一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2018-121,DC2018-103 (2019-03)

PLCの高速化に関する研究(5)
—プリコンピューティングによる実行命令数の削減—

梶　　夢敏† 　堀口　雄輝† 　井口　幸洋†

† 明治大学理工学部情報科学科　〒214–8571 神奈川県川崎市多摩区東三田 1–1–1
E-mail: †{ce175012,ce175025,iguchi}@meiji.ac.jp

あらまし　プログラムコードを変更するだけで，PLC(Programmable Logic Controller) の実行を高速化する方法を提案する．PLC のプログラムにはラダー図が広く用いられている．高速化の方法はシンプルである：(1) ラダー図を，シーケンス命令に変換する．(2) シーケンス命令は，0 か 1 かのどちらかの値を持つ論理デバイスを数多く持っている．ある論理デバイス M_i が，論理値 0 (1) を持つと仮定すると，後続の M_i に関わる論理命令が省略できる．論理デバイス M_i に 0 を代入した場合と，1 を代入した場合の簡単化した二つの命令列を予め生成しておく．(3) それらを条件ジャンプで結合する．　本方法で，総命令数は増加するが，実行命令数は減少し，スキャンタイムは短くできる．予備実験により，実行命令数は, 実行命令数は 3.0 〜 4.7 % 程度に削減できることを示す．単純な方式では、総命令数を 6.3 倍に増加させるが、重複命令を省略することで、総命令を 2.9 倍程度に抑えられたことを示す．
キーワード　PLC(Programmable Logic Controller), プリコンピューティング

A Speed-up Method for PLCs(5)
—Reduction of Number Executed Instructions by Precomputing—

Yumeharu KAJI†, Yuki HORIGUCHI†, and Yukihiro IGUCHI†

† Department of Computer Science, Meiji University　Higashimita, Tama-ku, Kawasaki, Kanagawa, 214–8571 Japan
E-mail: †{ce175012,ce175025,iguchi}@meiji.ac.jp

Abstract　We propose a speed-up method for PLCs (Programmable Logic Controllers) by only modifying program codes. Ladder diagram (ladder logic) is widely used to PLCs. The idea of the speed-up method is simple: (1) We convert ladder diagrams to sequence instructions. (2) Sequence instructions have many logic devices which have values either 0 or 1. Assume that a logic device M_i has the value 0 (1), we can reduce some subsequent logical instructions which involves M_i. We pregenerate two reduced sequences; one is the codes in which M_i is assigned to 0, the other is the codes in which M_i is assigned to 1. (3) We connect them using a CJ (conditional jump) operation. Preliminary experimental results show that the number of executed instructions is reduced by 3.0〜 4.7 percent. Although the naive method increases the number of total instructions by 6.3 times. By eliminating duplicate instructions, we prevent the rate of increase by 2.9 times.
Key words　PLC(Programmable Logic Controller), Precomputation

1. はじめに

PLC (Programmable Logic Controller, プログラマブル・ロジック・コントローラ) は, シーケンサ, プログラマブル・コントローラとも呼ばれる. PLC は, 工場の生産設備, 発電所等の予め決められた処理を順次行うシーケンス制御に使われる [1] [6]. PLC のプログラム言語には, ラダー図やシーケンス命令などの 5 種類がある [6]. 生産現場では, 古典的なラダー図が好んで用いられている.

PLC を用いる FA(Factory Automation) の現場では, より高速な PLC が望まれている. より高速な PLC があれば, 1 つの PLC でより多数のセンサやスイッチなどの入力, アクアチュー

タ, ソレノイドなどの出力部品を制御できるからである.

