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

山口, 大介
九州大学

勝木, 裕二
九州大学

松永, 裕介
九州大学

出版情報：情報処理学会研究報告, 2004-ARC-157, pp.91-96, 2004-03. 情報処理学会ARC研究会
バージョン：

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

山口大介[†]　勝木裕二[†]　松永裕介[†]

筆者らは，PLC のような 1 ビット論理演算命令を多用するアプリケーションに適したアーキテクチャLEP(Logic Evaluation Processor) および命令セットアーキテクチャLIME(Logic Instruction Multi Evaluation) を提案している．LEP は，複数の変数データをプロセッサ内部に蓄え，かつ複数リテラルの論理式を一度に評価可能なモジュールをもっており，これらの機能を十分に使用することでデータメモリおよび命令メモリへの冗長なメモリアクセス回数を削減をするのが LEP のねらいである．

本稿では，LEP の性能評価を行い，その有効性についての考察を行う．

An Evaluation on the Processor Archtecture for a Programable Controller

Daisuke Yamaguchi ,[†] Yuji Katsuki [†]
and Yusuke Matsunaga [†]

We presently propose the processor named *LEP(Logic Evaluation Processor)* and the instruction set architecture named *LIME(Logic Instruction Multi Evaluation)* for the Programable Controller(PLC) which mainly uses the logic instruction of 1 bit. LEP has some modules which can load(or store) some data at once and can evaluate a logic expression at once. In this paper, we describe outline of the LEP and the LIME. Also we evaluate the performance of the LEP.

1. はじめに

現在，工場などの産業分野では FA(Factory Automation) 化が行われており，その中心技術として自動制御技術が重要な役割を果たしている．自動制御技術の中でシーケンス制御は，処理の複雑化および高度化の要請により，マイクロプロセッサ技術を応用したプログラマブルコントローラ（以下，PLC) と呼ばれる制御機器が幅広く使用されている[1),2)]．

現在 PLC が抱えている問題の一つに、メモリアクセスの冗長性が挙げられる．PLC は，一般的にラダー図から変換された論理演算命令を実行することで対象物を制御している．論理演算命令で必要とする情報量は 1 ビットであるのにも関わらず，現在 PLC の既存プロセッサでは 1 ワード（16 ビット）単位のメモリアクセスが発生している．そのため，同アドレスのデータにも関わらずビット指定が違うだけで複数回のメモリアクセスが必要となる．また，既存プロセッサの代表的な構成であるアキュムレータ型のプロセッサでは，内部で複数データを蓄えることができない．これによっても，複数回のメモリアクセスが必要となる．そこで，我々の研究グループでは，PLC のような論理演算を多用するアプリケーションに対して，効果的なメモリアクセスを実現するプロセッサ・アーキテクチャの開発を行っている[3)]．

冗長なメモリアクセスを削減するため，筆者らは下記のような PLC のプログラムの特徴に着目した．

- PLC は論理演算命令が 9 割以上を占める
- メモリアドレスに対する参照の局所性がある

上記の 2 点への対策として，LEU(Logic Evaluation Unit) とバッファというモジュールを内部にもつプロセッサ LEP(Logic Evaluation Processor) を提案している．LEP は，複数リテラルの論理式が一度に評価可能であり，複数のメモリデータを内部に蓄えておくことができるプロセッサである．また同時に筆者らは，これらの機能を用いるために LEP 用命令セットアーキテクチャLIME(Logic Instruction Multi Evaluation) も提案している．この命令セットアーキテクチャLIME によって，同一アドレスの複数ビットを同時にバッファ内に蓄え，一命令で 4 リテラルまでの論理式を評価することが可能である．

