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
CPSY2018-120,DC2018-102 (2019-03)

PLCの高速化に関する研究(4)

ーPLC用MPUアーキテクチャと専用コンパイラについてー

堀口　雄揮† 　梶　　夢敏† 　井口　幸洋†

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

あらまし　高速な PLC(Programmable Logic Controller) を実現するための MPU アーキテクチャとそれらのためのコンパイラを提案する. 予め論理演算の計算結果を計算しておき, LUT に格納しておく. 論理演算を実行するかわりに表引きをすることで高速化する. LUT を 2 個入れた MPU では同時に 3 命令を 2 並列で実行できる. 予備実験の結果から 67% の実行ステップを削減できたことを示す.

キーワード　プログラマブル Programmable Logic Controller, LUT, FPGA, アーキテクチャ

A Speed-up Method for PLCs (4)

-MPU Architecture for PLCs and Its Compilers-

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

† Department of Computer Science, Meiji University　1–1–1 Higashimita, Tama-ku, Kanagawa, 214–8571 Japan
E-mail: †{ce175025,ce175012,iguchi}@meiji.ac.jp

Abstract　We propose a MPU architecture for PLCs (Programmable Logic Controllers) and its complier. The idea of the speed-up method is simple; (1) we precompute the results of three logic operations, and store them in LUTs, (2) we replace rumtime computation with retrieving a value from memory. The MPU with two LUTs including the results of all three logic operations is proposed. Preliminary experimental results show that the proposed MPU can reduce the number of steps by 67 percents.

Key words　Programmable Logic Controller, LUT

1. 序　　論

プログラマブル・ロジック・コントローラ (PLC) は, 工場の生産設備, 発電プラントや化学プラント, 遊園地の遊具, エレベータ等の制御に使われる [8]. PLC は, PC(Programmable Controller) やシーケンサ (Sequencer) とも呼ばれる. PLC は, プログラムを変更することで様々な作業工程に対応できる. PLC のプログラム言語には, ラダー図やシーケンス命令等の 5 種類がある [8]. 現在でも, 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 個の PLC で, より多く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, ランプなどの出力部品を制御でき, 経済的であり, 保守管理の面で優れているからである. また, 高速な制御が必要な場合もあり, 従来は専用のハードウェアを構成して制御していたものも高速な PLC なら置換できる. 高速化のために, PLC の専用 MPU を開発する方法 [3] [9] [10] や, ラダー図から直接マッピングする方法 [4] も提案されている.

我々の研究グループは, メモリを活用した PLC を FPGA 上に実装し, 商用の PLC に比べ平均 200 倍の高速化を実現した [11]. しかし, 配線領域の増加のために小中規模なプログラムのみにしか対応できなかった. 本稿では, PLC 専用のアーキテクチャを提案する. 記憶領域に演算結果を保存し, 表引きにより論理演算の削減を行う. これにより, 大規模なプログラムに対応しつつ高速化を図る方法を提案する.

2. PLCの概要 [1] [8]

本章では, PLC を概説する. 詳細は文献 [1] [8] を参照されたい.

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2019　by　IEICE

2.1 PLC の用途と歴史

PLC は, シーケンス制御を行う機器である. 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. これは, 以前はリレー, スイッチなどを使った非同期順序回路でシーケンス制御を行っていたなごりである. ラダー図の例を図 1 に示す.

例 2.1 ラダー図の例　図 1 の 1 行目のラダー図では, X0 の値を読み込み M0 の値と OR 演算を行う. その後, X1 の値を否定したものと AND を取り, その結果を M0 に格納する. 2 行目のラダー図では, M0 の値を読み取り 1 であれば [+ K1 D0] を実行し, 0 であれば実行をしない. つまり, M0 が 1 になると D0 の値が 1 増えるプログラムであり, PLC の動作は X1 の入力が 1 になるまで D0 をカウントアップする.

リレー回路は, 非同期順序回路である. 一方, PLC は入力言語の体裁は回路であっても動作は 1 文ずつ順次, プログラムを実行しており, その振る舞いはリレー回路とは異なる.