PLC 高速化のためには, PLC の専用 MPU を開発する方法 [2] や, ラダー図から FPGA に直接マッピングする方法 [3] も提案されている. 我々の研究グループでは, PLC に関するフリーの環境を提供したいと考えている. 現在はシュミレータ, FPGA 内に高速 PLC 向きの MPU アーキテクチャを実現するハードウェア, 市販のマイクロプロセッサ基板を PLC として使うツールなどを作成中である

今回, PLC プログラムのデバイスへの代入することで, 2 種類の命令列を予め計算し生成することで実行命令数を削減する高速化手法を考案する. 基本的なアイディアは我々の研究グループが文献 [4] で示したが, [4] では PLC で実現したい機能をいったん C 言語に直し, 実現した. 今回は PLC での直接の実行を目指すため, シーケンス命令での変換を提案する. また, 実機でシェア No.1 の三菱電機製のシーケンサ (PLC の商品) のシーケンス命令を入力し, 本方法を処理するソフトウェアツールも開発した. 本方法を用いると PLC のハードウェアの改良は必要なく, プログラムの変更だけで高速化できる. 現存の PLC プログラムに適用できる高速化の条件についても述べる.

2. PLC の概要 [1] [6]

本章では, PLC の概要について述べる. 詳細は文献 [1] [6] を参照されたい.

シーケンス制御には, 電磁リレー, スイッチ等を模した部品とタイマやセンサなどの部品が用いられる. これらから構成されるラダー図 (Ladder Diagram) が, プログラミング言語として最もよく用いられる. 図 1 に使用する記号を, 図 2 (a) にラダー図の例を示す.

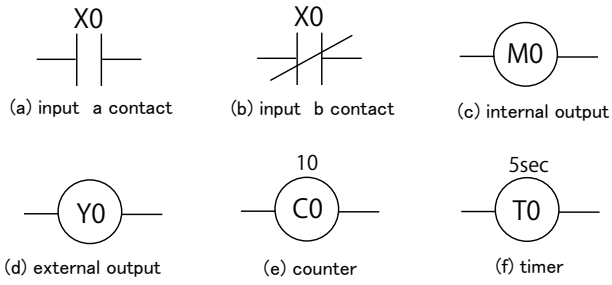

図 1 ラダー図に用いる記号

図 1 (a) の a 接点 (arbeit 接点) は, 入力 X0 が ON のとき, 両端が導通する. 図 1 (b) の b 接点 (break 接点) は入力 X0 が OFF のとき, 両端が導通する. 図 1 (c) は PLC の内部状態を記憶するレジスタへの出力である. M 以外にも B や F も内部レジスタを表す英字として使える. 図 1 (d) は外部出力, 図 1 (e) はカウンタ, 図 1 (f) はタイマを表す.

PLC 内部には, 入力値を保持する入力リレー, 内部状態を保持する内部リレー, 外部出力値を保持する外部出力リレー, カウンタ, タイマなどが存在する. ここでリレーと呼ぶのは, シーケンス制御が電磁リレー, スイッチ, ソレノイド, ランプ等で電気回路として構成されていた時からの慣例であり, 実体はレジスタやメモリで実現される記憶素子である. これらの値を保持するリレーをデバイスと呼ぶ. また, デバイスには大きく論理デバイス (ビット・デバイス) とワード・デバイスの 2 種類に分けられる. 論理デバイスは, 0 または 1 の二値を表現する. ワード・デバイスは, 16 ビットで表現可能な符号付き数 [-32768, 32767] と符号なし数 [0, 65535] を表現できる.

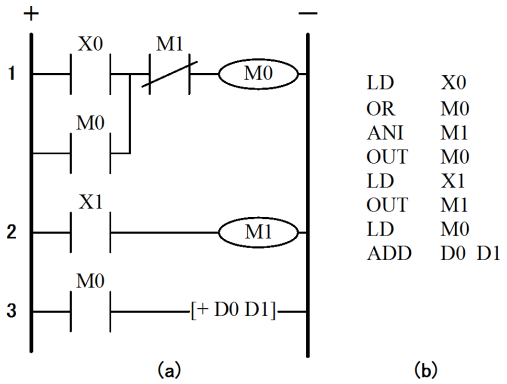

図 2 ラダー図と対応するシーケンス命令

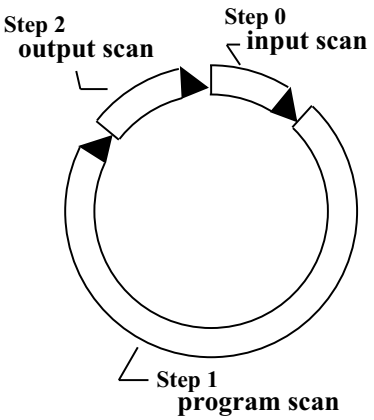

図 3 PLC の動作とスキャンタイム

PLC は, 図 3 に示す動作周期で以下の 3 ステップの動作を繰り返す.

