ラダー図開発ツールLD Cv！

Version1．5取扱説明書（初版）

■LD Cv！とは

LD Cv！は、シーケンス制御を記述するためのプログラミング言語であるラダー図（Ladder Diagram）の開発ツールです。これを使って開発したプログラムは最終的にROM化して、I／Oポートのあるマイクロコンピュータ上で実行させることを想定しています。

LD Cv！の機能の本質は、ラダー図を編集するエディタ機能と、ラダー図で記述された制御ロジックを等価なZ80アセンブリ言語またはC言語プログラム（テキストデータ）に変換する変換機能です。扱えるラダー図容量は、横7シンボル×縦1000行です。

変換前言語	変換後言語	ターゲットマシンのCPU	LD Cv！以外に必要なツール
ラダー図	Z80アセンブリ言語	Z80、または互換CPU	Z80アセンブラ、リンカ
ラダー図	C言語	あらゆるCPU	Cコンパイラ、リンカ

LD Cv！で開発する場合、制御ロジック以外の部分（電源投入時の初期設定やI／Oとメモリ間のデータ転送プログラムなど）は、開発者（プログラマ）が、ターゲットマシンの仕様に合わせて自分でプログラムを作る必要があります。LD Cv！が生成した制御ロジック部のプログラムと、プログラマが作成した制御ロジック部以外のプログラムを、アセンブラまたはCコンパイラにかけることで、ターゲットマシン上で実行可能なプログラムになります。

なお、LD Cv！が生成するプログラムには、自身の異常（CPUの暴走、メモリ変化など）を検出・処理させるコードは含まれていません。適用分野によっては、一般のマイクロコンピュータ応用システムと同様に、フェイルセーフの仕組みを追加する必要があります。

■動作環境

日本語版 Microsoft Windows95 で動作します。

Windows98/Me/2000/XP でも動作するはずですが、確認はしていません。

解像度は1024×768以上、メモリは Windows95 の場合で32MB 以上必要です。

動作させるには、次に示す Microsoft Visual Basic5.0 のRUNタイムライブラリ等が必要です。

COMDLG32.OCX　VB5JP.dll　MSVBVM50.dll　StdOle2.tlb　OleAut32.dll

OlePro32.dll　AsycFilt.dll　Ctl3d32.dll　ComCat.dll　CmDlgJP.dll

■インストールと削除

適当なZIP解凍ツールで解凍し、解凍したファイル全体を一つのフォルダに入れます。フォルダの中の ldcv.exe を実行することでLD Cv！が起動します。ラダー図の I/O 番号、コメントの文字が小さくて潰れて表示される場合は、コントロールパネルの「画面」を実行し、ディスプレイの詳細のフォントサイズを「大きいフォント」に変更してください。

削除する場合は、LD Cv！をフォルダごと削除してください。

■使用条件・保証範囲

LD Cv! Version1.5 はフリーソフトです。無償・無保証で提供されるソフトウェアと考えてください。

複製、再配布は、自由におこなってかまいません。

第1章　操作方法

■簡単な操作例

LD Cv！の操作を簡単な操作例を示しまします。

(1)LD Cv！を起動する

LD Cv！の実行ファイルldcv.exeを実行すると、ラダー図編集ウィンドウが開きます。起動直後のラダー図編集ウィンドウは、回路が何もないラダー図を読み出している状態あります。このときの画面上の左にある0は、未プログラムエリアの先頭の行番号を示しています。現在の行番号0に行カーソルがあるため背景が水色になっています。タイトルバーは、ファイル名とモードを表示しますが、起動直後はファイル名が「未定義．lad」、モードが「読出モード」になっています。(図1．1)

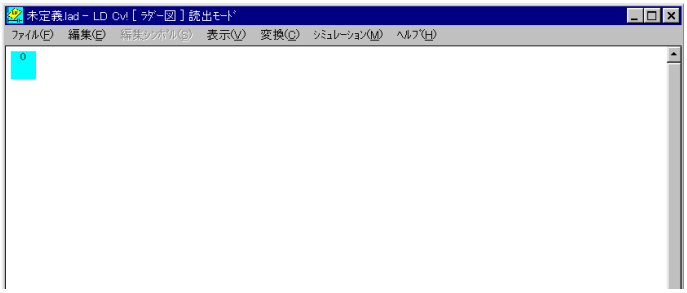


図1．1　LD Cv！起動直後(読出モード)

