コード１６３の変数ｃの値）と比較される整数の固定値である。他とは異なる変数の値（例えば、コード１６３の変数ｂの値）と比較される整数の固定値は、比較値から除外される。比較値ベクトル１５３では、複数の比較値が昇順にソートされる。比較値ベクトル１５３のサイズは、比較値の個数を示す。例えば、前述のコード１６３から、比較値「９」，「１０」，「１３」，「３２」，「３４」，「３９」，「４７」，「５９」，「１２３」，「１２５」を含むサイズ１０の比較値ベクトル１５３が生成される。
【００９０】
間隔マップ１５４は、間隔と比較値とを対応付ける。間隔は、ある比較値とその次に大きい比較値との間に存在する比較されない整数の個数である。間隔は、隣接する２つの比較値のうちの小さい方の比較値と対応付けられる。ただし、間隔マップ１５４には、間隔「０」は登録されない。間隔マップ１５４では、複数の間隔が降順にソートされる。
【００９１】
例えば、比較値「５９」と比較値「１２３」の間には６３個の整数が存在するため、間隔「６３」と比較値「５９」が登録される。また、比較値「１３」と比較値「３２」の間には１８個の整数が存在するため、間隔「１８」と比較値「１３」が登録される。同様にして、間隔「１１」と比較値「４７」、間隔「７」と比較値「３９」、間隔「４」と比較値「３４」、間隔「２」と比較値「１０」、間隔「１」と比較値「３２」、間隔「１」と比較値「１２３」が、間隔マップ１５４に登録される。
【００９２】
命令テーブル１５５は、比較値と命令種類とを対応付ける。命令種類は、ある比較値が属するブロックで使用される分岐命令の種類を示す。図９の例では、比較値「３９」とｂｇｅ命令、比較値「３４」とｂａ命令、比較値「１３」とｂｇｅ命令、比較値「１０」とｂａ命令、比較値「５９」とｂｇｅ命令、比較値「４７」とｂａ命令が対応付けられている。命令テーブル１５５の内容は、前述のブロック情報のｂｇｅフラグやｂａフラグに反映される。比較値と命令種類との対応付け方法については後述する。
【００９３】
図１０は、数値判定の最適化例を示す図である。
マシン依存最適化部１４５は、前述のコード１６３が示す制御構造を、変数ｃの値１つ

当たりの平均の比較回数が減少するように、図１０に示すような制御構造に変換する。変数ｃの値に対する一連の判定処理は、境界判定１６４ａ、隙間判定１６４ｂ、分岐点判定１６４ｃ，１６４ｄ，１６４ｅおよび個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏを含む。

【００９４】

個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏは、前述のコード１６３におけるラベルＬ１～Ｌ４，Ｌ６～Ｌ１１に対応する。すなわち、個別判定１６４ｆは、変数ｃの値が「９」であるか判定する。個別判定１６４ｇは、変数ｃの値が「１０」であるか判定する。個別判定１６４ｈは、変数ｃの値が「１３」であるか判定する。個別判定１６４ｉは、変数ｃの値が「３２」であるか判定する。個別判定１６４ｊは、変数ｃの値が「３４」であるか判定する。個別判定１６４ｋは、変数ｃの値が「３９」であるか判定する。個別判定１６４ｌは、変数ｃの値が「４７」であるか判定する。個別判定１６４ｍは、変数ｃの値が「５９」であるか判定する。個別判定１６４ｎは、変数ｃの値が「１２３」であるか判定する。個別判定１６４ｏは、変数ｃの値が「１２５」であるか判定する。

【００９５】

この一連の判定処理では、個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏの前に、境界判定１６４ａ、隙間判定１６４ｂおよび分岐点判定１６４ｃ，１６４ｄ，１６４ｅが挿入される。

【００９６】

境界判定１６４ａは、変数ｃの値が最小の比較値「９」より小さいかまたは最大の比較値「１２５」より大きいか判定する。境界判定１６４ａの判定結果が真である場合、変数ｃの値が何れの比較値とも一致しないことが明らかである。この場合、隙間判定１６４ｂ、分岐点判定１６４ｃ，１６４ｄ，１６４ｅおよび個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏをスキップして、処理Ｘにジャンプすることが可能である。

【００９７】

境界判定１６４ａの判定結果が偽である場合、隙間判定１６４ｂが実行される。隙間判定１６４ｂは、変数ｃの値が比較値「６０」以上でかつ比較値「１２３」より小さいか判定する。隙間判定１６４ｂの判定結果が真である場合、「５９」と「１２３」の間に他の比較値が存在しないため、変数ｃの値が何れの比較値とも一致しないことが明らかである。この場合、分岐点判定１６４ｃ，１６４ｄ，１６４ｅおよび個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏをスキップして、処理Ｘにジャンプすることが可能である。なお、図１０では最も広い間隔のみについて隙間判定を行っているが、２以上の間隔について隙間判定を行ってもよい。隙間判定を行うか否かの基準については後述する。

【００９８】

隙間判定１６４ｂの判定結果が偽である場合、分岐点判定１６４ｃが実行される。分岐点判定１６４ｃは、変数ｃの値が比較値「３９」以上であるか判定する。分岐点判定１６４ｃの判定結果が真である場合、次に分岐点判定１６４ｄが実行される。分岐点判定１６４ｃの判定結果が偽である場合、次に分岐点判定１６４ｅが実行される。

【００９９】

分岐点判定１６４ｄは、変数ｃの値が比較値「１３」以上であるか判定する。分岐点判定１６４ｄの判定結果が真である場合、個別判定１６４ｆにジャンプする。分岐点判定１６４ｄの判定結果が偽である場合、個別判定１６４ｈにジャンプする。分岐点判定１６４ｅは、変数ｃの値が比較値「５９」以上であるか判定する。分岐点判定１６４ｅの判定結果が真である場合、個別判定１６４ｋにジャンプする。分岐点判定１６４ｅの判定結果が偽である場合、個別判定１６４ｍにジャンプする。分岐点判定１６４ｃ，１６４ｄ，１６４ｅの判定方法は、二分探索アルゴリズムに相当する。

【０１００】

個別判定１６４ｆの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｆの判定結果が偽である場合、個別判定１６４ｇに進む。個別判定１６４ｇの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｇの判定結果が偽である場合、変数ｃの値が「１３」未満であることが判定済みであり「９」，「１０」以外の比較値と一致することがないため、処理Ｘにジャンプすることが可能である。

【０１０１】

個別判定１６４ｈの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｈの判定結果が偽である場合、個別判定１６４ｉに進む。個別判定１６４ｉの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｉの判定結果が偽である場合、個別判定１６４ｊに進む。個別判定１６４ｊの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｊの判定結果が偽である場合、変数ｃの値が「１３」以上「３９」未満であることが判定済みであり「１３」，「３２」，「３４」以外の比較値と一致することがないため、処理Ｘにジャンプすることが可能である。

【０１０２】

個別判定１６４ｋの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｋの判定結果が偽である場合、個別判定１６４ｌに進む。個別判定１６４ｌの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｌの判定結果が偽である場合、変数ｃの値が「３９」以上「５９」未満であることが判定済みであり「３９」，「４７」以外の比較値と一致することがないため、処理Ｘにジャンプすることが可能である。

【０１０３】

個別判定１６４ｍの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｍの判定結果が偽である場合、個別判定１６４ｎに進む。個別判定１６４ｎの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｎの判定結果が偽である場合、個別判定１６４ｏに進む。個別判定１６４ｏの判定結果が真である場合、処理Ｙにジャンプする。個別判定１６４ｏの判定結果が偽である場合、変数ｃの値が「５９」以上であることが判定済みであり「５９」，「１２３」，「１２５」以外の比較値と一致することがないため、処理Ｘにジャンプすることが可能である。

【０１０４】

なお、分岐点判定１６４ｃ以降の判定方法は、逐次的に実行される個別判定が含まれているため、純粋な二分探索アルゴリズムとは異なる。この点で、分岐点判定１６４ｃ以降の判定方法は、準二分探索アルゴリズムと言うこともできる。

【０１０５】

図１１は、最適化されたプログラムの例を示す図である。

マシン依存最適化部１４５は、図１０に示した制御構造を表すように前述のブロック情報を更新する。マシン依存最適化部１４５は、更新したブロック情報に基づいて中間コードを書き換える。コード１６５は、コード１６３を最適化したものである。図１１では、コード１６５を疑似アセンブラ形式で記述している。

【０１０６】

コード１６５では、他のブロックと比較する変数が異なるラベルＬ５のブロックが、先頭に移動している。ラベルＬ５のブロックの次に、ラベルＬ１４，Ｌ１５のブロックが挿入されている。ラベルＬ１４，Ｌ１５のブロックは、境界判定１６４ａに対応する。ラベルＬ１４では、変数ｃの値が「９」より小さいか判定し、判定結果が真の場合にラベルＬ１２にジャンプする。ラベルＬ１５では、変数ｃの値が「１２５」より大きいか判定し、判定結果が真の場合にラベルＬ１２にジャンプする。

【０１０７】

ラベルＬ１５のブロックの次に、ラベルＬ１６，Ｌ１７のブロックが挿入されている。ラベルＬ１６，Ｌ１７のブロックは、隙間判定１６４ｂに対応する。ラベルＬ１６では、変数ｃの値が「６０」より小さいか判定し、判定結果が真の場合にラベルＬ１８にジャンプする。ラベルＬ１７では、変数ｃの値が「１２３」より小さいか判定し、判定結果が真の場合にラベルＬ１２にジャンプする。