Step0 制御システムに加えた入力値を読み取り, 入力デバイスに保持する.

Step1 ラダー図で記述された所定の処理を上から下に, 左から右に順に行い, ラダー図の最終行まで各行の右端にある各デバイスを更新する.

Step2 出力デバイスに保持された値を外部に出力し, Step 0 へ戻る.

これらの 3 ステップを 1 周期とし, これに要する時間を**スキャンタイム**と呼ぶ. これの短いものが望まれている. また, この時間が短いものを高速な PLC と呼ぶ.

PLC で制御を行うとき, ある特定の機種で特定のプログラムを動作させ, その動作タイミングを現場合わせで調整し, 以後そのタイミングを使うことがある. これは, 本来行ってはいけない. PLC としても保証していない. しかし, 現場のユーザが期待する場合もある. この場合は, スキャンタイムは常に一

定であるか, ある区間に関しての処理時間が一定であることが望まれる. この本来, 使用しては行けない特性をユーザーが利用するためにキャッシュメモリを搭載した MPU は利用できない(注1).

PLC 上での実行は, ラダー図からシーケンス命令に変換できる. 例えば, 図 2 (a) のラダー図は, 図 2 (b) のシーケンス命令に変換される. 1 行のラダーが複数のシーケンス命令に展開される. 1 個のシーケンス命令は, 最終的に複数の各 MPU 特有の機械語命令に変換される.

[例 2.1] カウントアップを実現するラダー図を図 2(a) に示す. 図 2 に, シーケンス命令に変換例を示す. ラダー図の 1 行目は, 一度 X0 が 1 になると M0 は 0 から 1 に変わり, M0 は 1 になり続けることを実現する. 再び M0 を 0 にしたい場合は, M1 を 1 にすればよい. この構造を自己保持回路と呼ぶ. 2 行目は, X1 を 1 にすることで M1 を 1 になる. これは 1 行目の自己保持回路のリセット信号を作る. 3 行目は, M0 が 1 であるときに D0 と D1 の値を足しあわせ, 結果を D1 に保存することを実現する. つまり, X0 を 1 にした後, X1 が 1 になるまで 1 回のスキャン毎に D1 をカウントアップし続ける動作を本プログラムは実現している.

3. プリコンピューティングによるスキャンタイムの高速化

ラダー図では, 論理演算が主体である. 論理演算の結果が 1 であれば処理を実行し, 論理演算の結果が 0 であれば処理を実行しない場合が多数現れる. 論理演算に使用する論理デバイスは 0 と 1 のみを扱う.

この特徴に注目し, ある論理デバイス M_i が, 論理値 0(1) を持つと仮定すると, 後続の M_i に関わる論理命令が省略できる. 論理デバイス M_i に 0 を代入した場合と, 1 を代入した場合の簡略化した二つの命令列を生成する. それらを分岐命令 (CJ: 条件ジャンプ) を使用し, 結合することで高速化されたソースコードを作成する.

我々の研究グループでは, このアイディアを文献 [4] で示したが, その文献内では PLC の動作を C 言語で構成する方法を用いた. ここでは, PLC で実際に使われるシーケンス命令を直接並べかえるより実機よりの方法とそのツールを開発した.

3.1 プリコンピューティングによる PLC の高速化手法 [4]

以下のいずれかの条件を満たすとき, ソースコードを分割し, 変数にあらかじめ値を代入する.