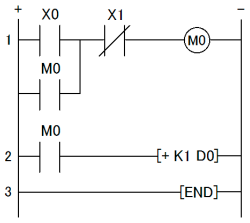

図 1　ラ ダ ー 図

```
LD    X0
OR    M0
ANI   X1
OUT   M0
LD    M0
ADD   K1   D0
END
```

図 2　図 1 に対応したシーケンス命令

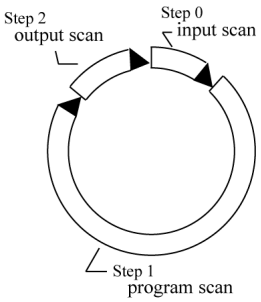

図 3　PLC の動作とスキャンタイム

PLC は, 図 2.1 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, 入力レジスタに保持する.

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レジスタを更新するか, 右端にある算術演算命令や代入命令などを実行する.

Step2　外部出力レジスタ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. これの短いものが高速な PLC である.

入力を一斉に取り込んで保持する. 次にプログラムを順次上から下へ左から右へ実行する. 最終行終了後に一斉に出力を行う. 本方式は, リフレッシュ方式と呼ばれ, 最も一般的な方法である. これ以外にも, スキャン中に入力値や外部出力の更新を随時行う方式 (ダイレクト方式), 入力は一括で取り込み, 出力は随時更新する混合方式がある. 本論文ではリフレッシュ方式を採用する. また, スキャンタイムは常に一定であることが望まれることも多い. 本来, PLC のスキャンタイムを一定にしないといけないわけではないが, 多くのユーザーがタイミングを合わせるのにこの性質を利用してしまうからである. このことより, PLC 内部に用いる MPU にキャッシュメモリによる高速化を施すことはできない(注1).

本稿では, PLC の高速化を目的の一つとしている. 従って, 我々が提案する PLC は, 外部から見た時の出力の変化の値だけではなく, 変化の順序も完全に通常の PLC と一致していることを条件にする. したがって, 命令の順序を入れかえることにより, 高速化を行う一般的な MPU に用いられる高速化技術も用いない.

2.2 ラダー図からシーケンス命令への変換

PLC 上での実行は, ラダー図からシーケンス命令に変換され, MPU で順次実行される. 例えば, 図 1 のラダー図は, 図 2 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. このシーケンス命令を直接, 解釈実行できる MPU でなければ, これらを更に複数の機械語命令に変換して MPU 上で実行する. シーケンス命令は, 論理演算を表す接点命令と算術演算や出力代入を行う基本命令の 2 つに分類できる. 本研究では, シーケンス命令中の 接点命令の高速化を行う.

例 2.2 接点命令と基本命令の例　図 2 の LD, OR, ANI 命令は接点命令に分類され, OUT, ADD 命令は基本命令に分類される.

2.3 論理演算の個数

図 2 では, 1〜3 行目で接点命令が LD,OR,ANI と 3 つ連続しているので, 連続する接点命令が 3 のものが 1 個, 同様に 6 行目では LD 命令が 1 つなので, 1 個とカウントする. 接点命令数が 2 のものはないので, 0 個とする. つまり, 図 2 のシーケンス命令は表 1 に示すように接点命令数が 1 個のものが 1 回, 2 個のものが 0 回, 3 個のものが 1 回存在したと数える. 以後の節において命令の個数はこのように数えるものとする.

表 1　接点命令とその表現

連続する接点命令の回数	個数
1	1
2	0
3	1

3. 接点命令の集約による高速化

3.1 接点命令の評価方法

PLC の実行形式の中心となる接点命令の評価方法に注目し, 高速化を図る. 従来の PLC では, 与えられた命令を逐次実行す

(注1)：ルネサスエレクトロニクスでシーケンサの MPU の供給に関わった松嶋潤氏のコメントによる

る. 従って, 少なくとも接点命令数のステップを必要とする.