本稿では，上記で説明した LEP，LIME の概要を

† 九州大学 Kyushu University

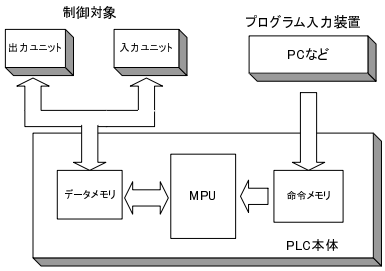

図 1 PLC の概要

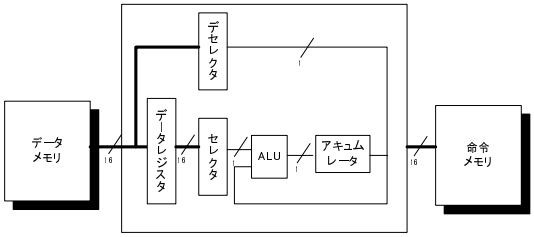

図 2 MPU の内部構成

説明し，論理演算命令から LIME 命令列への変換方法を述べる．また，評価実験によって既存のアキュムレータ型プロセッサと比較したメモリアクセス回数の削減率を示す．本稿の構成は下記のとおりである．まず 2 章で PLC の説明および代表的な既存プロセッサの構成を述べ，3 章で筆者らが提案しているプロセッサ LEP，およびその命令セットアーキテクチャLIME の概要を示す．4 章では，論理式から LIME 命令列への変換方法について説明した後，5 章で，提案プロセッサの評価実験を行い，6 章でまとめる．

2. プログラマブルコントローラ（PLC）

本章では，プログラマブルコントローラ（PLC）の概要と，内部に組み込まれている既存プロセッサの概要を示す．

2.1 PLC とは

PLC とは，シーケンス制御を行うために特有の命令や機能をもち，プログラマブルなメモリを内蔵する制御装置のことある[2)]．プログラム言語としては，リレー回路を基に作成された「ラダー図」が一般的に使用されている．ユーザがラダー図を用いて制御内容を書き換えれば思い通りの制御が可能になる．制御方法としては，ユーザが作成したラダー図と同等なニーモニックを生成して，それを内部に組み込まれたプロセッサが実行することで制御を行っている．ニーモニックは主に論理演算命令が全体の 9 割を占め，その他の命令として算術命令やタイマー命令などがある．PLC の内部は，プログラムや機器の接点情報を格納するメモリ，プログラムを実行する MPU(Micro Processor Unit)，および，外部との通信を行う通信モジュールから構成されており，通常の組み込みシステムと同様の構成になっている．PLC の構成図を図 1 に示す．

2.2 MPU（Micro Processor Unit）の構成

内部の既存プロセッサの内部記憶方式は，昔から図 2 のようなアキュムレータ・アーキテクチャになっている場合が多く，命令セットも 1 オペランドである．オペランドには，直接メモリのアドレスを指定するため各命令でデータメモリへのアクセスが発生する．データメモリのアドレスにはビット単位で指定されるが，データはワード単位で転送され，プロセッサ内部でセレクタによってビットを取り出す．なお，データを 1 ビット書き戻す際は，一度 16 ビット単位で指定したアドレスをプロセッサ側に読み出し，必要なビットを書き換えた後，16 ビット単位でメモリに書き戻すという手順で行われるため，メモリアクセスは 1 命令で 2 回発生する．

ここでは，ワード単位でのアドレス空間を「ワードアドレス」，ビット単位でのアドレス指定を「ビットアドレス」と呼ぶことにする．また，ビットアドレスで指定するメモリアドレスを単に「変数」と呼ぶこともある．

2.3 既存 MPU での問題点

現在，PLC が抱えている問題としてメモリアクセスの冗長性が挙げられる．前節でも説明したように PLC は，ラダー図から論理演算命令に変換することで制御対象を制御している．論理演算命令ではメモリ上にあるスイッチ情報や一時的に使用するメモリデータを読み込んで処理しているが，論理演算で必要な情報量は 1 ビットであるにも関わらず 1 ワード（16 ビット）単位でのメモリアクセスが発生するため，冗長なメモリアクセスが発生している．また既存の MPU は，アキュムレータ型であるため，メモリデータを 1 つしか格納できない．そのため，データを一時的に格納することもできないため，同一データにも関わらず複数回のメモリアクセスが必要とする場合もある．

そこで，本稿では PLC のメモリアクセス回数に着目しメモリアクセス回数を削減するアーキテクチャを提案する．

3. 提案プロセッサ LEP

2.3 節で述べた問題点を解消するため，筆者らは LEP(Logic Evaluation Processor) と呼ぶ論理式を高速に行うことが可能なプロセッサ・アーキテクチャを

提案している．本章では，LEP の特徴を示すと同時に LEP 用命令セットアーキテクチャである LIME(Logic Instruction Multi Evaluation) の概要を示す．

3.1 LEP の特徴

筆者らは，PLC の主な命令が論理演算命令であること，および同じ変数を複数回使用する場合があるなどの事実から下記に示すような特徴をもつプロセッサ・アーキテクチャを提案している．[3)]