【０１０８】
ラベルＬ１７のブロックの次に、ラベルＬ１８のブロックが挿入されている。ラベルＬ１８のブロックは、分岐点判定１６４ｃに対応する。ラベルＬ１８では、変数ｃの値が「３９」以上か判定し、判定結果が真の場合にラベルＬ２０にジャンプする。ラベルＬ１８のブロックの次に、ラベルＬ１９のブロックが挿入されている。ラベルＬ１９のブロックは、分岐点判定１６４ｄに対応する。ラベルＬ１９では、変数ｃの値が「１３」以上か判定し、判定結果が真の場合にラベルＬ４にジャンプする。
【０１０９】
ラベルＬ１～１１のブロックは、比較値の昇順にソートされている。ラベルＬ１９のブロックの次に、ラベルＬ３のブロックが配置されている。ラベルＬ３のブロックは、個別判定１６４ｆに対応する。ラベルＬ３のブロックの次に、ラベルＬ１のブロックが配置されている。ラベルＬ１のブロックは、個別判定１６４ｇに対応する。ラベルＬ１のブロックの末尾には、ラベルＬ１２にジャンプするｂａ命令が挿入されている。
【０１１０】
ラベルＬ１のブロックの次に、ラベルＬ４のブロックが配置されている。ラベルＬ４のブロックは、個別判定１６４ｈに対応する。ラベルＬ４のブロックの次に、ラベルＬ２のブロックが配置されている。ラベルＬ２のブロックは、個別判定１６４ｉに対応する。ラベルＬ２のブロックの次に、ラベルＬ６のブロックが配置されている。ラベルＬ６のブロックは、個別判定１６４ｊに対応する。ラベルＬ６のブロックの末尾には、ラベルＬ１２にジャンプするｂａ命令が挿入されている。
【０１１１】
ラベルＬ６のブロックの次に、ラベルＬ２０のブロックが挿入されている。ラベルＬ２０のブロックは、分岐点判定１６４ｅに対応する。ラベルＬ２０では、変数ｃの値が「５９」以上か判定し、判定結果が真の場合にラベルＬ９にジャンプする。ラベルＬ２０のブロックの次に、ラベルＬ７のブロックが配置されている。ラベルＬ７のブロックは、個別判定１６４ｋに対応する。ラベルＬ７のブロックの次に、ラベルＬ８のブロックが配置されている。ラベルＬ８のブロックは、個別判定１６４ｌに対応する。ラベルＬ８のブロックの末尾には、ラベルＬ１２にジャンプするｂａ命令が挿入されている。
【０１１２】
ラベルＬ８のブロックの次に、ラベルＬ９のブロックが配置されている。ラベルＬ９のブロックは、個別判定１６４ｍに対応する。ラベルＬ９のブロックの次に、ラベルＬ１０のブロックが配置されている。ラベルＬ１０のブロックは、個別判定１６４ｎに対応する。ラベルＬ１０のブロックの次に、ラベルＬ１１のブロックが配置されている。ラベルＬ１１のブロックは、個別判定１６４ｏに対応する。そして、ラベルＬ１１のブロックの次に、ラベルＬ１２，Ｌ１３のブロックが配置されている。このように、最適化されたコード１６５は、図１０に示した制御構造をもっている。
【０１１３】
次に、コンパイラ１３１のコンパイル手順について説明する。
図１２は、コンパイルの手順例を示すフローチャートである。
（Ｓ１）ソースコード入力部１３２は、ソースコード記憶部１２１からソースコードを読み出す。中間コード生成部１３３は、読み出されたソースコードを解析し、ソースコードを中間コードに変換して中間コード記憶部１２２に格納する。ソースコードの解析には、字句解析、構文解析、意味解析などのフロントエンド処理が含まれる。
【０１１４】
（Ｓ２）最適化部１３４の汎用最適化部１４１は、中間コード記憶部１２２に記憶された中間コードに対して、汎用的な最適化を実行する。汎用的な最適化には、使用されていない変数の削除、静的に決定できる値のみに依存する計算式の定数化、複数の計算式に共通に含まれる部分式の計算結果の再利用などが含まれる。
【０１１５】
（Ｓ３）最適化部１３４のループ最適化部１４２は、中間コード記憶部１２２に記憶さ

れた中間コードからループを検出し、ループの最適化を実行する。ループの最適化には、ループ内で値が変わらない変数に関する演算をループ外に移動することや、ループ内の命令を展開（アンローリング）して、ループの繰り返し数を減らすことなどが含まれる。

【０１１６】

（Ｓ４）最適化部１３４のＳＩＭＤ最適化部１４３は、中間コード記憶部１２２に記憶された中間コードに対して、ＳＩＭＤ命令を用いた最適化を行う。すなわち、ＳＩＭＤ最適化部１４３は、中間コードの中から依存関係がなく演算の種類が同じスカラ命令の組を検出し、検出したスカラ命令の組をＳＩＭＤ命令に変換する。

【０１１７】

（Ｓ５）最適化部１３４の命令変換部１４４は、中間コード記憶部１２２に記憶された中間コードで使用されている命令形式を、ターゲットのＣＰＵアーキテクチャがサポートする命令セットの命令形式に変換する。コンパイルコマンドのオプションなどによってユーザからターゲットのＣＰＵアーキテクチャが指定された場合、命令変換部１４４は、中間コードの命令を指定されたＣＰＵアーキテクチャの命令形式に変換する。

【０１１８】

（Ｓ６）最適化部１３４のマシン依存最適化部１４５は、ＣＰＵ依存の命令の種類を活用した最適化を行う。マシン依存最適化部１４５は特に、１つの変数の値と複数の比較値とを比較し比較結果に応じて処理が分岐する制御構造を最適化する。以下では、このような制御構造を最適化するマシン依存最適化を中心に説明する。

【０１１９】

（Ｓ７）最適化部１３４の命令スケジューリング部１４８は、中間コード記憶部１２２に記憶された中間コードに対して命令のスケジューリングを行う。スケジューリングには、命令の並列化や、パイプライン処理を考慮した命令の順序の入れ替えなどが含まれる。

【０１２０】

（Ｓ８）アセンブリコード生成部１３５は、中間コード記憶部１２２から最適化後の中間コードを読み出し、中間コードからアセンブリコードを生成する。

（Ｓ９）オブジェクトコード出力部１３６は、ステップＳ８で生成されたアセンブリコードをオブジェクトコードに変換し、オブジェクトコード記憶部１２３に格納する。生成されたオブジェクトコードは、リンカ１３７によって他のオブジェクトコードやライブラリとリンクされ、実行ファイルに変換されることになる。

【０１２１】

図１３は、マシン依存最適化の手順例を示すフローチャートである。

マシン依存最適化は、前述のステップＳ６で実行される。

（Ｓ１０）解析部１４６は、中間コードの中の最初の命令を命令ｉとして選択する。

【０１２２】

（Ｓ１１）解析部１４６は、命令ｉとして選択する命令が存在しないか（命令ｉがＮＵＬＬであるか）判断する。命令ｉがＮＵＬＬである場合はマシン依存最適化が終了する。命令ｉがＮＵＬＬでない場合はステップＳ１２に処理が進む。

【０１２３】

（Ｓ１２）解析部１４６は、命令ｉが比較命令（ｃｍｐ命令）であるか判断する。命令ｉが比較命令である場合はステップＳ１４に処理が進み、命令ｉが比較命令でない場合はステップＳ１３に処理が進む。

【０１２４】

（Ｓ１３）解析部１４６は、現在の命令ｉの次の命令を新たな命令ｉとして選択する。そして、ステップＳ１１に処理が進む。

（Ｓ１４）解析部１４６は、命令ｉの次の命令を命令ｊとして選択する。

【０１２５】

（Ｓ１５）解析部１４６は、命令ｊが分岐命令（ｂｅｑ命令やｂｎｅ命令など）であるか判断する。命令ｊが分岐命令である場合はステップＳ１７に処理が進み、命令ｊが分岐命令でない場合はステップＳ１６に処理が進む。

【０１２６】
（Ｓ１６）解析部１４６は、命令ｊの次の命令を新たな命令ｉとして選択する。そして、ステップＳ１１に処理が進む。
（Ｓ１７）解析部１４６は、命令ｉから始まる一連の比較処理および分岐処理を中間コードの中から検出し、検出した一連の比較処理および分岐処理についてのブロック情報を生成する。ブロック情報生成の詳細は後述する。
【０１２７】
（Ｓ１８）解析部１４６は、ステップＳ１７で生成したブロック情報に基づいて比較値ベクトル１５３を生成し、比較値ベクトル１５３に基づいて間隔マップ１５４を生成する。ベクトル・マップ生成の詳細は後述する。
【０１２８】
（Ｓ１９）解析部１４６は、ステップＳ１８で生成した間隔マップ１５４に基づいて、制御構造を変更した場合の比較パターン数を、隙間判定の個数を変化させながら試算する。解析部１４６は、比較パターン数が最小になる隙間判定の個数とそのときの比較パターン数を特定する。また、解析部１４６は、制御構造を変更しない場合の比較パターン数を試算する。解析部１４６は、制御構造を変更する前後の比較パターン数を比べ、制御構造を変更することで実行効率が改善されるか判定する。効率評価の詳細は後述する。
【０１２９】
（Ｓ２０）ステップＳ１９で効率改善ありと判定された場合、ステップＳ２２に処理が進む。効率改善ありと判定されなかった場合、ステップＳ２１に処理が進む。
（Ｓ２１）解析部１４６は、ステップＳ１７で生成したブロック情報と、ステップＳ１８で生成した比較値ベクトル１５３および間隔マップ１５４を、制御情報記憶部１２５から消去する。そして、ステップＳ１１に処理が進む。
【０１３０】
（Ｓ２２）変換部１４７は、個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏに先行して、境界判定１６４ａおよび隙間判定１６４ｂが実行されるように、ブロック情報の追加および更新を行う。先行判定追加の詳細は後述する。
【０１３１】
（Ｓ２３）変換部１４７は、境界判定１６４ａおよび隙間判定１６４ｂの後、分岐点判定１６４ｃ，１６４ｄ，１６４ｅが実行されるように、ブロック情報の追加および更新を行う。また、変換部１４７は、個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｊ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎ，１６４ｏの比較処理が、二分探索木に準じた制御構造をもつようにブロック情報の更新を行う。探索木変換の詳細は後述する。
【０１３２】
（Ｓ２４）変換部１４７は、ステップＳ２２，Ｓ２３で更新されたブロック情報に基づいて中間コードを書き換える。そして、ステップＳ１１に処理が進む。
図１４は、ブロック情報生成の手順例を示すフローチャートである。
【０１３３】
ブロック情報生成は、前述のステップＳ１７で実行される。
（Ｓ３０）解析部１４６は、命令ｉの第１オペランドを変数ｃとして抽出する。また、解析部１４６は、命令ｊのオペランドをジャンプ先ｌａｂｅｌとして抽出する。
【０１３４】
（Ｓ３１）解析部１４６は、命令ｉとして選択する命令が存在しないか（命令ｉがＮＵＬＬであるか）判断する。命令ｉがＮＵＬＬである場合はブロック情報生成が終了する。命令ｉがＮＵＬＬでない場合はステップＳ３２に処理が進む。
【０１３５】
（Ｓ３２）解析部１４６は、命令ｉが比較命令（ｃｍｐ命令）であるか判断する。命令ｉが比較命令である場合はステップＳ３３に処理が進み、命令ｉが比較命令でない場合はステップＳ３８に処理が進む。