例 3.1 接点命令と基本命令の例　図 2 の接点命令の実行の様子を図 3. 1 に示す. X0 をロードし, 次に M0 と OR をとる. そして X1 との AND Inverse をとることで接点命令の実行を終える. 最後に基本命令として, その結果を M0 に書き出す. 以上の接点命令 3 ステップ, 基本命令 1 ステップ, 合計 4 ステップで実行される.

接点命令は論理式で表現できる. これを予め計算し真理値表の形に記憶しておけば表引きにより接点命令の実行を置換できる. 表引きを行うアドレスとして, デバイスの値と本来実行する命令のオペコードを集約したもの (集約した命令) を使用する. この方式を用いた PLC 専用 MPU(マイクロプロセッサ) を本稿では提案する. プログラムの実行までの過程を以下に示す.

- コンパイラですべての接点命令の組み合わせに対する出力を予め計算し, 表に格納する. これを MPU 内の LUT にダウンロードする.
- 実行時は, 接点命令はデバイスの値と集約した命令とをもとに表引き (メモリ参照) を行い, 結果を得る.

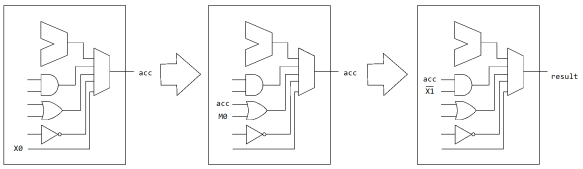

図 4　従来の接点命令の評価方法

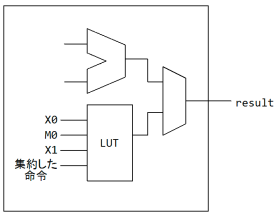

図 5　提案する接点命令の評価方法

このとき, 図 3. 1 に示すように, 3 回のステップを必要とする命令が, 図 3.2 では, 1 ステップで結果が得られる.

3. 2　PLC の命令と表現

LD	OR	ANI	X0	M0	X1

図 6　集約後の命令列

0	011	010	1	0	0

図 7　集約後の命令列 (実行時)

複数の命令を集約する工程を実現するための構造について説明する. 集約する命令を表 2 に示す. 集約対象の命令は NOP(No Operation) を含め 7 種類とする. LD, LDI 命令のどちらかが集約する命令の先頭に現れるので, 1 ビットで表す. その他の 5 種類の命令は後続命令として 3 ビットで表現する. 命令の集

表 2　接点命令とその表現

命令	2 進数表現
LD(Load)	0
LDI(Load Inverse)	1
NOP(No Operation)	000
AND	001
ANI(AND Inverse	010
OR	011
ORI(OR Inverse)	100

約について整理する．オペコードとオペランドに集約し，オペコード，オペランドの順に並べる形とする．

例 3.2　**図 2 の命令を 3 個ずつ集約し，実行時に値を代入した例** 3 命令を集約した例を図 6 に示す．1 行目の LD X0, 2 行目の OR M0, 3 行目の ANI X1 からオペコードの LD, OR, ANI とオペランドの X0, M0, X1 を抜き出し，オペコード，オペランドの順に並べ，命令メモリに格納する．命令の実行時には，オペランドの X0, M0, X1 にメモリから読みだした結果を代入し，メモリ参照を行う．メモリ参照を行う際に X0 が 1，M0 が 0，X1 が 0 だった例を図 7 に示す．

X0, X1 にはインプットスキャンで確定した 0 か (1) が入り，M0 にはプログラムの実行中に確定した 0 か (1) が入る．

3.3　メモリ参照と結果の格納

表 3　実行状態のメモリの様子

X0	M0	X1	$(X0 \vee M0) \cdot \overline{X1}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

実行過程を説明する．プログラムの実行前に，命令をコンパイルし出現した接点命令を決められた数に収まるように集約する．集約した命令と変数が取りうる値 0 か (1) をもとに，演算結果をメモリに保存する．コンパイル後のメモリの状態を表 3 に示す．例としてプログラムの実行中に図 6 の集約命令に X0 に 1, M0 に 0, X1 に 0 の値が入った場合，図 7 のようにアドレスが構成され，保存してある結果の 1 が出力される．