(1) 複数のメモリデータを一時的にプロセッサ内に記憶しておくことが可能

(2) 複数リテラルの論理式を一度に評価可能

上記の 2 点の特徴をもつアーキテクチャにより，データメモリおよび命令メモリ両方のメモリアクセス回数を削減するのが目標である．

3.2 LEP 概要

LEP は，下記の 2 つのような特徴的なモジュールをもつアーキテクチャである．

- バッファ
- LEU(Logic Evaluation Unit)

LEP の概念図を図 3 に示す．データ処理は，バッファを介してアクセスされ，その後 ALU(Arithmetic and Logical Unit) もしくは LEU(Logic Evaluation Unit) で演算を行う．論理演算は LEU というモジュールを用いて評価を行い，四則演算などの処理は ALU を用いて行う．

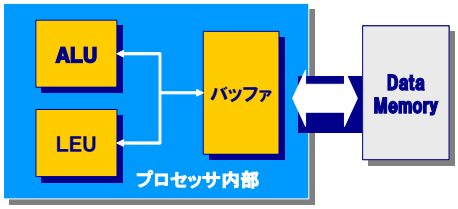

図 3 LEP の概念図

3.2.1 バッファ

バッファは，複数のデータを保持しておくためのモジュールである．バッファには複数のビットデータが格納できるようになっており，バッファ内のビット指定はバンクアドレスとオフセットという 2 つのアドレスで指定する．ここで，バッファ内部のビット単位の記憶部のことを，ここでは「レジスタ」と呼ぶことにする．内部にはバンクレジスタというレジスタが存在し，命令でオフセットを指定するだけで，所望のデータを取り出せる仕組みになっている．バッファの利点は，複数データの記憶が可能なのでデータメモリとのやりとりが削減できる点と，すでにデータがバッファ内にあればロード命令やストア命令を削減できるため命令メモリへのアクセス回数も削減できる点である．

現在，想定しているバッファサイズは，後述する命令セットアーキテクチャの関係で 256 個のデータを記憶できる．その為，バンクアドレス 4 ビットとオフセット 4 ビットで任意のバッファアドレスを指定する．

3.2.2 LEU

図 4 に LEU の構成図を示す．LEU は，論理式評価に特化したモジュールであり，複数リテラルの論理式が一度に評価可能なモジュールである．そのため，変数がバッファ内にあるならば一命令で処理することが可能であり，その分命令メモリへのアクセス回数が削減できる．LEU で一度に評価できるリテラル数は，LEU の基本素子である LU（Logic Unit）を組み合わせればどのようにも対応することが出来るが，現在想定しているのは 4 つまでとしている．

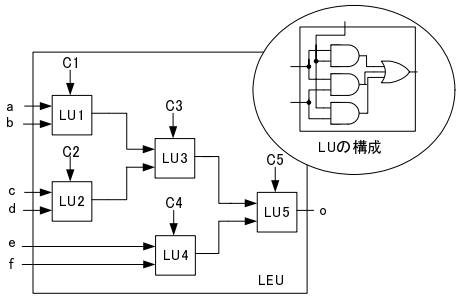

図 4 LEU の構成図

3.3 LEP 用命令セットアーキテクチャLIME

LIME の命令セットは，表 1 に示す．LIME の命令セットは，大きく分けて下記の 3 つに分類できる．

- メモリアクセス命令
- 論理評価命令
- その他の命令

メモリアクセス命令とは，メモリからバッファへの読み込み命令（LD 命令），バッファからメモリへの書き込み命令（ST 命令）などメモリとのデータのやりとりに使用される命令である．特に LIME では，同一ワードアドレスの異なる変数に関して同時に 3 ビットまで転送可能な命令（MLD 命令，MST 命令）を持つ．MLD，MST 命令に関しては，2 つをまとめて「マルチ転送命令」と呼ぶことにする．マルチ転送命令の詳しい説明に関しては，後述する．また，今後 MLD 命令，MST 命令を含む命令列を「LIME-M 命令列」と呼び，含まずに LD 命令および ST 命令のみの命令列を「LIME-S 命令列」と呼ぶことにする．