【０１３６】
（Ｓ３３）解析部１４６は、命令ｉの第１オペランドがステップＳ３０で抽出した変数ｃと同じか判断する。命令ｉの第１オペランドが変数ｃと同じ場合はステップＳ３４に処理が進み、異なる場合はステップＳ３８に処理が進む。
【０１３７】
（Ｓ３４）解析部１４６は、命令ｉの次の命令を命令ｊとして選択する。
（Ｓ３５）解析部１４６は、命令ｊとして選択する命令が存在しないか（命令ｊがＮＵＬＬであるか）判断する。命令ｊがＮＵＬＬである場合はステップＳ４４に処理が進む。命令ｊがＮＵＬＬでない場合はステップＳ３６に処理が進む。
【０１３８】
（Ｓ３６）解析部１４６は、命令ｊが分岐命令（ｂｅｑ命令やｂｎｅ命令など）であるか判断する。命令ｊが分岐命令である場合はステップＳ３７に処理が進み、命令ｊが分岐命令でない場合はステップＳ４４に処理が進む。
【０１３９】
（Ｓ３７）解析部１４６は、命令ｊのオペランドがステップＳ３０で抽出したジャンプ先ｌａｂｅｌと同じか判断する。命令ｊのオペランドがジャンプ先ｌａｂｅｌと同じ場合はステップＳ４２に処理が進み、異なる場合はステップＳ４４に処理が進む。
【０１４０】
（Ｓ３８）解析部１４６は、命令ｉが属するブロックの末尾が分岐命令であるか判断する。末尾が分岐命令である場合はステップＳ３９に処理が進み、末尾が分岐命令でない場合はステップＳ４４に処理が進む。
【０１４１】
（Ｓ３９）解析部１４６は、命令ｉが属するブロックの末尾の分岐命令のオペランドが、ステップＳ３０で抽出したジャンプ先ｌａｂｅｌと同じか判断する。末尾の分岐命令のオペランドがジャンプ先ｌａｂｅｌと同じ場合はステップＳ４０に処理が進み、異なる場合はステップＳ４４に処理が進む。
【０１４２】
（Ｓ４０）解析部１４６は、命令ｉが属するブロック内に比較命令が存在するか判断する。比較命令が存在する場合はステップＳ４１に処理が進み、比較命令が存在しない場合はステップＳ４４に処理が進む。
【０１４３】
（Ｓ４１）解析部１４６は、現在の命令ｉが属するブロック内の比較命令を新たな命令ｉとして選択する。また、解析部１４６は、命令ｉが属するブロックの末尾にある分岐命令を新たな命令ｊとして選択する。
【０１４４】
（Ｓ４２）解析部１４６は、命令ｉ，ｊに基づいて、命令ｉ，ｊが属するブロックに対応するブロック情報を生成する。ブロック名（項目＃Ａ）は、当該ブロックのラベルである。比較値（項目＃Ｂ）は、命令ｉの第２オペランドである。境界フラグ（項目＃Ｃ）、隙間フラグ（項目＃Ｄ）および分岐点フラグ（項目＃Ｅ）は、ｆａｌｓｅである。連続フラグ（項目＃Ｆ）は、ステップＳ４１を経由していない場合はｔｒｕｅ、ステップＳ４１を経由した場合はｆａｌｓｅである。真ブロック（項目＃Ｇ）は、命令ｊのオペランドである。偽ブロック（項目＃Ｈ）は、当該ブロックの次のブロックのラベルである。
【０１４５】
ｂｌフラグ（項目＃Ｉ）、ｂｇフラグ（項目＃Ｊ）、ｂｇｅフラグ（項目＃Ｋ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、当該ブロックの前のブロックのラベルである。ただし、前のブロックがない場合、前ポインタはＮＵＬＬである。次ポインタ（項目＃Ｎ）は、当該ブロックの次のブロックのラベルである。
【０１４６】
（Ｓ４３）解析部１４６は、現在の命令ｊの次の命令を新たな命令ｉとして選択する。また、解析部１４６は、新たな命令ｉの次の命令を新たな命令ｊとして選択する。そして

、ステップＳ３１に処理が進む。

【０１４７】

（Ｓ４４）解析部１４６は、末尾のブロック情報を生成する。末尾のブロック情報のブロック名（項目＃Ａ）は、命令ｉが属するブロック（一連の比較処理および分岐処理を行うブロック群の次のブロック）のラベルである。連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、当該ブロックの前のブロックのラベルである。次ポインタ（項目＃Ｎ）は、ＮＵＬＬである。他の項目は空欄でもよい。

【０１４８】

図１５は、ベクトル・マップ生成の手順例を示すフローチャートである。

ベクトル・マップ生成は、前述のステップＳ１８で実行される。

（Ｓ５０）解析部１４６は、比較値ベクトル１５３を生成する。

【０１４９】

（Ｓ５１）解析部１４６は、先頭のブロック情報をブロック情報ｂとして選択する。

（Ｓ５２）解析部１４６は、ブロック情報ｂの次ポインタ（項目＃Ｎ）がＮＵＬＬであるか判断する。次ポインタがＮＵＬＬである場合はステップＳ５６に処理が進み、次ポインタがＮＵＬＬでない場合はステップＳ５３に処理が進む。

【０１５０】

（Ｓ５３）解析部１４６は、ブロック情報ｂの連続フラグ（項目＃Ｆ）がｔｒｕｅであるか判断する。連続フラグがｔｒｕｅである場合はステップＳ５４に処理が進み、連続フラグがｆａｌｓｅである場合はステップＳ５５に処理が進む。

【０１５１】

（Ｓ５４）解析部１４６は、ブロック情報ｂの比較値（項目＃Ｂ）を比較値ベクトル１５３に追加する。

（Ｓ５５）解析部１４６は、ブロック情報ｂの次ポインタ（項目＃Ｎ）が示すブロック情報を次のブロック情報ｂとして選択する。そして、ステップＳ５２に処理が進む。

【０１５２】

（Ｓ５６）解析部１４６は、比較値ベクトル１５３の比較値を昇順にソートする。

（Ｓ５７）解析部１４６は、インデックスｐに０を代入する。

（Ｓ５８）解析部１４６は、インデックスｐが、比較値ベクトル１５３のサイズから１を引いた値より小さいか判断する。上記条件を満たす場合はステップＳ５９に処理が進み、満たさない場合はステップＳ６３に処理が進む。

【０１５３】

（Ｓ５９）解析部１４６は、比較値ベクトル１５３からインデックスｐが示す比較値とインデックスｐ＋１が示す比較値とを取得する。解析部１４６は、間隔ｄｉｆとして、インデックスｐ＋１の比較値からインデックスｐの比較値を引き１を引いた値を算出する。

【０１５４】

（Ｓ６０）解析部１４６は、間隔ｄｉｆが０より大きいか判断する。上記条件を満たす場合はステップＳ６１に処理が進み、満たさない場合はステップＳ６２に処理が進む。

（Ｓ６１）解析部１４６は、間隔ｄｉｆとインデックスｐの比較値とを対応付けて、間隔マップ１５４に追加する。間隔ｄｉｆは、間隔マップ１５４のキーに相当する。

【０１５５】

（Ｓ６２）解析部１４６は、インデックスｐに１を加算する（インデックスｐをインクリメントする）。そして、ステップＳ５８に処理が進む。

（Ｓ６３）解析部１４６は、間隔マップ１５４の間隔の降順にソートする。

【０１５６】

図１６は、効率評価の手順例を示すフローチャートである。

効率評価は、前述のステップＳ１９で実行される。

（Ｓ７０）解析部１４６は、区間数ｘに０を代入する。また、解析部１４６は、変換後の制御構造の比較パターン数を示す評価値ａｆｔｅｒに、整数型の最大値を代入する。例えば、解析部１４６は、評価値ａｆｔｅｒに、符号なしｌｏｎｇ（ｕｎｓｉｇｎｅｄ　ｌ

ｏｎｇ）型整数の最大値である４２９４９６７２９５を代入する。

【０１５７】

（Ｓ７１）解析部１４６は、区間数ｘが間隔マップ１５４のサイズより小さいか判断する。区間数ｘが間隔マップ１５４のサイズより小さい場合はステップＳ７２に処理が進み、それ以外の場合はステップＳ７７に処理が進む。

【０１５８】

（Ｓ７２）解析部１４６は、間隔マップ１５４が示す比較値の間の区間のうち、ｘ個の区間について隙間判定を行うと仮定する。このとき、解析部１４６は、間隔が大きい方から優先的にｘ個の区間を選択するようにする。

【０１５９】

（Ｓ７３）解析部１４６は、変換後の制御構造の比較パターン数であって、ｘ個の区間について隙間判定を行う場合の比較パターン数を算出する。

ここで、ｋを比較値の個数（比較値ベクトル１５３のサイズ）とする。図７の例の場合、ｋ＝１０である。ｄを二分木の深さとする。二分木の深さは、分岐点判定の段数に相当し、図１０の例ではｄ＝２である。ｄの値は予め決められていてもよいし、比較値の数に応じて可変としてもよい。また、K_{max}を変数ｃの型の最大値、K_{min}を変数ｃの型の最小値とする。図７の例の場合、変数ｃはｃｈａｒ型であるため、K_{max}＝１２７，K_{min}＝－１２８である。また、k_{max}を比較値の最大値、k_{min}を比較値の最小値とする。図７の例の場合、k_{max}＝１２５，k_{min}＝９である。

【０１６０】

また、$P = K_{max} - K_{min} + 1$，$K = k_{max} - k_{min} + 1$，$m = k / 2^d$の商，$n = k / 2^d$の余りとする。図７の例の場合、Ｐ＝２５６，Ｋ＝１１７，ｍ＝２，ｎ＝２である。また、ｗを、間隔マップ１５４に登録された間隔を降順にソートした配列とする。

【０１６１】

解析部１４６は、上記のパラメータの値を数式（１）に代入することで、変換後の制御構造の比較パターン数を算出することができる。数式（１）によって算出される比較パターン数は、変数ｃが取り得る全ての整数が入力されたときの比較回数の合計を示す。なお、ｔ，ｙ，ｚは数式内で値が変化する一時的な変数である。

【０１６２】

【数１】

$$2P + 2\left(K \cdot x - \sum_{z=0}^{x-2}\sum_{y=0}^{z} w[y]\right) + \left(K - \sum_{y=0}^{x-1} w[y]\right) d \qquad (1)$$
$$+ \sum_{t=0}^{m-1}\left(K - \sum_{y=0}^{x-1} w[y] - 2^d \cdot t\right) + \frac{n}{2^d}\left(K - \sum_{y=0}^{x-1} w[y] - 2^d \cdot m\right)$$