4.　PLC 専用 MPU のアーキテクチャ

4.1　基 本 構 造

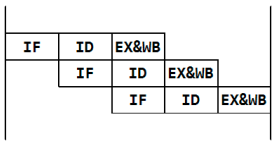

図 8　パイプラインの構成

3 種類の MPU を Verilog HDL で記述しシミュレーション環境を整えた．まず，3 つ MPU に共通するアーキテクチャについて説明する．3 つの MPU は図 8 に示すように 3 ステージのパイプライン構造とした．

- 命令を取得し，ProgramCount(pc) を 1 すすめる IF(Instruction Fetch) ステージ
- 命令を解析しメモリから計算に使用するデータを読みだし，レジスタに格納する ID(Instruction Decode) ステージ
- 与えられたデータと命令をもとに演算を行い，結果をメモリに書き込む EX&WB(Execute & Write Back) ステージ

表 4　命　令　表

命令	意味
NOP	No Operation
LD rs	exc <- rs
LDI rs	exc <- ¬rs
AND rs	exc <- exc ∧ rs
ANI rs	exc <- exc ∧ ¬rs
OR rs	exc <- exc ∨ rs
ORI rs	exc <- exc ∨ ¬rs
OUT rd	rd <- exc
MV rs rd	rd <- rs
ADD rs rd	rd <- rd + rs
ADD rs1 rs2 rd	rd <- rs1 + rs2
SUB rs rd	rd <- rd - rs
SUB rs1 rs2 rd	rd <- rs1 - rs2
MUL rs rd	rd <- rd × rs
MUL rs1 rs2 rd	rd <- rs1 × rs2
DIV rs rd	rd <- rd ÷ rs
DIV rs1 rs2 rd	rd <- rs1 ÷ rs2

PLC で基本的な命令を実行するのに必要な機械語命令を実装した．その内容を表 4 に示す．

4.2　従来方式の MPU

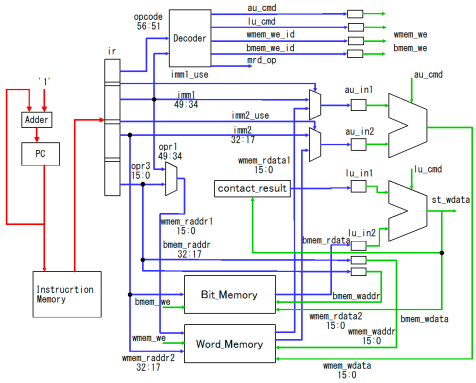

図 9　MPU_1 の概要

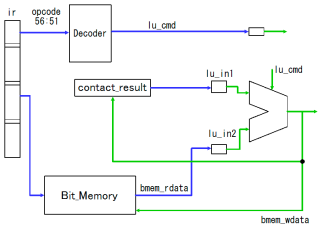

図 10　MPU_1 の論理演算

従来通りの 1 命令ずつ命令を実行する MPU を MPU_1 と呼ぶ．これは標準的な PLC 向きの MPU であり，シーケンス命令を直接実行できる．図 9 に MPU_1 の概要を示す．赤色の線が IF ステージを表し，命令を取得し，ProgramCount(pc) を 1 すすめる．青色の線が ID ステージを表し，命令の解析，メモリからデバイスの値の取得を行う．緑色の線が EX&WB ステージ

を表し, ID ステージで解析した命令を取得した値に対して実行し, write enable 信号が High になっていればメモリへ書き込みを行う. MPU_1 において接点命令を処理する ID, EX&WB ステージを図 10 に示す. 接点命令の処理は, 以下の工程で処理する.

- デコーダーによる論理演算の内容を判断する.
- メモリアクセスを行い, 演算に使用する値を取得する. 直前の処理で書き換えられていた場合書き換えた値を使用する.
- ID ステージで取得した値と演算内容をもとに計算する.