次に論理評価命令は，論理式の評価を行うための命

令であり，LEU で評価を行う命令（EX 命令，EX2 命令）とバンクレジスタの値を変更する命令（BANK 命令）がある．BANK 命令の役割に関しては，後述する．現在，EX 命令は命令フォーマットの関係で 1 命令で 4 リテラルの論理式まで 1 度に評価可能である．5 リテラル以上の論理式の場合は，論理式を分割して評価を行う．EX 命令と EX2 命令との違いは，EX 命令が 4 リテラルまでの論理式が評価可能なのに対して，EX2 命令は 2 リテラルまでの論理式しか評価できない．しかし，EX 命令ではバッファアドレスの指定ができないので BANK 命令が必要なのに対して，EX2 命令はバンクアドレスの指定ができるので BANK 命令を必要とせず済むという利点がある．

最後に上記 2 つに含まれない命令に関しては，その他の命令として扱う．タイマー命令や比較命令，加減算命令などは，これに当たる．論理演算には関係の無い命令は，こちらの分類になる．

表 1 命令セット

LD 命令	データをバッファに格納
ST 命令	バッファのデータをメモリに格納
MLD 命令	アドレス内の複数のデータをバッファに格納
MST 命令	バッファ内の複数のデータをメモリに格納
EX 命令	LEU で論理式の評価を行う
EX2 命令	LEU で 2 リテラルの論理式評価を行う
BANK 命令	演算のバンクを切替える
その他の命令	論理演算以外の命令

3.3.1 マルチ転送命令

MLD 命令は，変数のワードアドレスが一致しているならば同時に最大 3 変数までを一度にバッファに格納することが可能な命令である．同様に，MST 命令も変数のワードアドレスが一致しているならば最大 3 変数までを一度にバッファからメモリに転送することが可能である．これにより，データメモリへのアクセス回数を削減することが可能になる．ただし，マルチ転送命令は，命令語長の関係で 2 ワード命令となっており，一命令発効するためには 2 回の命令メモリへのアクセスが必要となる．しかし，通常の LD 命令，ST 命令を 2 回発行する場合に比べればデータメモリのアクセス回数が半分になるという利点がある．また 3 回発行する場合に比べれば，データメモリへのアクセス回数は 1/3，命令メモリへのアクセス回数は 2/3 になる．

3.3.2 BANK 命令

LIME のもう一つの特徴的な命令として BANK 命令がある．この命令は，EX 命令とセットで使用される．EX 命令は命令語長の関係でバッファ内の全てのアドレスを指定することが出来ない．そこで，バッファのアドレスを「バンクアドレス」と「オフセット」で指定することにしている．このうち，EX 命令では「オフセット」のみを指定し,「バンクアドレス」の指定には BANK 命令を使用する．コンパイル時にレジスタ割り当てを行うことによって，予め使用するバッファレジスタが決まるので，これを基に BANK 命令が発効される．また，BANK 命令で指定した「バンクアドレス」の値は，バンクレジスタに格納され，一つ前の EX 命令のバンクアドレスと相違無ければ BANK 命令を発効する必要がなくなる．

4. LEP 用コンパイラ

LEP および LIME の性能評価を行うためには，既存 MPU の命令列から LIME-M の命令列に変換する必要がある．そこで，今回既存 MPU の命令列から LIME-M 命令列に変換する LEP 用コンパイラを作成したので，その内部処理について説明する．

4.1 概　　要

LEP 用コンパイラでは，既存 MPU の命令列（ニーモニック）から生成される論理式から LIME-M 命令列への変換を行う処理を行っている．既存のアキュムレータ型命令列から LEP のようなレジスタ型命令列に変換するにあたり，下記を制約条件として変換を行う．

- メモリ上のデータ位置は変更しない
- 論理式の評価順序は変更しない
- 論理の接続のための変数は，メモリに書き戻さない

4.2 処 理 手 順

下記に LIME-M 命令列への命令変換手順を述べる．当然，この変換の入力はニーモニックから生成される論理式であり，出力は LIME-M 命令列である．命令への変換には，下記の 2 つの手続きが行われる．

(1) 論理式から LIME-S 命令列への変換（LIME-S 化）
(2) LIME-S 命令列から LIME-M 命令列への変換（LIME-M 化）

以下に，各手続きの詳細を述べる．

4.2.1 LIME-S 化

LIME-S 化は，与えられた論理式をマルチ転送命令を含まない命令列（LIME-S 命令列）を変換する処理である．LIME-S 化の手続きでは，主に大きく 3 つの処理を行っている．