【０１６３】

数式（１）の第１項は、境界判定で行われる比較の回数を示す。数式（１）の第２項は、境界判定の判定結果が偽になった整数について、隙間判定で行われる比較の回数を示す。隙間判定およびそれ以降の判定処理の比較回数は、区間数ｘに依存する。数式（１）の第３項は、隙間判定の判定結果が偽になった整数について、分岐点判定で行われる比較の回数を示す。数式（１）の第４項は、個別判定のうち全経路に共通の深さまでの個別判定で行われる比較の回数を示す。図１０の例の場合、全経路に共通の個別判定の深さは２であり、第４項は、個別判定１６４ｆ，１６４ｇ，１６４ｈ，１６４ｉ，１６４ｋ，１６４ｌ，１６４ｍ，１６４ｎの比較回数を示す。数式（１）の第５項は、残りの個別判定で行われる比較の回数を示す。図１０の例の場合、第５項は、第４項でカウントされなかった個別判定１６４ｊ，１６４ｏの比較回数を示す。

【０１６４】

（Ｓ７４）解析部１４６は、ステップＳ７３で算出された比較パターン数が評価値ａｆｔｅｒより小さいか判断する。算出された比較パターン数が評価値ａｆｔｅｒより小さい場合、ステップＳ７５に処理が進む。算出された比較パターン数が評価値ａｆｔｅｒ以上である場合、ステップＳ７６に処理が進む。

【０１６５】

（Ｓ７５）解析部１４６は、ステップＳ７３で算出された比較パターン数を評価値ａｆｔｅｒに代入し、更新した評価値ａｆｔｅｒと対応付けて区間数ｘを記憶しておく。

（Ｓ７６）解析部１４６は、区間数ｘに１を加算する（区間数ｘをインクリメントする）。そして、ステップＳ７１に処理が進む。以上のステップＳ７１～Ｓ７６によって、評価値ａｆｔｅｒが最小になる区間数ｘとそのときの評価値ａｆｔｅｒが決定される。

【０１６６】

なお、図７の例の場合、区間数ｘ＝０に対応する比較パターン数は５１２＋２３４＋２３０＋５４＝１０３０である。区間数ｘ＝１に対応する比較パターン数は５１２＋２３４＋１０８＋１０４＋２３＝９８１である。区間数ｘ＝２に対応する比較パターン数は５１２＋３４２＋７２＋６８＋１４＝１００８である。区間数ｘ＝３に対応する比較パターン数は５１２＋４１４＋５０＋４６＋８＝１０３０である。区間数ｘ＝４に対応する比較パターン数は５１２＋４６４＋３６＋３２＋５＝１０４９である。区間数ｘ＝５に対応する比較パターン数は５１２＋５００＋２８＋２４＋３＝１０６７である。区間数ｘ＝６に対応する比較パターン数は５１２＋５２８＋２４＋２０＋２＝１０８６である。

【０１６７】

以上から、評価値ａｆｔｅｒ＝９８１，区間数ｘ＝２と決定される。上記のように、区間数ｘを増やすと、分岐点判定や個別判定に到達する整数が少なくなり、分岐点判定や個別判定の比較回数を削減することができる。一方で、区間数ｘを増やすと、隙間判定自体の比較回数が増加する。そのため、評価値ａｆｔｅｒを最小にする区間数ｘは、０（隙間判定なし）と最大値（全区間について隙間判定あり）の間になることがある。

【０１６８】

（Ｓ７７）解析部１４６は、変換前の制御構造の比較パターン数を算出し、算出された比較パターン数を評価値ｂｅｆｏｒｅに代入する。変換前の制御構造の比較パターン数は、上記のＰ，ｋを数式（２）に代入することで算出できる。数式（２）によって算出される比較パターン数は、変数ｃが取り得る全ての整数が入力されたときの比較回数の合計を示す。なお、ｙは数式内で値が変化する一時的な変数である。

【０１６９】

【数２】

$$\sum_{y=0}^{k-1}(P-y) \qquad (2)$$

【０１７０】

（Ｓ７８）解析部１４６は、評価値ａｆｔｅｒが評価値ｂｅｆｏｒｅよりも小さいか判断する（ａｆｔｅｒ＜ｂｅｆｏｒｅ）。評価値ａｆｔｅｒが評価値ｂｅｆｏｒｅよりも小さい場合はステップＳ７９に処理が進み、評価値ａｆｔｅｒが評価値ｂｅｆｏｒｅ以上である場合はステップＳ８０に処理が進む。

【０１７１】

（Ｓ７９）解析部１４６は、制御構造を変換することで実行効率が改善する（効率改善あり）と判定する。そして、効率評価が終了する。

（Ｓ８０）解析部１４６は、制御構造を変換しても実行効率は改善しない（効率改善なし）と判定する。なお、評価値ａｆｔｅｒが評価値ｂｅｆｏｒｅより小さい場合であっても、両者の差が閾値未満である場合には効率改善なしと判定するようにしてもよい。

【０１７２】

図１７は、先行判定追加の手順例を示すフローチャートである。

先行判定追加は、前述のステップＳ２２で実行される。
（Ｓ１１０）変換部１４７は、先頭のブロック情報をブロック情報ｂとして選択する。
【０１７３】
（Ｓ１１１）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）がＮＵＬＬであるか判断する。次ポインタがＮＵＬＬである場合はステップＳ１１５に処理が進み、次ポインタがＮＵＬＬでない場合はステップＳ１１２に処理が進む。
【０１７４】
（Ｓ１１２）変換部１４７は、ブロック情報ｂの連続フラグ（項目＃Ｆ）がｔｒｕｅであるか判断する。連続フラグがｔｒｕｅである場合はステップＳ１１４に処理が進み、連続フラグがｆａｌｓｅである場合はステップＳ１１３に処理が進む。
【０１７５】
（Ｓ１１３）変換部１４７は、ブロック情報ｂを先頭に移動する。すなわち、変換部１４７は、ブロック情報ｂの前ポインタ（項目＃Ｍ）をＮＵＬＬに更新し、ブロック情報ｂの次ポインタ（項目＃Ｎ）を元の先頭のブロック情報のブロック名（項目＃Ａ）に更新する。また、変換部１４７は、元の先頭のブロック情報の前ポインタ（項目＃Ｍ）、ブロック情報ｂの前のブロック情報の次ポインタ（項目＃Ｎ）、ブロック情報ｂの後ろのブロック情報の前ポインタ（項目＃Ｍ）に、順序変更を反映させる。
【０１７６】
（Ｓ１１４）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）が示すブロック情報を次のブロック情報ｂとして選択する。ただし、ステップＳ１１３でブロック情報ｂを移動した場合には、変換部１４７は、移動前の次ポインタ（項目＃Ｎ）が示すブロック情報を選択する。そして、ステップＳ１１１に処理が進む。
【０１７７】
（Ｓ１１５）変換部１４７は、先頭のブロック情報をブロック情報ｂとして選択する。
（Ｓ１１６）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）がＮＵＬＬであるか判断する。次ポインタがＮＵＬＬである場合はステップＳ１２１に処理が進み、次ポインタがＮＵＬＬでない場合はステップＳ１１７に処理が進む。
【０１７８】
（Ｓ１１７）変換部１４７は、ブロック情報ｂの連続フラグ（項目＃Ｆ）がｔｒｕｅであるか判断する。連続フラグがｔｒｕｅである場合はステップＳ１１９に処理が進み、連続フラグがｆａｌｓｅである場合はステップＳ１１８に処理が進む。
【０１７９】
（Ｓ１１８）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）が示すブロック情報を次のブロック情報ｂとして選択する。そして、ステップＳ１１６に処理が進む。
（Ｓ１１９）変換部１４７は、ブロック情報ｂの前に新たなブロック情報を追加する。ここで追加するブロック情報は、境界判定において変数ｃの値と比較値の最小値（下限境界）とを比較するブロックのブロック情報である。
【０１８０】
ブロック名（項目＃Ａ）は、既存のラベルと重複しない新たなラベルである。比較値（項目＃Ｂ）は、最小の比較値（k_{min}）である。境界フラグ（項目＃Ｃ）は、ｔｒｕｅである。隙間フラグ（項目＃Ｄ）、分岐点フラグ（項目＃Ｅ）および連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。真ブロック（項目＃Ｇ）は、末尾のブロック情報のブロック名である。偽ブロック（項目＃Ｈ）は、以下のステップＳ１２０で追加されるブロック情報のブロック名である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。ｂｇフラグ（項目＃Ｊ）、ｂｇｅフラグ（項目＃Ｋ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、１つ前のブロック情報のブロック名である。次ポインタ（項目＃Ｎ）は、ステップＳ１２０で追加されるブロック情報のブロック名である。また、１つ前のブロック情報の次ポインタ（項目＃Ｎ）を更新する。
【０１８１】
（Ｓ１２０）変換部１４７は、ステップＳ１１９のブロック情報の次に、新たなブロッ

ク情報を追加する。ここで追加するブロック情報は、境界判定において変数ｃの値と比較値の最大値（上限境界）とを比較するブロックのブロック情報である。

【０１８２】

ブロック名（項目＃Ａ）は、既存のラベルと重複しない新たなラベルである。比較値（項目＃Ｂ）は、最大の比較値（k_{max}）である。境界フラグ（項目＃Ｃ）は、ｔｒｕｅである。隙間フラグ（項目＃Ｄ）、分岐点フラグ（項目＃Ｅ）および連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。真ブロック（項目＃Ｇ）は、末尾のブロック情報のブロック名である。偽ブロック（項目＃Ｈ）は、ブロック情報ｂのブロック名である。ｂｇフラグ（項目＃Ｊ）は、ｔｒｕｅである。ｂｌフラグ（項目＃Ｉ）、ｂｇｅフラグ（項目＃Ｋ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、ステップＳ１１９で追加したブロック情報のブロック名である。次ポインタ（項目＃Ｎ）は、ブロック情報ｂのブロック名である。変換部１４７は、ブロック情報ｂの前ポインタ（項目＃Ｍ）を更新する。そして、ステップＳ１２１に処理が進む。

【０１８３】

図１８は、先行判定追加の手順例を示すフローチャート（続き）である。

（Ｓ１２１）変換部１４７は、先頭のブロック情報をブロック情報ｂとして選択する。

（Ｓ１２２）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）がＮＵＬＬであるか判断する。次ポインタがＮＵＬＬである場合、先行判定追加が終了する。次ポインタがＮＵＬＬでない場合、ステップＳ１２３に処理が進む。