条件 1 値が確定した論理デバイスより下に, その論理デバイスが多数存在する.

これにより, スキャンタイムの高速化が図れる.

[例 3.1] 条件 1 が適用できるラダー図の例を図 4(a) に示す. 図 4(a) の 1 行目で内部論理デバイス M0 が確定する. 1 行目以降で M0 に関する値の変更がないので (PLC のラダーではリレーへの代入は 1 回限りと制限される. 2 個以上あるときはダブルコイルというエラーになる) , M0 に着目し, ソースコードを分割する. 分割すると図 4(b) に示す. 1 行目で M0=0 となる場合, 2 行目では X2 の値のみを参照し, X2=1 の時だけ M0V K2 D0 を実行し, 3 行目は実行不要であることがわかる. 同様に 1 行目で M0=1 になる場合は, 2 行目では X3 をロードした後, X2 と OR を取り, 結果が 1 の時だけ MOV K2 D0 を実行する. 3 行目では X2 をロードし, X2=1 のときだけ + D0 D1 を実行すればよい. この変更を CJ (条件付きジャンプ) を使い, 示したのが図 5 となる. 実行命令数は変化前が 12 である. 変化後は M0 = 0 の場合は 9, M0 = 1 の場合は 12 となる. この場合 M0=0 の時のみスキャンタイムが減少することが期待できる. 総命令数は 15 に増加する.

3.2 実験用ツール

高速化手法の条件 1 をシーケンス命令に適用し, 命令の削減を図るツールを作成した. 現状は, LDI, AND, ANI, ANB, OR, ORI, ORB の論理演算命令を削減できる.

作成したツールは, シーケンス命令が記述された実験データ (ソースコード) に対して, 以下の手順で操作を行う.

手順 1 実験データを木構造に変換.
手順 2 着目する論理デバイスを選択し, 実験データを分割.
手順 3 着目した論理デバイスに値を割当て.
手順 4 割当てを元に, シーケンス命令を削減.
手順 5 木構造から実験データを復元.

手順 1 で, 実験データを木構造に変換する. 文字列の実験データに対して追記や変更を行う場合, そのままで処理するより, 一時的に木構造へ変換したほうが取り扱いやすかったからである.

図 6 に示すシーケンス命令列を考える. 入力用論理デバイス X0 の値を内部論理デバイス M0 に保存. その後, 入力用論理デバイス X1 と M0 との AND の結果が 1 のみ [ADD D0 D1] を実行し, 0 の場合は実行しないことを示している.

図 6 を図 7 の木構造に変換する. 図 7 において, 円はデバイス, 四角形は論理演算命令, 三角形は論理演算以外の命令を示す. DO は左の子が先に実行される命令木であることを示す.

手順 2 で, 着目する論理デバイスをユーザー側が選択する. OUT 命令によって内部論理デバイスは 0 か 1 に確定されるため, 確定箇所以下のプログラムを複製することで実験データを分割する. 複製した木は分割を意味する CJ と書かれた Node の左右に繋げ, 木構造に挿入する. 今回の例での, 内部論理デバイス M0 に着目する場合, M0 の確定箇所は 2 行目にあたる. 図 7 においては DO の左側の OUT にあたる. よって, 確定箇所以下のプログラムは DO の右の子である. つまり, 図 7 は図 8 の形に変形する.

手順 3 で, 手順 2 にて着目した論理デバイスに値を割り当てる. 手順 2 で複製したプログラムの一方は, 着目した論理デバイスに 0 を代入し, もう一方には, 着目した論理デバイスに 1 を代入したものとする. 図 8 に対して, CJ の左の子のプログラム上の M0 に 0 を, 右の子のプログラム上の M0 に 1 を割り当てる. 割り当て後を図 9 に示す. なお, PLC のシーケンス命令で整数値の 1 は K1, 整数値の 0 は K0 と記述する.

(注1) : ルネサスエレクトロニクスで PLC メーカに MPU を供給する技術者 松嶋 潤氏との議論から得られた事項.