(1) 論理式の分割
(2) レジスタ割り当て
(3) LIME-S への命令変換

LEP は 4 リテラルの論理式までしか一度に評価することができないので，5 リテラル以上の論理式が与えられた場合は論理式の分割を行わなければならない．分割した際は，論理の意味を変更せずに途中の演算結果をレジスタに記憶させておくことで分割する．そのため，分割を行うと EX 命令の数が増えレジスタを余計に使用するというオーバーヘッドがかかる．論理式を分割した後は，各論理式に割り当てられている変数をバッファ内のレジスタに割り当てなければならない．通常のコンパイラでは，種々のレジスタ割付け法が使用されている[4] が，今回行った変換では，順次割り当てによる方法で行った．この方法は，対象とするプログラムの範囲の先頭から空いてるレジスタがある限り，順次割り当てていく方法である．問題は，すべてのレジスタが割り当てられた場合だが，今回は次回参照されるまでが一番遅いレジスタの変数を待避（spill）させて，レジスタの数を確保した．

レジスタ割り当てを行った後，LIME-S 命令列の出力を行う．LIME-S 命令列を出力する場合，下記の規則をもとに各命令を出力した．

- LD 命令は，変数が始めて使用される場合か，一度メモリに待避させたデータを読み込む場合に発行する
- ST 命令は，データが書き換えられ，最後に使用したときに発行する
- EX 命令・EX2 命令は，論理式を評価する場合に発行する
- BANK 命令は，前回の EX 命令と比較してバンクアドレスに変更があった場合に発行する

4.2.2 LIME-M 化

LIME-M 化とは，LIME-S 命令列から LIME-M 命令列への変換する処理である．

LIME-M 化は，同じワードアドレスをもつ変数の LD 命令や ST 命令をグループ化することでマルチ転送命令に変換する．しかし，全ての変数がマルチ転送命令に変換可能な訳ではない．LIME-M 化で注意しなければならない点は，変数の生死のタイミングが変更される点である．つまり，MLD 命令化される場合は，変数の生まれるタイミングが一番早いもの LD 命令と同じになり，逆に MST 命令化する場合は変数の死ぬタイミングが一番遅いもの ST 命令と同じになる．ここでもし，マルチ転送命令に変換することによってレジスタが別の変数に割り当てられていた場合は，LIME-M 化を行うことができない．また，MLD 命令や MST 命令は，レジスタの使用区間を変更するので，変換する順序によって MLD 命令や MST 命令の数が異なる場合もある．

今回，LIME-M 化は以下の手順で行った．まず LIME-S の命令列からメモリアクセス命令を調べて各変数の生存区間を計算する．次に，同じワードアドレスを持つ変数をグループ化し，レジスタの使用区間を考慮してマルチ転送命令に変換可能かをチェックする．マルチ転送命令へ変換する場合は，3 変数の MST 命令化，2 変数の MST 命令化，3 変数の MLD 命令化，2 変数の MLD 命令化の順に行った．

5. 評 価 実 験

LEP および LIME の有用性を示すために，下記のような実験を行った．

5.1 実 験 概 要

3 つのラダー図プログラム（Program A，Program B，Program C）をサンプルとして用いて，実験を行う．これらのサンプルプログラムは，実際工場などで使用されており，プログラムサイズはそれぞれ異なる．各サンプルプログラムのデータメモリおよび命令メモリへのアクセス回数は，表 2 に示す．N_{DM},N_{IM} はそれぞれデータメモリへのアクセス回数，命令メモリへのアクセス回数を表している．

LIME 命令列への変換は，4 章で説明した手順で行い，命令のスケジューリングなどは行わない．また，LIME 命令列の各メモリへのメモリアクセス回数（N_{DM},N_{IM}）の算出法に関しては，以下の計算式で行う．

$$N_{DM} = N_{LD} + N_{MLD} + (N_{ST} + N_{MST}) * 2$$
$$N_{IM} = N_{INS} + N_{MLD} + N_{MST}$$

N_{INS} は全体の命令数，N_{LD},N_{MLD},N_{ST},N_{MST} はそれぞれ LD 命令，MLD 命令，ST 命令，MST 命令の数を表している．

表 2 各サンプルプログラムのデータ

	Program A	Program B	Program C
N_{IM}	4717	13173	8030
N_{DM}	5098	13852	9003

5.2 実験結果と考察

実験結果を表 3 , 表 4 に示す．表 3 は，各サンプルプログラムで現れた論理式から LIME 命令列に変換したときの各命令数の出現数を示している．表 4 は，表 3 を基にデータメモリと命令メモリへのアクセス回数を算出した結果である．