【０１８４】

（Ｓ１２３）変換部１４７は、ブロック情報ｂの境界フラグ（項目＃Ｃ）がｆａｌｓｅ、かつ、連続フラグ（項目＃Ｆ）がｔｒｕｅであるか判断する。上記条件を満たす場合はステップＳ１２５に処理が進み、満たさない場合はステップＳ１２４に処理が進む。

【０１８５】

（Ｓ１２４）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）が示すブロック情報を次のブロック情報ｂとして選択する。そして、ステップＳ１２２に処理が進む。

（Ｓ１２５）変換部１４７は、インデックスｐに０を代入する。

【０１８６】

（Ｓ１２６）変換部１４７は、インデックスｐが、前述の効率評価で決定された隙間判定の区間数ｘ未満であるか判断する。インデックスｐが区間数ｘ未満である場合はステップＳ１２７に処理が進み、それ以外の場合は先行判定追加が終了する。

【０１８７】

（Ｓ１２７）変換部１４７は、間隔マップ１５４からｐ＋１番目の比較値を抽出する。

（Ｓ１２８）変換部１４７は、ブロック情報ｂの前に新たなブロック情報を追加する。ここで追加するブロック情報は、隙間判定において変数ｃの値と隙間の下限とを比較するブロックのブロック情報である。

【０１８８】

ブロック名（項目＃Ａ）は、既存のラベルと重複しない新たなラベルである。比較値（項目＃Ｂ）は、ステップＳ１２７で抽出した比較値に１を加えた整数である。隙間フラグ（項目＃Ｄ）は、ｔｒｕｅである。境界フラグ（項目＃Ｃ）、分岐点フラグ（項目＃Ｅ）および連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。真ブロック（項目＃Ｇ）は、以下のステップＳ１２９で追加されるブロック情報の１つ後ろのブロック情報のブロック名である。偽ブロック（項目＃Ｈ）は、ステップＳ１２９で追加されるブロック情報のブロック名である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。ｂｇフラグ（項目＃Ｊ）、ｂｇｅフラグ（項目＃Ｋ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、１つ前のブロック情報のブロック名である。次ポインタ（項目＃Ｎ）は、ステップＳ１２９で追加されるブロック情報のブロック名である。また、１つ前のブロック情報の次ポインタ（項目＃Ｎ）を更新する。

【０１８９】

（Ｓ１２９）変換部１４７は、ステップＳ１２８のブロック情報の次に、新たなブロッ

ク情報を追加する。ここで追加するブロック情報は、隙間判定において変数ｃの値と隙間の上限とを比較するブロックのブロック情報である。

【０１９０】

ブロック名（項目＃Ａ）は、既存のラベルと重複しない新たなラベルである。比較値（項目＃Ｂ）は、ステップＳ１２７で抽出した比較値より１つ大きい比較値である。隙間フラグ（項目＃Ｄ）は、ｔｒｕｅである。境界フラグ（項目＃Ｃ）、分岐点フラグ（項目＃Ｅ）および連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。真ブロック（項目＃Ｇ）は、末尾のブロック情報のブロック名である。偽ブロック（項目＃Ｈ）は、１つ後ろのブロック情報のブロック名である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。ｂｇフラグ（項目＃Ｊ）、ｂｇｅフラグ（項目＃Ｋ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。前ポインタ（項目＃Ｍ）は、ステップＳ１２８で追加したブロック情報のブロック名である。次ポインタ（項目＃Ｎ）は、１つ後ろのブロック情報のブロック名である。１つ後ろのブロック情報の前ポインタ（項目＃Ｍ）を更新する。また、区間数ｘが２以上である場合、２以上の区間についての隙間判定が連続的に実行されるように、真ブロック（項目＃Ｇ）などを適宜更新する。そして、ステップＳ１２６に処理が進む。

【０１９１】

図１９は、ブロック情報の第１の更新例を示す図である。

上記のステップＳ１１９において、例えば、ブロック情報１５２ｍが追加される。ブロック名（項目＃Ａ）は、新たなラベルであるＬ１４である。比較値（項目＃Ｂ）は、最小の比較値「９」である。境界フラグ（項目＃Ｃ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、Ｌ１２である。偽ブロック（項目＃Ｈ）は、ブロック情報１５２ｎを示すＬ１５である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、先頭に移動したＬ５である。次ポインタ（項目＃Ｎ）は、Ｌ１５である。

【０１９２】

また、上記のステップＳ１２０において、例えば、ブロック情報１５２ｎが追加される。ブロック名（項目＃Ａ）は、新たなラベルであるＬ１５である。比較値（項目＃Ｂ）は、最大の比較値「１２５」である。境界フラグ（項目＃Ｃ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、Ｌ１２である。偽ブロック（項目＃Ｈ）は、ブロック情報１５２ｏを示すＬ１６である。ｂｇフラグ（項目＃Ｊ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ１４である。次ポインタ（項目＃Ｎ）は、Ｌ１６である。なお、ブロック情報１５２ｍ，１５２ｎは、境界判定１６４ａに対応する。

【０１９３】

また、上記のステップＳ１２８において、例えば、ブロック情報１５２ｏが追加される。ブロック名（項目＃Ａ）は、新たなラベルであるＬ１６である。比較値（項目＃Ｂ）は、間隔マップ１５４の１番目の比較値に１を加えた「６０」である。隙間フラグ（項目＃Ｄ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、ブロック情報１５２ｐの後ろのブロック情報１５２ａを示すＬ１である。偽ブロック（項目＃Ｈ）は、ブロック情報１５２ｐを示すＬ１７である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ１５である。次ポインタ（項目＃Ｎ）は、Ｌ１７である。

【０１９４】

また、上記のステップＳ１２９において、例えば、ブロック情報１５２ｐが追加される。ブロック名（項目＃Ａ）は、新たなラベルであるＬ１７である。比較値（項目＃Ｂ）は、間隔マップ１５４の１番目の比較値より１つ大きい比較値である「１２３」である。隙間フラグ（項目＃Ｄ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、Ｌ１２である。偽ブロック（項目＃Ｈ）は、Ｌ１である。ｂｌフラグ（項目＃Ｉ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ１６である。次ポインタ（項目＃Ｎ）は、Ｌ１である。なお、ブロック情報１５２ｏ，１５２ｐは、隙間判定１６４ｂに対応する。

【０１９５】

図２０は、探索木変換の手順例を示すフローチャートである。

探索木変換は、前述のステップＳ２３で実行される。

（Ｓ１３０）変換部１４７は、変数ｑに１を代入する。

【０１９６】

（Ｓ１３１）変換部１４７は、変数ｑの値が二分木の高さｄ以下であるか判断する。高さｄは予め与えられていてもよいし、比較値の個数から決定してもよい。変数ｑの値が高さｄ以下である場合はステップＳ１３２に処理が進み、変数ｑの値が高さｄを超える場合はステップＳ１３８に処理が進む。

【０１９７】

（Ｓ１３２）変換部１４７は、変数ｒに１を代入する。

（Ｓ１３３）変換部１４７は、変数ｒの値が変数ｑの値以下であるか判断する。変数ｒの値が変数ｑの値以下である場合はステップＳ１３５に処理が進み、変数ｒの値が変数ｑの値を超える場合はステップＳ１３４に処理が進む。

【０１９８】

（Ｓ１３４）変換部１４７は、変数ｑの値に１を加える（変数ｑをインクリメントする）。そして、ステップＳ１３１に処理が進む。

（Ｓ１３５）変換部１４７は、インデックスｐ＝ｋ／2^{q}×（２ｒ－１）を算出する。ｋは、比較値の個数、すなわち、比較値ベクトル１５３のサイズである。上記の式でインデックスｐの値が非整数になった場合、最終的な計算結果の小数点以下を切り捨てる。例えば、ｋ＝１０，ｑ＝２，ｒ＝２の場合、ｐ＝１０／４×３＝３０／４＝７となる。

【０１９９】

（Ｓ１３６）変換部１４７は、比較値ベクトル１５３からインデックスｐの比較値とインデックスｐ－１の比較値を取得する。変換部１４７は、インデックスｐの比較値とｂｇｅ命令とを対応付けて命令テーブル１５５に追加する。また、変換部１４７は、インデックスｐ－１の比較値とｂａ命令とを対応付けて命令テーブル１５５に追加する。

【０２００】

（Ｓ１３７）変換部１４７は、変数ｒの値に１を加える（変数ｒをインクリメントする）。そして、ステップＳ１３３に処理が進む。

（Ｓ１３８）変換部１４７は、先頭のブロック情報をブロック情報ｂとして選択する。

【０２０１】

（Ｓ１３９）変換部１４７は、ブロック情報ｂの境界フラグ（項目＃Ｃ）がｆａｌｓｅ、隙間フラグ（項目＃Ｄ）がｆａｌｓｅ、かつ、連続フラグ（項目＃Ｆ）がｔｒｕｅであるか判断する。上記条件を満たす場合はステップＳ１４１に処理が進み、満たさない場合はステップＳ１４０に処理が進む。

【０２０２】

（Ｓ１４０）変換部１４７は、ブロック情報ｂの次ポインタ（項目＃Ｎ）が示すブロック情報を次のブロック情報ｂとして選択する。そして、ステップＳ１３９に処理が進む。

（Ｓ１４１）変換部１４７は、ブロック情報ｂおよびそれより後ろのブロック情報を、比較値の昇順にソートする。例えば、図７の例の場合、ラベルＬ１～Ｌ１１を、ラベルＬ３，Ｌ１，Ｌ４，Ｌ２，Ｌ６，Ｌ７～Ｌ１１の順にソートする。

【０２０３】

（Ｓ１４２）変換部１４７は、命令テーブル１５５に基づいてブロック情報の追加や更新を行う。具体的には、変換部１４７は、命令テーブル１５５から命令種類がｂｇｅである比較値を検索し、検索した比較値それぞれに対応するブロック情報を追加する。ここで追加するブロック情報は、分岐点判定を示すブロックのブロック情報である。

【０２０４】

ブロック名（項目＃Ａ）は、既存のラベルと重複しない新たなラベルである。比較値（項目＃Ｂ）は、命令テーブル１５５から検索した比較値である。分岐点フラグ（項目＃Ｅ）は、ｔｒｕｅである。境界フラグ（項目＃Ｃ）、隙間フラグ（項目＃Ｄ）および連続フラグ（項目＃Ｆ）は、ｆａｌｓｅである。ｂｇｅフラグ（項目＃Ｋ）は、ｔｒｕｅである。ｂｌフラグ（項目＃Ｉ）、ｂｇフラグ（項目＃Ｊ）およびｂａフラグ（項目＃Ｌ）は、ｆａｌｓｅである。このブロック情報は、二分探索が実現される位置に挿入される。真ブ