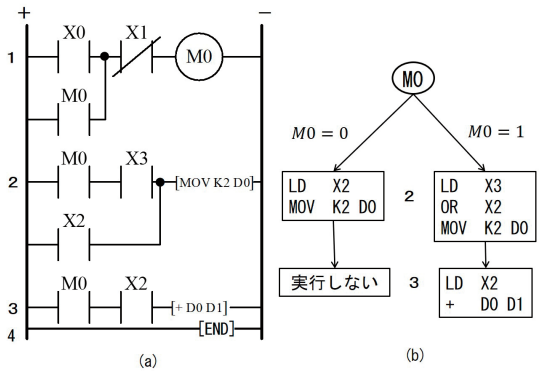

図 4　条件 1 に対応する例

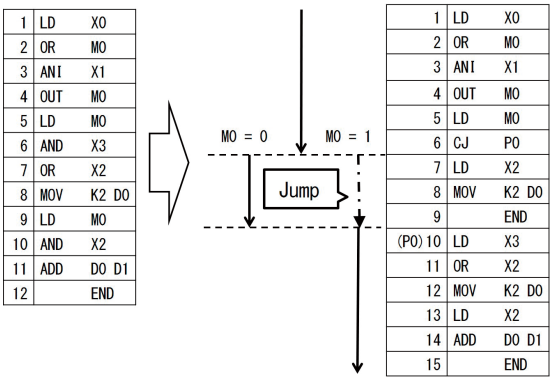

図 5　条件 1 における命令の変化

1	LD	X0
2	OUT	M0
3	LD	X1
4	AND	M0
5	ADD	D0 D1
6		END

図 6　論理演算後に ADD を行うソースコード

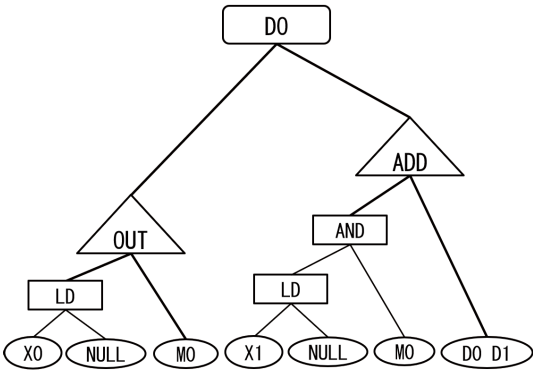

図 7　木構造に変換したソースコード (手順 1)

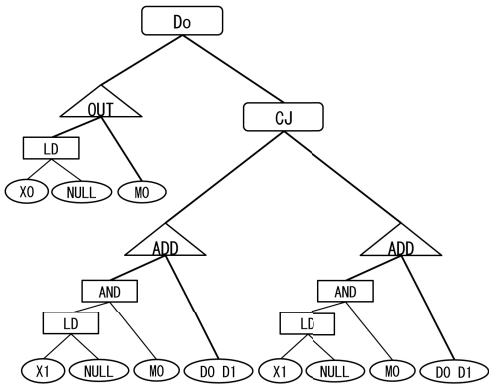

図 8　論理デバイスを元に分割を施したソースコード (手順 2)

手順 4 で, シーケンス命令の削減を行う. 整数値が関連している箇所において木構造の変形ができないかを検討する. 現在のツールでは, LDI, AND, ANI, ANB, OR, ORI, ORB について変形ができる. 図 9 の場合は, AND 命令について変形を検

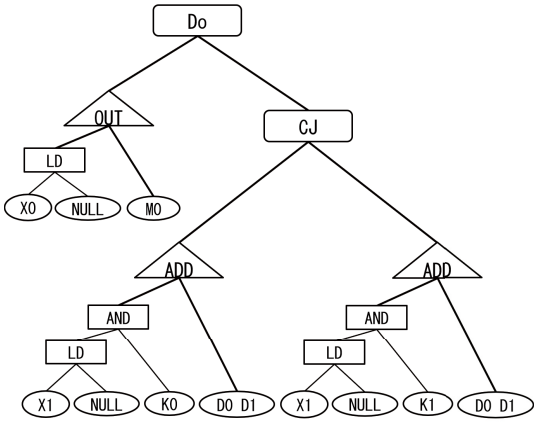

図 9　論理デバイスに値を割り当てたソースコード (手順 3)