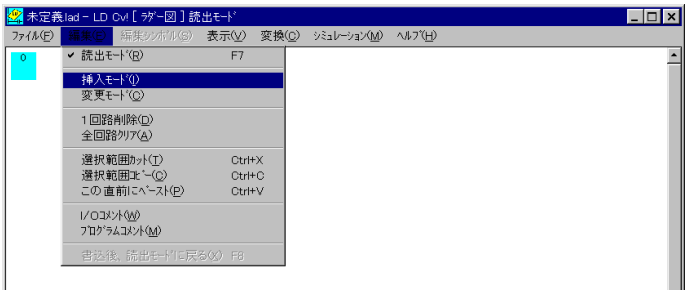


図1．2　挿入モードを選択

(2)[編集:挿入モード]メニューを実行する

回路を作成するため、まず[編集:挿入モード]メニューを実行します。これは、0行目に挿入するラダー図(1回路分)をこれから作成することを意味します。なお、ラダー図の編集作業は回路単位でおこなうため、1回路入力ごとに[編集:挿入モード]メニューを実行する必要があります。(図1．2)

[編集:挿入モード]メニューを実行すると、ウィンドウは、7行分の編集作業の画面に切り替わり、行番号は＋0から＋6を水色の背景で表示し、画面上にはセルカーソル(灰色のカーソル)が表示されます。このセルカーソルは、シンボルやI／Oアドレスをどこに入力するかという編集対象のセルを示すカーソルで、上下左右の矢印キーやマウスで目的のセルをクリックすることで移動できます。(図1．3)

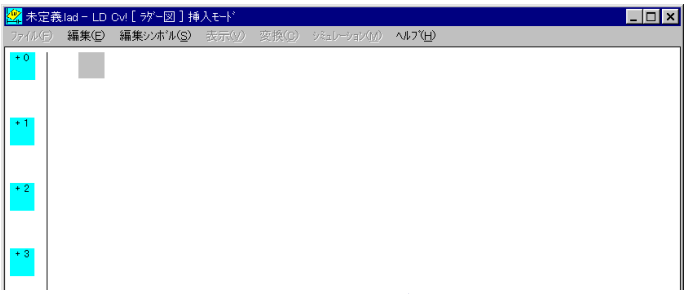


図1．3　編集モードに移行

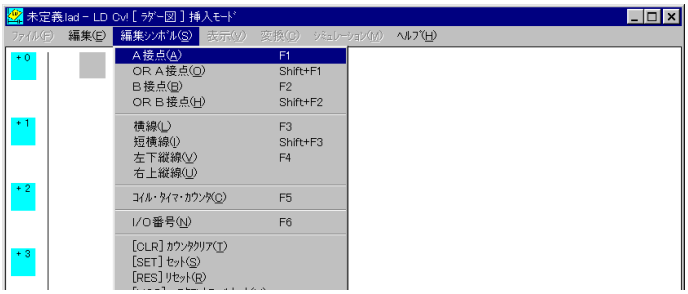


図1．4　A接点を選択

(3)[編集シンボル:A接点]メニューを実行

A接点を入力するため[編集シンボル:A接点]メニューを実行します。(図1．4)

すると、セルカーソル上にA接点を表示した後、I／O番号入力ボックスが表示されます。(図1．5)

I／O番号を入力します。OKボタンをクリックするとセルカーソルの位置にI／Oアドレスが表示され、セルカーソルは右へ1セル分移動します。(図1．6)