ロック（項目＃Ｇ）、偽ブロック（項目＃Ｈ）、前ポインタ（項目＃Ｍ）および次ポインタ（項目＃Ｎ）は、挿入された位置に応じた適切なラベルとなる。偽ブロック（項目＃Ｈ）のラベルと次ポインタ（項目＃Ｎ）のラベルは一致する。

【０２０５】

また、変換部１４７は、命令テーブル１５５から命令種類がｂａである比較値を検索し、検索した比較値を含むブロック情報を検索する。変換部１４７は、検索したブロック情報のｂａフラグ（項目＃Ｌ）をｔｒｕｅに変更する。また、変換部１４７は、検索したブロック情報の偽ブロック（項目＃Ｈ）を末尾のブロック情報のラベルに変更する。

【０２０６】

図２１は、ブロック情報の第２の更新例を示す図である。

上記のステップＳ１４２において、例えば、ブロック情報１５２ｑ，１５２ｒ，１５２ｓが追加される。ブロック情報１５２ｑは、命令テーブル１５５の比較値「３９」に対応し、分岐点判定１６４ｃに対応する。ブロック情報１５２ｒは、命令テーブル１５５の比較値「１３」に対応し、分岐点判定１６４ｄに対応する。ブロック情報１５２ｓは、命令テーブル１５５の比較値「５９」に対応し、分岐点判定１６４ｅに対応する。

【０２０７】

ブロック情報１５２ｑについて、ブロック名（項目＃Ａ）は、新たなラベルであるＬ１８である。比較値（項目＃Ｂ）は、命令テーブル１５５から検索された「３９」である。分岐点フラグ（項目＃Ｅ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、ブロック情報１５２ｓを示すＬ２０である。偽ブロック（項目＃Ｈ）は、ブロック情報１５２ｒを示すＬ１９である。ｂｇｅフラグ（項目＃Ｋ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ１７である。次ポインタ（項目＃Ｎ）は、Ｌ１９である。

【０２０８】

ブロック情報１５２ｒについて、ブロック名（項目＃Ａ）は、Ｌ１９である。比較値（項目＃Ｂ）は、命令テーブル１５５から検索された「１３」である。分岐点フラグ（項目＃Ｅ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、Ｌ４である。偽ブロック（項目＃Ｈ）は、Ｌ３である。ｂｇｅフラグ（項目＃Ｋ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ１８である。次ポインタ（項目＃Ｎ）は、Ｌ３である。

【０２０９】

ブロック情報１５２ｓについて、ブロック名（項目＃Ａ）は、Ｌ２０である。比較値（項目＃Ｂ）は、命令テーブル１５５から検索された「５９」である。分岐点フラグ（項目＃Ｅ）は、ｔｒｕｅである。真ブロック（項目＃Ｇ）は、Ｌ９である。偽ブロック（項目＃Ｈ）は、Ｌ７である。ｂｇｅフラグ（項目＃Ｋ）は、ｔｒｕｅである。前ポインタ（項目＃Ｍ）は、Ｌ６である。次ポインタ（項目＃Ｎ）は、Ｌ７である。

【０２１０】

また、上記のステップＳ１４２において、例えば、ブロック情報１５２ａ，１５２ｆ，１５２ｈが更新される。ブロック情報１５２ａ，１５２ｆ，１５２ｈの偽ブロック（項目＃Ｈ）がＬ１２に変更されている。また、ブロック情報１５２ａ，１５２ｆ，１５２ｈのｂａフラグ（項目＃Ｌ）がｔｒｕｅに変更されている。この他に、上記のステップＳ１４１のソートによって、各ブロック情報の偽ブロック（項目＃Ｈ）、前ポインタ（項目＃Ｍ）および次ポインタ（項目＃Ｎ）が適切に変更される。更新された一連のブロック情報を先頭から末尾に向かって辿ることで、例えば、図１１のコード１６５が生成される。

【０２１１】

第２の実施の形態のコンパイル装置１００によれば、中間コードから、ある変数の値と複数の比較値それぞれとを比較し、変数の値が何れかの比較値と一致するか否かによって処理が分岐する制御構造が検出される。そして、検出された制御構造が、個々の比較値についての個別判定の前に、境界判定、隙間判定および分岐点判定が挿入された別の制御構造に変換される。境界判定では、変数の値が最小の比較値より小さいかまたは最大の比較値より大きい場合に、隙間判定と分岐点判定と全ての個別判定がスキップされる。隙間判定では、変数の値が間隔の広い隣接する２つの比較値の間に属する場合に、分岐点判定と

全ての個別判定がスキップされる。分岐点判定では、二分探索アルゴリズムに準じた探索方法によって一部の比較値についての個別判定がスキップされる。

【０２１２】

これにより、変数の値１つ当たりの平均の比較回数が削減され、比較処理および分岐処理を効率化できる。例えば、比較値を真になる確率が高い順にソートするだけでは、変数の値が何れの比較値とも一致しない場合、その変数の値と全ての比較値とを比較することになってしまう。また、比較値の間で真になる確率の差が小さい（ばらつきが小さい）場合には、比較回数を削減することが難しい。これに対し、第２の実施の形態の方法では、変数の値が何れの比較値とも一致しない場合でも比較回数を削減することができる。また、コンパイル装置１００は、比較結果が真になる確率を収集しなくてもよく、オブジェクトコードをテスト実行するコストやコンパイルのコストを削減することができる。

【０２１３】

また、変数の値１つ当たりの平均の比較回数が削減されることで、分岐命令の実行回数が削減される。分岐命令の実行回数を削減することで、オブジェクトコードを実行するプロセッサのパイプライン処理において、分岐予測の失敗により生じる再実行コスト（ペナルティ）を削減できる。よって、オブジェクトコードの実行効率を向上させることができる。また、隙間判定を行う比較値間の区間の個数は、比較パターン数が最小になるように決定される。また、制御構造を変換した後の比較パターン数が変換前よりも少なくなると推定される場合のみ、制御構造の変換が行われる。これにより、実行時の比較回数が一層削減され、比較処理および分岐処理を一層効率化できる。

【０２１４】

なお、前述のように、第１の実施の形態の情報処理は、コンパイル装置１０にプログラムを実行させることで実現できる。第２の実施の形態の情報処理は、コンパイル装置１００にプログラムを実行させることで実現できる。

【０２１５】

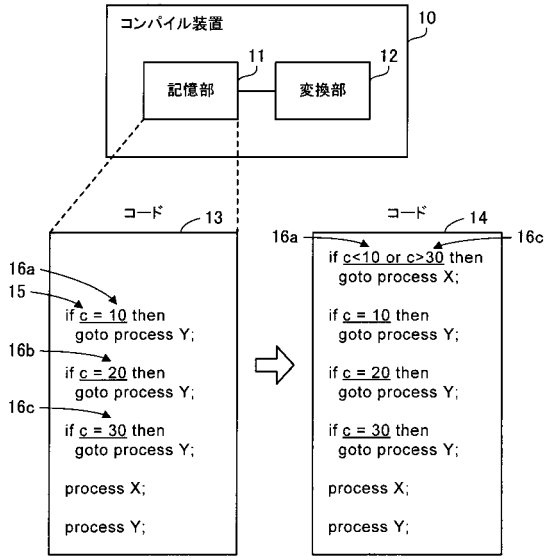
プログラムは、コンピュータ読み取り可能な記録媒体（例えば、記録媒体１１３）に記録しておくことができる。記録媒体として、例えば、磁気ディスク、光ディスク、光磁気ディスク、半導体メモリなどを使用できる。磁気ディスクには、ＦＤおよびＨＤＤが含まれる。光ディスクには、ＣＤ、ＣＤ－Ｒ（Recordable）／ＲＷ（Rewritable）、ＤＶＤおよびＤＶＤ－Ｒ／ＲＷが含まれる。プログラムは、可搬型の記録媒体に記録されて配布されることがある。その場合、可搬型の記録媒体から他の記録媒体（例えば、ＨＤＤ１０３）にプログラムをコピーして実行してもよい。

【符号の説明】

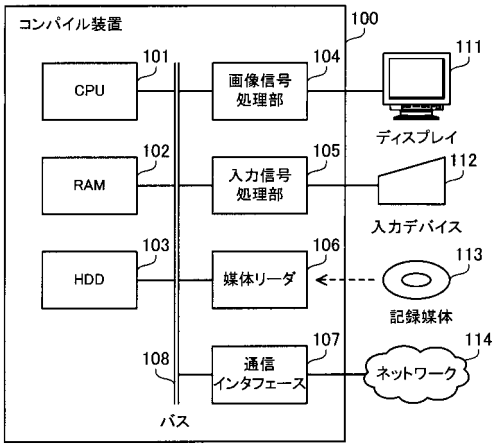
【０２１６】

１０　コンパイル装置
１１　記憶部
１２　変換部
１３，１４　コード
１５　変数
１６ａ，１６ｂ，１６ｃ　比較値

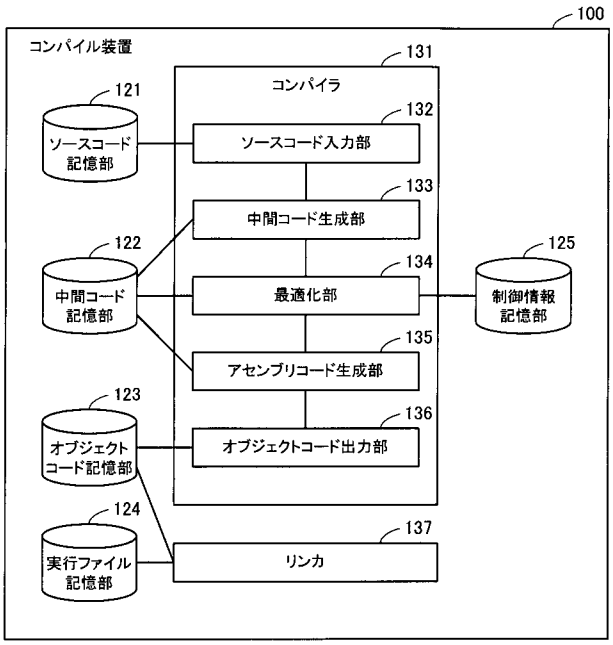
【図１】


コンパイル装置
10
記憶部
11
変換部
12
コード
13
コード
14
16a
16c
15
16b
if c = 10 then
goto process Y;
if c = 20 then
goto process Y;
if c = 30 then
goto process Y;
process X;
process Y;
if c<10 or c>30 then
goto process X;