討する. まず, CJ の左の子について考える. X1 と K0 の AND を取るので演算結果は K0. よって, ここでは AND 演算を省略し, K0 を LD するだけで良い. 次は CJ の右の子について考える. X1 と K1 の AND を取るので演算結果は X1 の値によって変わる. よって, ここでも AND 演算は省略でき, X1 を LD するだけで良い. つまり, 図 9 は図 10 に簡略化できる.

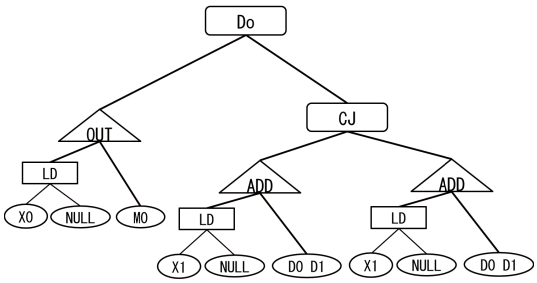

図 10　割当を元に削減を行ったソースコード (手順 4)

手順 5 で, 手順 4 で変形した木構造を元に, プログラムを復元する. 図 10 の場合は図 11 のソースコードを生成する.

4. 実 験 結 果

PLC プログラムに対して提案手法を適用する.

4.1 実 験 内 容

以下のような実験を行った.

1	LD	X0
2	OUT	M0
3	LD	M0
4	CJ	P0
5	LD	K0
6	ADD D0	D1
7		END
(P0) 8	LD	X1
9	ADD D0	D1
10		END

図 11　ツールによって生成されたソースコード (手順 5)

- 三菱電機提供のプログラムに対して手法を適用.
- エネルギー計測プログラム
  - 総命令数：1271
  - 内部論理デバイス B1, B15, B14 着目して分岐を適用.

実験に使用するエネルギー計測プログラムには, 条件 2 と条件 3 にあてはまる論理デバイスは存在しない. よって, 条件 1 を適用し, 実行命令数の削減を行う. また今回, 内部論理デバイス B1, B15, B14 に着目した理由は, 値の確定後の出現数が全デバイスの中で一番多かったためである. B1 は 30 回, B15 は 9 回, B14 は 8 回出現する.

4.2　実 験 結 果

実験の結果, 得られた実行命令数を表 1 に示す.

表 1　1 スキャンに実行する命令数の変化

分岐回数	内部論理デバイス	総命令数	平均実行命令数	最小実行命令数	最多実行命令数
0	-	1271	1271	1271	1271
1	B1	2268	1226	1210	1242
2	B15	4239	1221	1212	1236
3	B14	8010	1216	1208	1231

平均実行命令数は, それぞれの分岐率が等確率である場合の実行命令数である. 最小実行命令数は, 一番命令数が削減される分岐のみ通る場合の実行命令数である. 最多実行命令数は, 命令数の削減が一番少ない分岐のみ通る場合の実行命令数である.

表 1 より, 3.2〜5.0% 程度高速化できること, 総命令数は 6.3 倍になることが確認できる.

5.　総命令数増大の抑制

本章では, 実験において発生してしまった総命令数増大を抑制する手法と抑制を加味したツールの実装, 実験結果を述べる.

5.1　総命令数増大の抑制手法

現状のツールだと, 3.2 章の手順 2 より, 確定箇所以下のプログラム全てを複製するため, 総命令数は大きく増加する. これを抑制する方法について説明する.

図 12(a) のプログラムに注目する. A は, M0 確定箇所上方にあるコード. B は, M0 確定箇所下方にあり, M0 を含むコード. C は, M0 確定箇所下方にあり, M0 を含まないコードである. このプログラムに対して現状のツールは図 12(b) に示すように変形する. B_0 は, M0 に 0 を代入し, 命令を削減したコード. B_1 は, M0 に 1 を代入し, 命令を削減したコードである. つまり, 後続の C 全てを複製し, 総命令数が増大する.