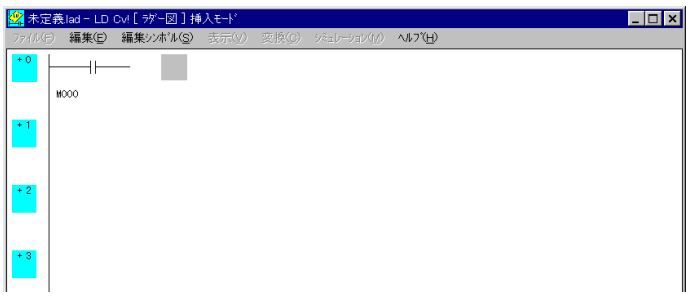


図1．5　I／O番号を入力

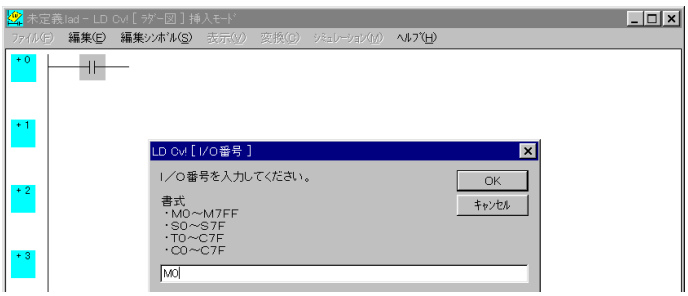


図1．6　I／O番号を入力した直後

(4)1回路を完成させる

このようにして、接点、コイル、横線、縦線、I/Oアドレス等を入力してゆき、1回路を完成させます。(図1．7)

図1．7　1回路を完成した直後

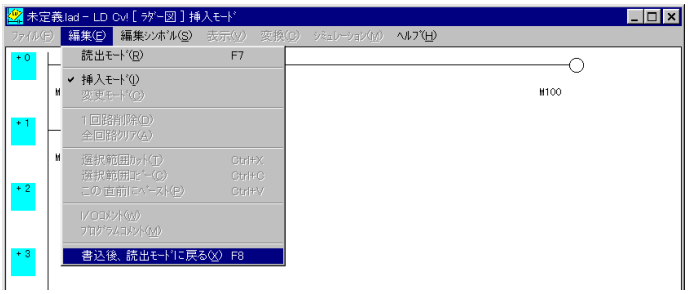


図1．8　書込後、読出モードに戻る

(5)[編集:書込後、読出モードに戻る]メニューを実行する

回路を完成させたら[編集:書込後、読出モードに戻る]メニューを実行します。(図1．8)

すると、LD　Cv！は回路を整形し、現在のラダー図に挿入して読出モードに戻ります。回路の挿入によって、以降の行番号は自動的に更新されます。さらに回路を入力する場合は、再び[編集:挿入モード]メニューを実行します。

(6)回路に誤りがあった場合

LD　Cv！が回路を整形した際に回路の誤りを検出したときは、エラーメッセージを表示し、モードは挿入モードのままです。回路の誤りを修正して再び[編集:書込後、読出モードに戻る]メニューを実行してください。

(7)編集を中止して読出モードに戻る場合

編集作業の途中で、編集を中止して読出モードに戻る場合は[編集:読出モード]メニューを実行します。この場合、現在のラダー図は変更しません。

(8)回路を変更する場合

すでに入力してある回路を変更する場合は、行カーソル(水色)を目的の回路(のどこでもよい)まで移動させた後、[編集:変更モード]メニューを実行します。以降は1回路入力モードと同じ操作です。行カーソルをダブルクリックしても変更モードに移行できます。

(9)変換条件を設定する

ラダー図が完成したら、変換実行をおこなう前に、どのような条件で変換するかの指示を前もって設定しておかなくてはなりません。この指示は[変換:変換条件ウィンドウを開く]メニューを実行し、開いたウィンドウ上でおこないます。(図1．9)

変換条件ウィンドウ上で各設定項目を入力しますが、既存の変換条件をファイルから読み込むこともできます。

ここでは、変換条件ウィンドウ上の[ファイル:変換条件をファイルから読込]メニューを実行し、既存のファイル Z80.cdt を読み込むことにします。

(10)ラダー図編集ウィンドウに戻る

変換条件を設定(または読み込み)したら、この変換条件ウィンドウを閉じて、ラダー図編集ウィンドウに戻ります。 この時点での変換条件は、変換条件ウィンドウを閉じる直前の状態になっています。