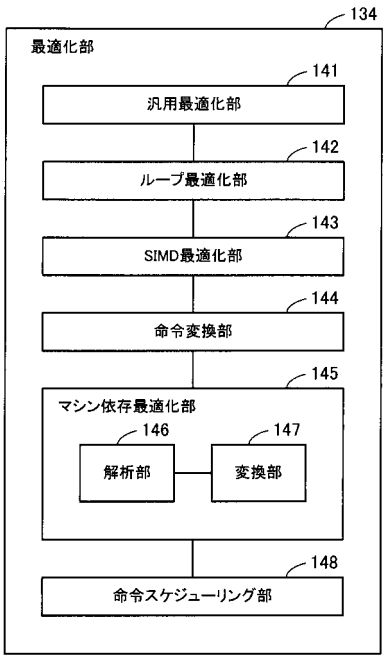
【図２】


コンパイル装置
100
CPU
101
RAM
102
HDD
103
画像信号処理部
104
入力信号処理部
105
媒体リーダ
106
通信インタフェース
107
バス
108
ディスプレイ
111
入力デバイス
112
記録媒体
113
ネットワーク
114

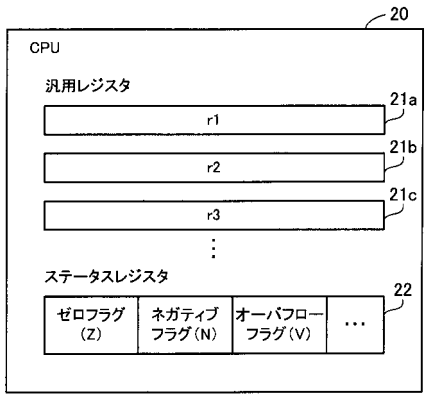
【図３】


100
コンパイル装置
131
コンパイラ
121
ソースコード記憶部
132
ソースコード入力部
133
中間コード生成部
122
中間コード記憶部
134
最適化部
125
制御情報記憶部
135
アセンブリコード生成部
123
オブジェクトコード記憶部
136
オブジェクトコード出力部
124
実行ファイル記憶部
137
リンカ

【図４】


134
最適化部
141
汎用最適化部
142
ループ最適化部
143
SIMD最適化部
144
命令変換部
145
マシン依存最適化部
146
解析部
147
変換部
148
命令スケジューリング部

【図５】

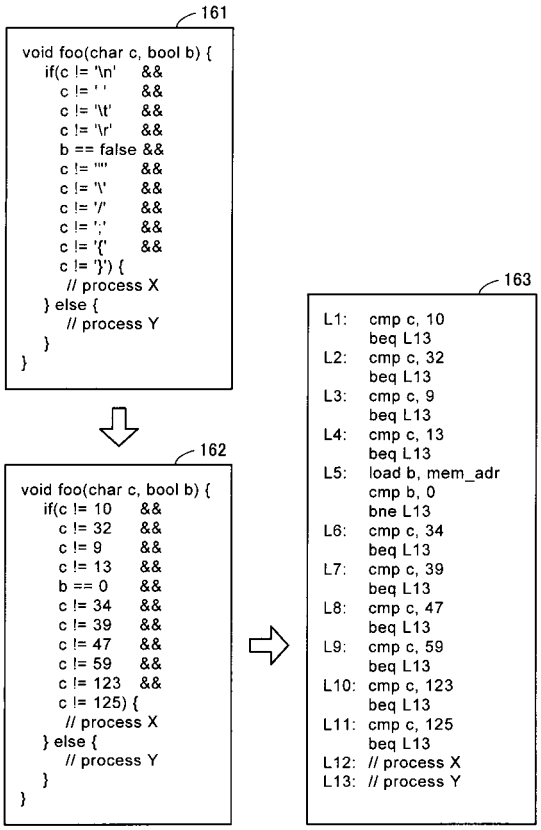


【図６】

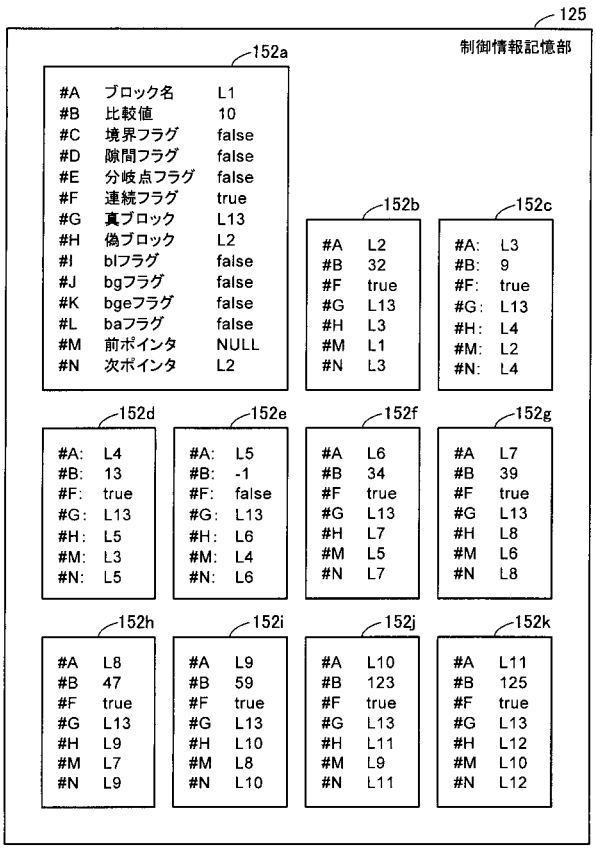
プロセッサ情報 151

命令	演算内容
cmp A, B	Z=1 if A=B N=1 if A<B V=1 if A-B<Min or A-B>Max
beq label	goto label if Z=1
bne label	goto label if Z=0
bl label	goto label if (N xor V)=1
bg label	goto label if (not (Z or (N xor V)))=1
bge label	goto label if (not (N xor V))=1
ba label	goto label