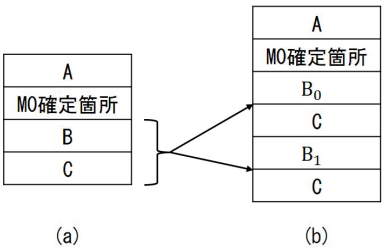

図 12　現状のツールにおけるプログラムの変換

これを, 展開する論理デバイスが出現する区間のみを複製すれば, 総命令数増大を抑制できる. 図 12(a) の場合は, 図 13(b) とする改良を行えばよい.

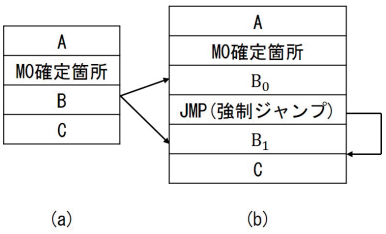

図 13　総命令数の増大を抑制したプログラムの変換

5.2　複製する区間を制限した実装

総命令数の抑制には, 展開する論理デバイスが出現する区間のみ複製すればよい. 今回作成したツールにおいて, 総命令数を抑制する手法を適応する. 3.2 章の手順 2 の時点で, 論理デバイスが出現する区間としない区間を判定し, 実験データを分割する. この際, 複製されたプログラムの片方に JMP 命令を挿入することで総命令数を抑制する.

図 12(a) を木構造で表したものを図 14 示す. 破線はプログラムが続いていくことを示し, A, B, C, は図 12 に則る.

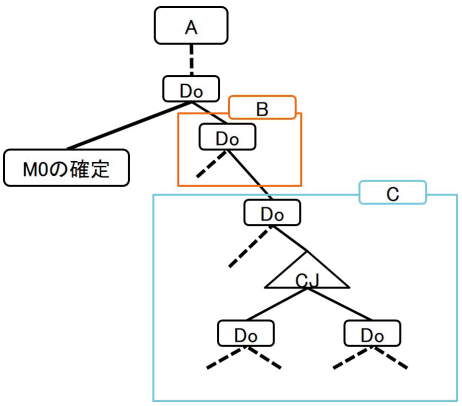

図 14　木構造に変換した図 12(a)

図 14 に対して, 総命令数の抑制を考慮する場合, 図 15 になる. また, 図 15 における A, B_0, B_1, C は図 12 に則る.

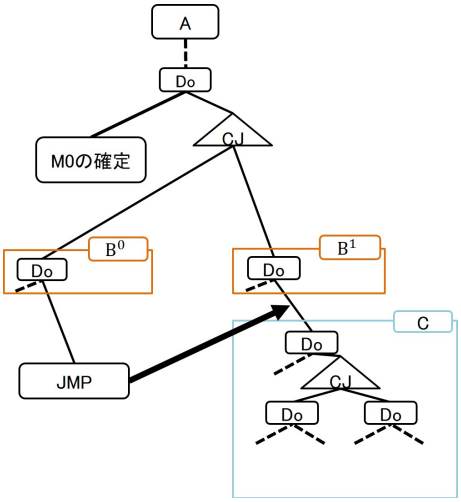

図 15　総命令数の抑制を考慮するプログラムの複製

5.3　総命令数の変化

総命令数を抑制する手法を適用し, 再度 4 章で行ったものと同じ内容の実験を行う. 実験結果を表 2 と表 3 に示す.