(11)ラダー図を変換し、ファイルに保存する

変換条件を設定したら、次に変換を実行します。変換は、ラダー図編集ウィンドウの変換メニューの[変換:変換実行]メニューを実行します。(図1．10) これで、変換が始まります。 変換が終了すると、ダイアログボックスが開き、保存場所を聞いてきますので、適当な場所にファイル名を付けて保存ボタンをクリックしてください。変換実行時には、自動的にラダー図チェックがおこなわれます。このチェックでエラーが検出された場合は、エラーメッセージが表示され、変換は中止されます。

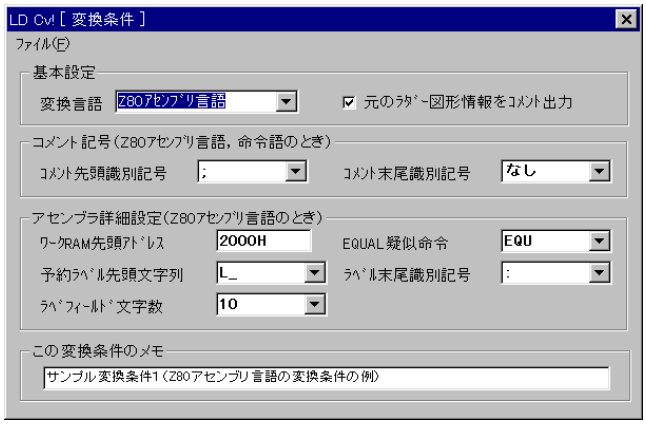


図1．9 変換条件ウィンドウ

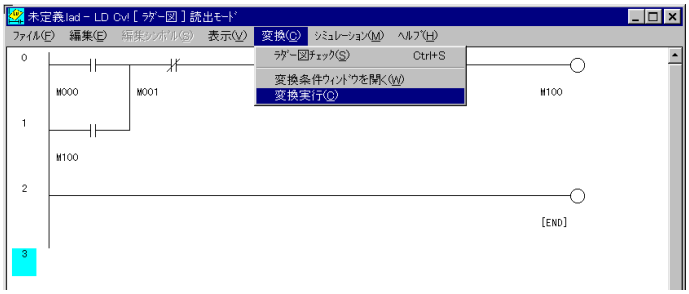


図1．10　変換実行する

(12)変換結果を見る

変換が正常に終了すれば、変換結果がファイルとなって保存されています。このファイルはテキストファイルなのでエディタで開いて見ることができます。 次のリストは、変換結果の一部分です。

```
L_EXEC:
;       |
;     0+----| |-----+----|/|-------------------------------------------------------------( )
;       |M000         |M001                                                          M100
;       |             |
;       |             |
;     1+----| |-----+
;       |M100
;       |
;[LD M000]
            RRA
            RL C
            LD A,(L_MO+000H)
;[OR M100]
            LD HL,L_MO+100H
            OR (HL)
;[ANDNOT M001]
            LD HL,L_MO+001H
            LD E,(HL)
             INC E
            AND E
;[OUT M100]
            AND B
            LD (L_MO+100H),A
```