4.3 提案する MPU のアーキテクチャ

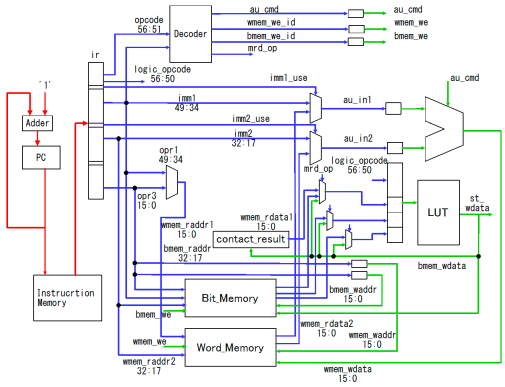

図 11　MPU_2 の概要

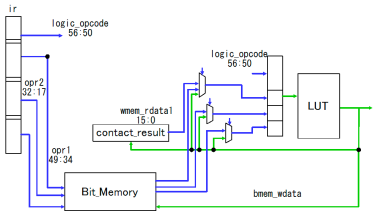

図 12　MPU_2 のデーブルルックアップ

接点命令を 3 個集約し, 命令を実行する MPU を MPU_2 と呼ぶ. MPU_2 のブロック図を図 11 に示す. 命令を取得する IF ステージは MPU_1 と共通だが, ID, EX&WB ステージに違いがあるので, その部分について説明する. 1 つの命令で最大 3 つのメモリにアクセスするためメモリのポート数が増加している. また, 複数回 LUT を参照する場合や書き換えるデバイスの値をロードした場合は直前の結果を使用する必要があるのでマルチプレクサによりメモリアクセスに使用するデバイスの値を決定する. 集約した命令と取得した値をもとに LUT の参照に使用するアドレスを決定する.

MPU_2 において接点命令を処理する ID, EX&WB ステージを図 12 に示す. 接点命令の処理は, 以下の工程で処理する.

- IF ステージで取得した命令から集約した命令を取得する.
- メモリアクセスを行い, 演算に使用する値を取得する. 直前の処理で書き換えられていた場合書き換えた値を使用する.
- 集約した命令と値をもとに LUT を参照し, 演算結果を得る.

MPU_2 を改良し, LUT を 2 個使用する MPU を MPU_3 と呼ぶ. 主な特徴としては, LUT を 2 個使用することで複数行の命令を並列に実行する. 同時に最大 6 個の値をメモリアクセスにより取得する必要があるので, 2 ポートメモリを 3 個持つ. MPU_3 のブロック図を図 11 に示す.

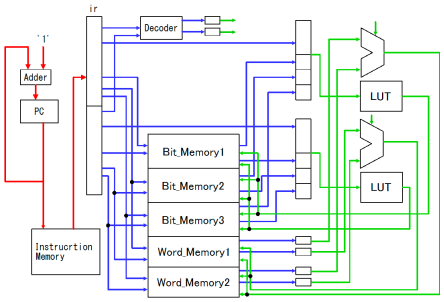

図 13　MPU_3 の概要

5. 専用コンパイラ

MPU_1, MPU_2, MPU_3 は受理する命令が異なる. そこで, それぞれのコンパイラを作成した.

5.1 命令表の作成

MPU_1 は従来通りの実行をするため特別な処理を必要としない. MPU_2 の命令表では接点命令の集約を行う. 3 個単位で集約するため, 連続する接点命令の数が 3 を超える場合は分割し, 不足する場合は NOP を追加する. MPU_3 の命令表は先に現れた命令を優先する (as soon as possible). また, 命令に依存関係がない命令は並列に実行する.