表 2　総命令数の変化

分岐回数	内部論理デバイス	複製箇所を制限しない総命令数	複製箇所を制限した総命令数
0	-	1271	1271
1	B1	2268	2109
2	B15	4239	3323
3	B14	8010	3715

表 3　複製箇所を制限した場合の命令数の変化

分岐回数	内部論理デバイス	平均実行命令数	最小実行命令数	最多実行命令数
0	-	1271	1271	1271
1	B1	1227	1211	1243
2	B15	1223	1214	1238
3	B14	1219	1211	1234

表 2 より, 総命令数は 6.3 倍から 2.9 倍になることが確認できる. また強制ジャンプを挿入するため, 実行命令数は増加する. 実行命令数の削減量は 3.2-5.0% 程度から表 3 より, 3.0-4.7% 程度に変化する.

5.4　考　　察

総命令数の抑制量は図 12(a) における C のような重複箇所の大きさに比例する. エネルギー計測プログラムにおける各デバイスの最終出現位置を考える. B1 は 1109 ステップ, B15 は 663 ステップ, B14 は 620 ステップに出現する. 通常時の実行命令数が 1271 ステップなので, B1 については重複箇所が 162 ステップと少ないため, 総命令数の抑制量は小さい. B15 については重複箇所が 608 ステップ, B14 について重複箇所が 651 ステップと多いため, 総命令数の抑制量は大きい.

6.　まとめと今後の課題

本稿では, 論理デバイスへの割当に基づく PLC プログラムの高速化を提案した. 実際に提案手法適用し, 有用性を確認した. 実験の結果より, 3.2〜5.0% 程度高速化できること, 総命令数は 6.3 倍になることを確認した.

また, スキャンタイムを一定にする場合, 一番実行命令数の多い分岐に合わせるので, 3.2% の高速化になる. なお, メモリに余裕がある場合は分岐を増やすことで, さらなる高速化が期待できる.

総命令数を抑制したい場合は, 抑制手法を適用した場合の実験結果より, 総命令数は 2.9 倍となった. なお, 実行命令数は 3.0-4.7% 程度に変化する.

今回は命令数のみに触れた. 本稿の手法では CJ を追加するので, 実機における CJ の実行時間が重要である. 三菱電気の PLC は LD などの基本論理命令の処理時間が 0.98ns, CJ の処理時間が 1000ns のため [8] 今回の手法は効果的ではない. しかし, H8 マイコンボードを PLC として利用する場合, LD の実行ステート数は 16, CJ の実行ステート数は 8 であり, 本手法が適用できる [5] [7].

今後の課題としては, より良い削減手法の提案, またニーモニック命令を機械語に変換するツールを作成する予定である.

文　　献

[1] W. Bolton, *Programmable Logic Controllers Third Edition,* Newnes, 2003.
[2] J. Kim, J. Park, W. H. Kwon, “Architecture of a ladder solving processor(LSP) for programmable controllers,” *Proc. IEEE IECON’91*, 1991.
[3] John T.Welth, Joan Carletta, “A Direct Mapping FPGA Architecture for Industrial Process Control Applications,” *Computer Design, 2000. Proceedings. 2000 International Conference on*, 2000.
[4] 浦野 雄太, 山田 陽平, 川口 亮二, 井口 幸洋, “MSP430 を用いた超低消費電力高速プログラマブル・ロジック・コントローラの提案,” 明治大学 理工学部 情報科学科 卒業論文, 2013 年 2 月.
[5] 川原 篤生, “ラダー図による H8 マイコン活用入門,” CQ 出版社, 2007 年.
[6] 関口 隆, 高橋 浩, 青木正夫, 下川勝千, 薦田憲久, “シーケンス制御工学,” 電気学会 , オーム社, 1988 年.
[7] 株式会社ルネサステクノロジ営業企画統括部 (2017), “H8/3694F グループ ハードウェアマニュアル,” http://akizukidenshi.com/download/ds/renesas/h83694f.pdf 2017 年 11 月 17 日アクセス.
[8] 三菱電機株式会社 (2017), “三菱電機汎用シーケンサ MELSEC iQ-R プログラミングマニュアル (命令/汎用 FUN/汎用 FB 編),” http://dl.mitsubishielectric.co.jp/dl/fa/members/document/manual/plc/sh081226/sh081226n.pdf 2017 年 11 月 17 日アクセス.