■操作機能

表 1．1、表 1．2に「ラダー図ウィンドウ」上のメニュー機能を説明します。

表 1．1 ラダー図ウィンドウ上のメニュー機能（1／2）

メニュー		機能説明
ファイル	新規作成 開く 上書き保存 名前を付けて保存	プログラムファイルのロード・セーブに関するコマンド群です。 LD Cv！では、ラダー図、I／Oコメント、プログラムコメントが一体になったものをプログラムとして扱います。
	印刷：プログラム全体	プログラムコメントとラダー図を印刷します。
	印刷：I／Oコメント定義	I／O番号とI／Oコメント文字列の対応を印刷します。
	印刷：選択範囲	選択された範囲のラダー図を印刷します。選択方法は、カット・コピーで回路範囲を指定する方法と同じです。
	終了	LD Cv！を終了します。
編集	読出モード 挿入モード 変更モード 書込後、読出モードに戻る	LD Cv！起動直後は、読出モードです。 ラダー図を編集するには、挿入モードまたは、変更モードに移行する必要があります。挿入モードは、新規に 1 回路を編集し、カーソル位置の回路の直前に挿入します。変更モードは、既存の1回路を編集します。 編集結果は「書込後、読出モードに戻る」を実行することで、ラダー図に書き込まれます。 メニュー機能以外に、行番号をダブルクリックしても、変更モードに移行します。
	1回路削除	行カーソル（水色）上の1回路を削除します。
	全回路クリア	ラダー図をすべて消去します。 I／Oコメント、プログラムコメントは残っています。
	選択範囲カット 選択範囲コピー この直前にペースト	選択範囲の始点は行カーソル（水色）です。終点の指定方法は、読出モード時にシフトキーを押しながら上下矢印キーを押すか、シフトキーを押しながらマウスで終点の行番号をクリックします。これで、選択範囲の行番号の背景が黄色になります。 カットまたはコピーした回路は、LD Cv！のクリップボード（ファイルldcv．clp）に保存されています。
	I／Oコメントウィンドウを開く	I／Oコメントコメントウィンドウを開き、I／O番号とI／Oコメント（自由に定義できる文字列）の対応を表示します。I／Oコメントは、半角12文字以内で、全角を使用することはできません。 I／Oコメントコメントウィンドウ上では、特殊メモリのコメントの追加,、CSV形式のコメントデータのインポート、イクスポートが可能です。図 1．11に外部で作成したI／Oコメントデータの例を示します。
	プログラムコメントウィンドウを開く	プログラムコメントウィンドウを開き、プログラムコメント（自由に定義できる文字列）を表示します。 プログラムコメントウィンドウを開いた直後は、読出モードになっていて、変更できません。変更する場合は「コメント編集」メニューの「編集モード」コマンドを実行します。
編集 シンボル	A接点／OR A接点 B接点／OR B接点 横線／左下縦線／右上縦線 コイル・タイマ・カウンタ I／O番号 ［CLR］ カウンタクリア ［SET］ セットコイル ［RES］ リセットコイル ［MCS］ マスタコントロールセット ［MCR］ マスタコントロールリセット セルクリア／左下縦線クリア 画面クリア	シンボルやをI／O番号を入力・削除するコマンド群です。 タイマ番号1Fで設定値1234を入力する場合は、I／O番号入力ボックスには「T1F 1234」のように、タイマ番号と設定値の間に半角スペースを空けて入力します。カウンタも同様です。 メニュー機能以外に、セルカーソル上でダブルクリックしても、編集シンボルメニューをポップアップメニューで表示します。

表 1．2 ラダー図ウィンドウ上のメニュー機能（2／2）

メニュー		機能説明
表示	前回路／次回路／先頭回路 最終回路の次／指定行に移動 カーソル位置に移動	表示モードのとき、ラダー図中のどの部分を画面に表示するかを指定するコマンド群です。
	I／O検索 コイル検索	［I／O検索］では、指定された番号の接点とコイルの両方が検索対象です。［コイル検索］では、指定された番号のコイルが検索対象です。
	I／Oコメント表示切替：表示 I／Oコメント表示切替：非表示	I／Oコメントの表示／非表示を切り替えます。
	命令語表示ウィンドウを開く	命令語表示ウィンドウを開きます。 命令語表示ウィンドウは、現在カーソルのある回路の命令語を表示します。
変換	ラダー図チェック	現在のラダー図の構文をチェックします。エラーがあった場合、エラー番号とエラーの内容をダイアログボックスに表示します。
	変換条件ウィンドウを開く	変換条件ウィンドウを開きます。このウィンドウでは、ラダー図をどのような条件で変換するかの指示を設定します。変換条件は、ファイル保存でき、また保存した変換条件読込むこともできます。変換条件のデフォルトの拡張子は“cdt”です。
	変換実行	現在のプログラム（ラダー図）を、現在の変換条件にしたがって変換し、変換結果をファイル名を付けて保存します。 変換実行時には自動的にラダー図チェックがおこなわれ、エラーが検出された場合は変換を中止します。
シミュレーション	シミュレーション開始	LD Cv！上で、仮想PLCを実行（シミュレーション）開始します。 シミュレーション中は、接点とコイルのON・OFF状態と、タイマ・カウンタの経過値を表示します。 接点とコイルのシンボルはONのとき緑色になり、OFFのとき白色になります。ただし、MCS、MCRコイルは、その状態に関らず常に白色で表示します。また、タイマ・カウンタの経過値は括弧付きで表示します。
	シミュレーション終了	シミュレーションを終了し、通常の読出モードに戻ります。
	モニタ	シミュレーション中の、最大32点のI／O番号の状態を表示します。 モニタするI／O番号の指定は、メニューから指定、Enterキーを押す、灰色の長方形をクリック、の方法があります。
	Mセット・リセット／タイマ経過値変更 カウンタ経過値変更	シミュレーション中の指定した番号のMの値、タイマ経過値、カウント経過値を変更します。
ヘルプ	バージョン情報	LD Cv！のバージョン情報などを表示します。