LIME-S 命令列から LIME-M 命令列に変換した場合，表 4 をみると命令メモリへのアクセス回数は，平均 2.1% の削減しか行うことが出来なかった．この理

表 3 各命令の出現数

	Program A		Program B		Program C	
	S	M	S	M	S	M
LD	491	290	2141	1108	2187	1484
MLD	0	81	0	394	0	298
ST	478	384	1391	824	1526	1038
MST	0	42	0	227	0	187
Total	3907	3735	12459	11480	10606	9900

表 4 メモリアクセス回数の結果

	Program A		Program B		Program C	
	S	M	S	M	S	M
N_{IM}	3907	3858	12459	12101	10606	10385
N_{DM}	1447	1223	4923	3604	5239	4232

由は，マルチ転送命令によって命令メモリへのアクセス回数が削減されるのは，LD 命令 3 回分もしくは，ST 命令 3 回分をマルチ転送命令に変換したときにのみである．また，このとき削減されるアクセス回数は 1 回なので，LIME-M 化によって命令メモリへのアクセス回数の削減には大きな効果を得ることが出来なかったと考えられる．しかし，データメモリに関して考察すると，アクセス回数は，平均 20.5% の削減を行うことができていた．これより，マルチ転送命令に変換は，データメモリへのアクセス回数の削減に大きく貢献していると考えられる．

次に元のサンプルプログラムから LIME-M 命令列に変換した場合を調べると，データメモリのアクセス回数が平均で 67.7% のアクセス回数を削減しているという結果であった．この理由は，同じデータをバッファに蓄えておくことで，変数を複数回データメモリにアクセスする必要がなくなったためだと考えられる．つまり，バッファによって確実にデータメモリへのアクセス回数を削減しており，この点では LEP および LIME の有効性が示せたのではないかと考えられる．しかし，命令メモリに関しては，Program C のみ命令メモリへのアクセス回数が増えてしまう結果となった．この原因を調べた結果，Program C は論理式内で同じ変数を使用する頻度が少ないため，バッファ内のデータを効果的に使用できなかったためと思われる．ただし，他の Program A，Program B に関しては，約 10% 前後の削減が行われており，命令メモリへのアクセス回数も削減が可能であることは分かった．

6. おわりに

本稿では，論理演算命令を多用するアプリケーションをターゲットとしたプロセッサアーキテクチャLEP と LEP 用命令セットアーキテクチャLIME の説明を行い，論理式から LIME 命令列への命令変換手法を述べるとともに実験によってメモリアクセス回数の見積もりを行った．実験では，データメモリへのアクセス回数は平均で 67.6% の削減が期待できたが，命令メモリへのアクセス回数に関しては逆に増えてしまうケースもあった．ただし，マルチ転送命令の数や BANK 命令の数は，バッファ内のレジスタ割り当てに依存する部分が大きいため，より効果的なレジスタ割り当てを検討することで更に多くの LIME-M 化や BANK 命令の削減が可能であると考える．そのため，今以上の命令メモリへのアクセス回数削減も期待出来ると考えている．

今後の課題としては，まず上記で示した効果的なレジスタ割り当てを検討し，命令メモリのアクセス回数削減を行う．また今回，命令実行時間に関しては何も触れていない．本来ならば議論しなければならない点であるが，今回はメモリアクセス回数のみで性能評価を行ったので，命令実行時間も含めた性能評価を行い効果的なスケジューリング手法を検討することも，今後の課題である．

謝辞 サンプルプログラムを提供して頂いたオムロン株式会社の方々に深く感謝の意を表します．

参 考 文 献

1) 望月傅, “図解でわかるシーケンス制御の基本,” 技術評論社, ISBN4-7741-0681-X C3054.
2) 関口隆, “プログラマブルコントローラ応用技術ハンドブック,” 電気書院, ISBN4-4-485-71608-2.
3) 山口 大介 and 松永 裕介, “プログラマブルコントローラ向けアーキテクチャの検討と評価”, 電子情報通信学会技術研究報告, CPSY2002-108, pp.19-24, Mar. 2003.
4) 中田育男, “コンパイラの構成と最適化,” 朝倉書店, ISBN4-254-12139-3 C3041.