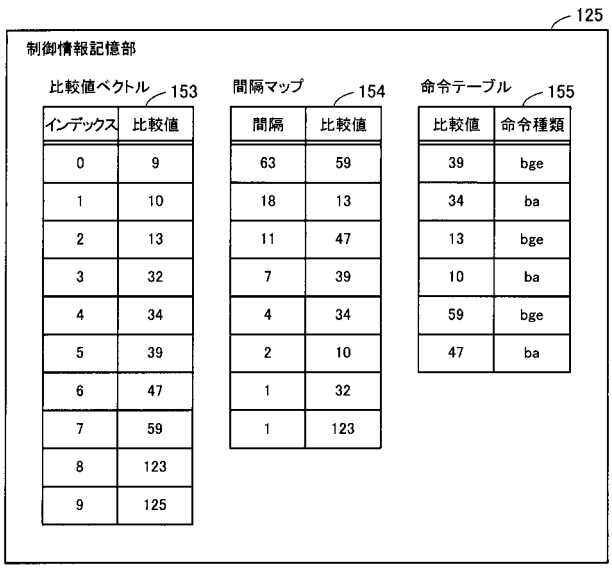
【図７】



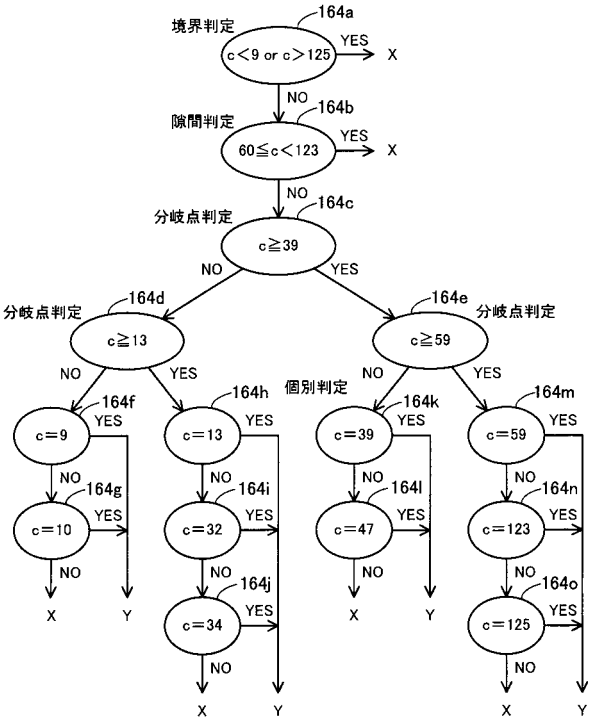
【図８】



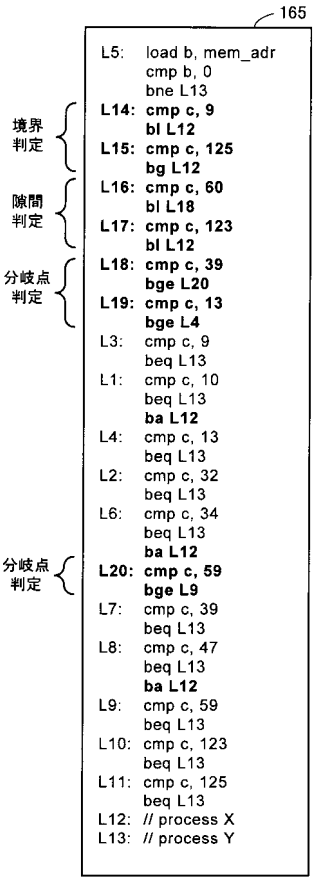
【図９】



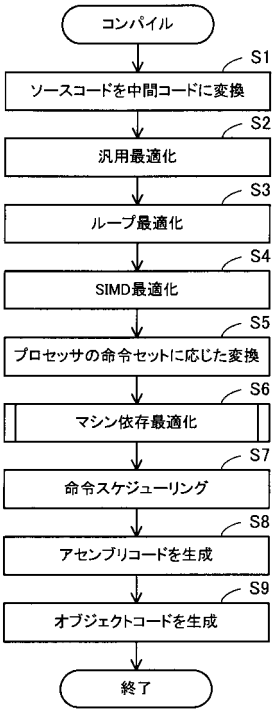
【図１０】



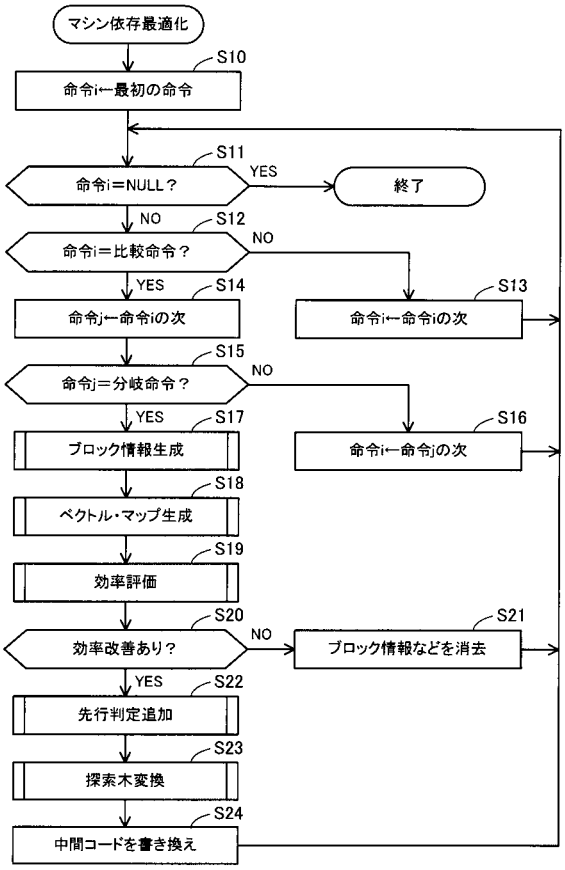
【図１１】



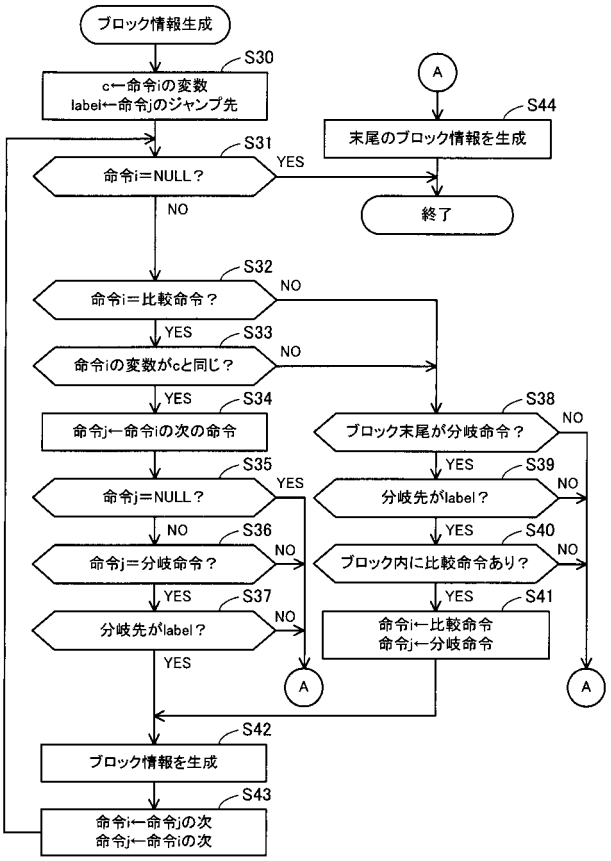
【図１２】



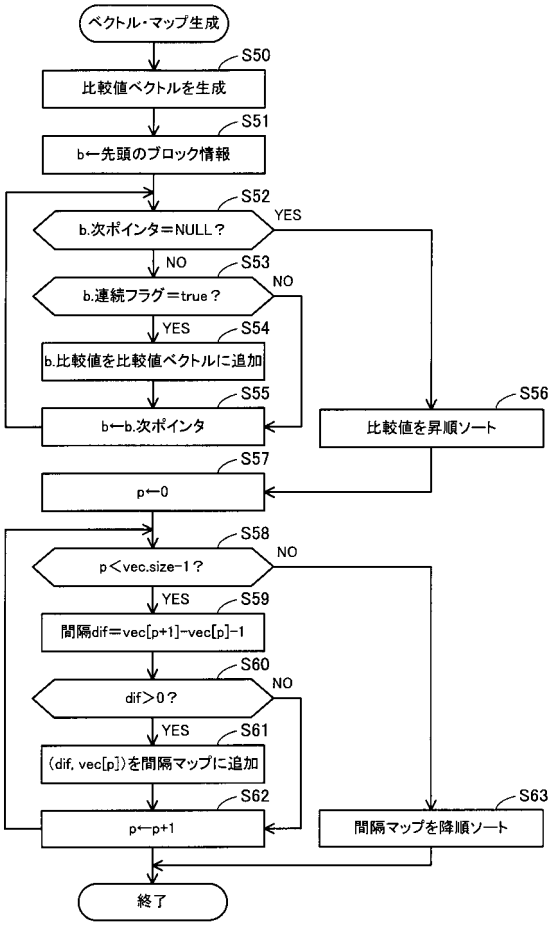
【図１３】



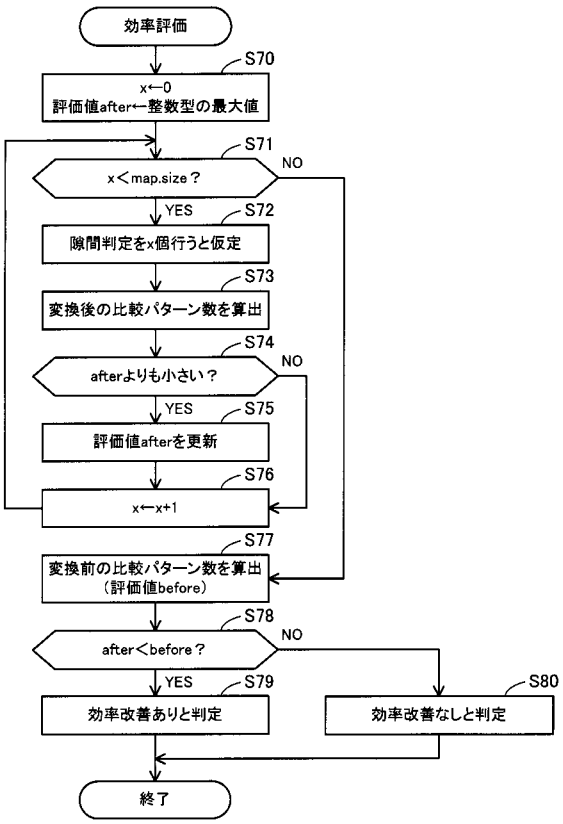
【図１４】



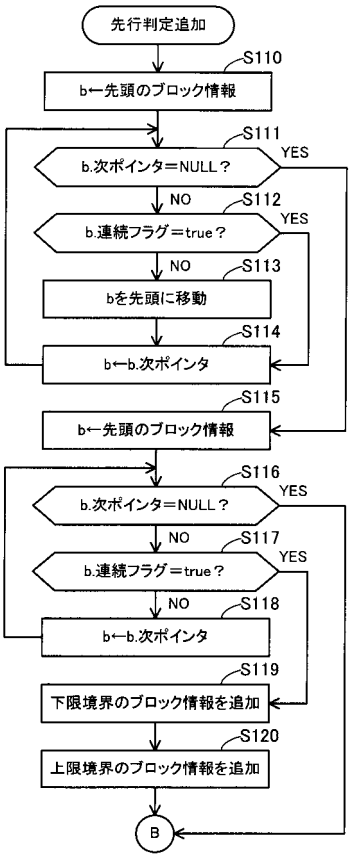
【図１５】



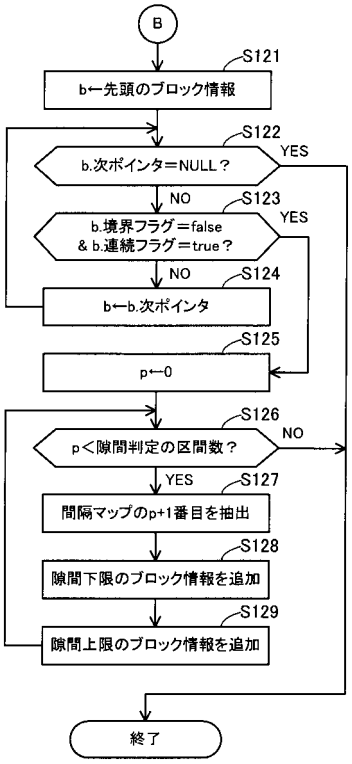
【図１６】



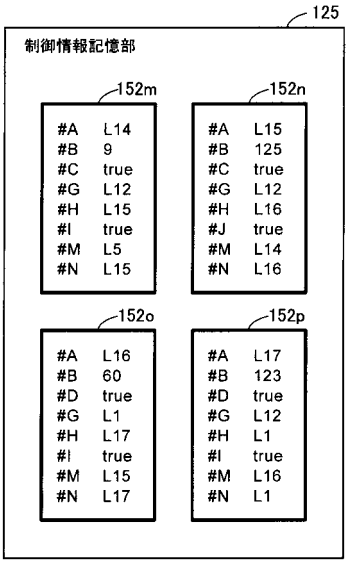
【図１７】



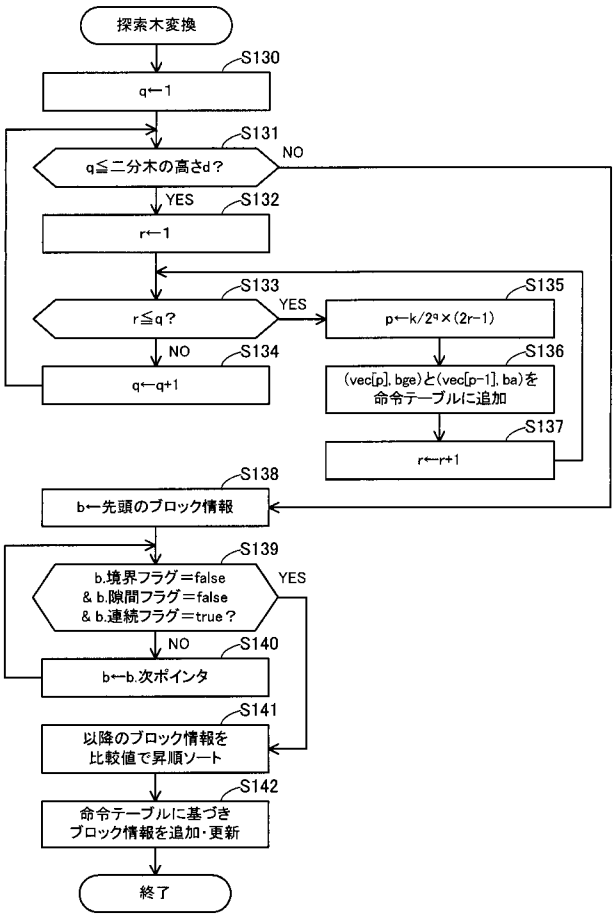
【図１８】



【図１９】



【図２０】



【図２１】