	A	B	
1	M0	ﾋｼｮｳﾃｲｼ	
2	m1	PB1	…I／O種別が小文字でもかまいません。
3	M2	PB2	
4	M2	PB25	…定義済みのコメントが再定義された場合、後に定義されたコメント文字列を優先させます。
5	M4		…コメントが空欄でもかまいません。
6	M3	PB12	…I／O番号の順番はランダムでかまいません。
7	M4FF	ｱｰﾑｼｮｳｹﾞﾝLS1	…12行を超えるコメント文字列は、後半をカットします。（インポート時に、メッセージを表示します。）
8	S0	ｶﾞｲﾌﾞ10ms	
9	T7F	まちじかん	…全角文字のコメント文字列は、可能であれば半角に変換します。（インポート時に、メッセージを表示します。）
10	C00	ｺｽｳA	
11	M7ff	ﾃﾞﾊﾞｯｸﾞﾌﾗｸﾞ1	…I／O番号の16進数を表すA～Fの文字は小文字でもかまいません。

図 1．11 外部で成したI／Oコメントデータの例

第2章 仮想PLC

■仮想PLCの概念

LDCv！による開発では、I／O番号や特殊機能の番号・機能を定めた、仮想的なPLC（Programmable Logic Controller）を想定し、これに対してラダー図で制御プログラムを記述します。仮想PLCの仕様を表2．1に示します。

表2．1 仮想PLC仕様

項目	仕様
データメモリ容量	I／Oメモリ（2048点）、特殊メモリ（8点）、タイマ（128点）、カウンタ（128点） ラダーシンボル A接点、B接点、コイル、カウンタクリア、セット、リセット、マスタコントロールセット、マスタコントロールリセット、エンド
ラダーシンボル	A接点、B接点、コイル、カウンタクリア、セット、リセット、マスタコントロールセット、マスタコントロールリセット、エンド
命令語	LD，LDNOT，AND，ANDNOT，OR，ORNOT，ANDSTACK，ORSTACK，OUT，OUTT，OUTC，CLRC，SET，RES，MCS，MCR，END
タイマ	オンディレイアップタイマ、点数128点、設定値0～65535
カウンタ	アップカウンタ、点数128点、設定値0～65535 リセット方法は、同じカウンタ番号のCLRCコイルをON
特殊メモリ	タイマ基準クロック、1／10／100／1000／10000スキャン反転、運転1スキャンON、常時ON

■仮想PLCのI／O

仮想PLCで使用できるI／O（実体はメモリ）は、表2．2のとおりです。

各I／Oは機能によって4つのI／O種別に分類され、先頭にI／O種別をあらわすI／O種別記号を付けて記述します。

表2．2 仮想PLCのI／O

I／Oの種類	点数	表記	意味
I／Oメモリ	2048	M0～M7FF	1ビットのリード・ライト可能なメモリ
特殊メモリ	128	S0～S7F	それぞれ特殊な動作をおこなうメモリ
タイマ	128	T0～T7F	指定時間以上のON入力の継続でONとなるコイル
カウンタ	128	C0～C7F	コイルに指定回数以上の立ち上がりエッジ入力でONとなるコイル

（1）I／Oメモリ（M）

I／Oメモリは、接点またはコイルとして記述できる1ビットのリード・ライト可能なメモリです。

LD Cv！にとってI／Oメモリは、ラダー図のプログラムにしたがってリード・ライトされるメモリにすぎません。しかし、I／Oメモリと開発対象のI／Oポートとの間でデータの転送（I／Oリフレッシュ）を定期的におこなうことで、I／Oメモリは開発対象のI／Oポートとして動作します。