1	LD　X0
2	OR　M0
3	ANI X1
4	OUT M0
5	LD　M0
6	OUT Y0
7	LD　X2
8	OR　X3
9	AND X4
10	ANI X5
11	OUT Y1

図 14　MPU_1 が受理可能な命令

1	LD OR　ANI X0 M0 X1
2	OUT M0
3	LD NOP NOP M0 N0 N0
4	OUT Y0
5	LD OR　AND X2 X3 X4
6	LD ANI NOP N0 X5 N0
7	OUT Y1

図 15　MPU_2 が受理可能な命令

具体的な命令を例に MPU_1, MPU_2, MPU_3 のそれぞれで受理する命令表について説明する. 例とする命令表を図 14 に示す. MPU_1 では, 従来通りの実行を行うため図 14 の通り 11 ステップを必要とする. MPU_2, MPU_3 が受理する命令表は図 14 の命令をコンパイルしたものである. MPU_2 が受理する命令列を図 15 に示す. LUT を 1 個使用し 3 命令を集約するため, 1, 2, 3 番目の命令と 7, 8, 9, 10 番目を 1 命令に集約する. 5 番目の命令も LUT の参照を行うが 3 命令を集約ので, NOP を挿入することで 3 命令として LUT の参照を行う. 7〜10 番目の命令を集約すると 2 回 LUT を参照する必要があるので, 2 度目の参照を行う際には, 直前の参照結果をロードし, 残りの演算と集約する. 今回は命令の集約を行うことで, 4 命令分の実行ステッ

プを削減できたので, 7 ステップで実行できる.

MPU_3 が受理する命令列を図 16 に示す. この MPU は並列に処理できる行を並列に処理するので, それぞれの行の依存関係を調べる. このプログラムは 1〜4 番目, 5〜6 番目, 7〜11 番目, の命令で各行を構成しているので, この 3 行の依存関係を調べれば良い. 今回は, 1 行目 (1〜4 番目) で M0 の値を書き換え, 2 行目 (5〜6 番目) で M0 の値を使用するため 1 行目と 2 行目に依存関係がある. そこで 1 行目と依存関係のない 3 行目を並列に実行する. 1 行目の命令は 2 ステップで, 3 行目の命令は 3 ステップで実行できるため 1 ステップ分の空白が生まれる. このとき 3 行目と依存関係のない行があれば挿入する. この例では 2 行目が依存関係にないので 2 行目の命令を挿入する. 3 行目の挿入が終了すると挿入する行がないので, 全ての命令の挿入が終わるまで NOP を挿入する. これにより 4 ステップで実行できる.

1	LD OR ANI X0 M0 X1	LD OR AND X2 X3 X4
2	OUT M0	LD ANI NOP N0 X5 N0
3	LD NOP NOP M0 N0 N0	OUT Y1
4	OUT Y0	NOP

図 16 MPU_3 が受理可能な命令

6. 性能評価のための予備実験

MPU_1, MPU_2, 及び, MPU_3 の性能比較を行った. 従来の実行方法, LUT を 1 個使用した場合, LUT を 2 個使用した場合の実行ステップ数の削減量を比較する. どの程度, 実行ステップ数が削減されるかを 3 つのサンプルプログラム (三菱電機提供) を用いた予備実験を行った. サンプルプログラムの概要を表 5 に示す. サンプル 1 は計測管理のプログラムである. 表 5 のようにサンプル 1 ではラダーの行数は 694 行で, 総命令数は 1425 個, そのうち接点命令より 773 個であり, 本提案の方式では, この 773 個の命令部の実行時間を短縮できる. サンプル 2 は光電センサ制御を行うプログラムであり, サンプル 3 は変位センサ制御を行うプログラムである.

表 5 サンプルプログラムの詳細

	サンプル 1	サンプル 2	サンプル 3
ラダーの行数	694	306	211
総命令数	1425	834	454
接点命令の個数	773	561	275

論理演算の回数を n, 集約する命令数を m とすると LUT の参照回数は $n \leq m$ のとき , n 回となり, $n > m$ のとき, $\lceil \frac{n-m}{m-1} \rceil + 1$ 回となる. サンプルプログラム 1 に対して提案手法を適用した場合について説明する. メモリの参照回数は表 6 に示すように減少する.

表 6 の参照回数をもとに, 表 7 の命令実行数が計算できる. この結果より, 773 回の実行を必要としていた命令が 3 命令をまとめると 394 回での実行で計算できる. また, サンプルプログラムの総命令数が 1425, 接点命令が 619 なので, 接点命令が占める割合は 773/1425 =54% となる. LUT を参照することで, 接点命令が 394 回になるので, 接点命令は 394/773 =51% に減少した. LUT を 1 個使用する場合の MPU_2 では約 27% の実行ステップを削減できた.

また, LUT を 2 個使用し並列に実行する MPU_3 では総ステップ数は 530 となった. 530/1425 = 約 37% なので, MPU_3 では, 約 63% の実行ステップを削減できた.

サンプル 2, 3 も同様に適用したところサンプル 2 では約 67%, サンプル 3 では約 58% 実行ステップを削減できた. サンプル 1, 2, 3 の実行ステップ数の減少を表 8 に示す.

表 6 メモリの参照回数

論理演算の回数	メモリ参照回数
1	1
2	1
3	1
4	2
5	2
6	3
7	3
8	4
9	4
10	5
11	5
27	13

表 7 サンプル 1 における実行ステップ数の減少

論理演算の命令数	出現回数	従来	提案手法
1	78	78	78
2	74	148	74
3	30	90	30
4	34	136	68
5	22	110	44
6	9	54	27
7	7	49	21
8	5	40	20
9	1	9	4
10	1	10	5
11	2	22	10
27	1	27	13
実行ステップ数の合計	-	773	394

表 8 全サンプルにおける実行ステップ数の減少

	実行ステップ数		
	従来	LUT1 個使用	LUT2 個使用
サンプル 1	1425	1046	530
サンプル 2	834	542	272
サンプル 3	454	317	192

7. まとめと今後の課題

本稿では, PLC 向きの MPU を提案した.

- 接点命令を同時に 3 個以上実行することを表引きにより高速化を行った.
- パイプラインプロセッサを実装し, 性能比較を行った.
- LUT を 1 個用いた場合と LUT を 2 個用いて並列に実行する場合の実行ステップ数を比較した.
- サンプルプログラムにおいて, 実行ステップ数を最大 67% 削減できた.
- これらに対応する専用コンパイラを作成した.

今後の課題として, より多くのサンプルプログラムに対して効果を調べ, 提案したアーキテクチャの改良することが今後の課題である. また, 今回は MPU を Verilog HDL で記述し, シミュレーション環境で動作を確認した. しかし, FPGA 向けのコンパイルを行ったのみなので, FPGA 基板での動作は確認していない. FPGA 上での動作を確認をし, 専用プロセッサとして大量に生産した場合の動作速度の見積もりも行う必要がある.

文　　献

[1] W. Bolton, *Programmable Logic Controllers Third Edition,* Newnes, 2003.

[2] Ryoji Kawaguchi, Yohei Yamada, Kenji Takahashi, Yuta Urano, Yukihiro Iguchi, “A Realization Method of Fast and Dependable Programmable Logic Controllers”, WRTLT 2012, 2012 年.

[3] J. Kim, J. Park, W. H. Kwon, “Architecture of a ladder solving processor(LSP) for programmable controllers,” *Proc. IEEE IECON’91*, 1991.

[4] John T.Welth, Joan Carletta, “A Direct Mapping FPGA Architecture for Industrial Process Control Applications,” *Computer Design, 2000. Proceedings. 2000 International Conference on*, 2000.

[5] 天野英晴, 西村克信, 作りながら学ぶコンピュータアーキテクチャ, 培風館, 2001

[6] 浦野雄太, 山田洋平, 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”, 2012 年.

[7] 熊谷英樹, 必携シーケンス制御プログラム定石集機構図付き, 日刊工業新聞社, 2003

[8] 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

[9] 山口大介, 松永祐介, ” プログラマブルコントローラ向けプロセッサ · アーキテクチャの検討と評価”, 信学会技術研究報告, CPSY2002-108, pp.19-24, March 2003.

[10] 山口大介, 勝木裕二, 松永祐介, ” プログラマブルコントローラ向けプロセッサ · アーキテクチャの評価”, 情報処理学会研究報告, 2004-ARC-157, pp.91-96, March 2004.

[11] 山田陽平, 高橋賢治, 石田篤志, 井口幸洋, “プログラマブル・ロジック・コントローラの NPN 同値類を利用した高速化”, 多値論理フォーラム, 2011 年 9 月.