I／Oメモリのうち、入力ポートとI／Oリフレッシュをおこなう範囲のMは、ラダー図上では接点として記述できますが、コイルとして記述してはいけません。

誤ってコイルとして記述すると、せっかく入力ポートから取り込んだデータがラダー図実行によって上書きされてしまいます。

これに対して、出力ポートとI／Oリフレッシュをおこなう範囲のMや、単なるメモリとして使う範囲のMにはこのような制約はなく、接点としてもコイルとしても記述できます。

（2）特殊メモリ（S）

特殊メモリは、あらかじめ番号によって機能が決められている1ビットの容量を持つメモリです。

特殊メモリの各機能を示します。S0はコイルとしてのみ、S10～S14，S1E，S1Fは接点としてのみ使用します。S10～S14とS1Eの変化はスキャンとスキャンの区切りでおこなわれるため、同一スキャン中は値が変化しないことが保証されています。7点のみ定義済みで、残り120点は未定義です。

表2．3 特殊メモリの動作

I／O番号	機能名	動作
S0	タイマ基準クロック	このS0の変化（の時間間隔）で、タイマの経過値が＋1増加します。 この時間間隔をタイムベースと呼びます。
S10	1スキャン反転	1スキャン毎に値が反転します。
S11	10スキャン反転	10スキャン毎に値が反転します。
S12	100スキャン反転	100スキャン毎に値が反転します。
S13	1000スキャン反転	1000スキャン毎に値が反転します。
S1E	運転開始1スキャンON	運転開始後、1スキャン目は値1を、2スキャン目以降は値0をとります。
S1F	常時ON	運転開始後、常に値1をとります。

図2．1に、S12の動作を示します。

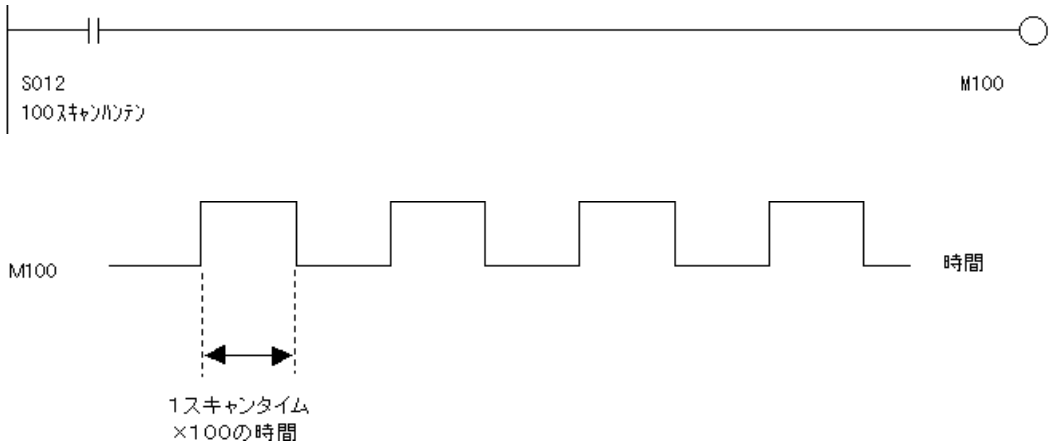


図2．1 S12の動作（100スキャン反転）

■ラダー図の意味

ここでは、ラダー図の意味について説明します。

（1）A接点

A接点は、そのI／O番号の値がONのとき導通となり、OFFのとき非導通になる接点です。それぞれのI／O番号について、接点はラダー図の中にいくあってもかまいません。図2．2に、A接点とコイルが接続された回路を示します。

図2．2 A接点

この回路を電気回路的にみれば、M0がONのとき、このA接点は導通となるためコイルM100は励磁されONになります。逆に、M0がOFFのとき、このA接点は非導通となるためコイルM100は励磁されずOFFになります。

（2）B接点

B接点は、そのI／O番号の値がONのとき非導通となり、OFFのとき導通になる接点です。論理回路でのNOT回路に相当します。それぞれのI／O番号について、接点はラダー図の中にいくあってもかまいません。図2．3に、B接点とコイルが接続された回路を示します。

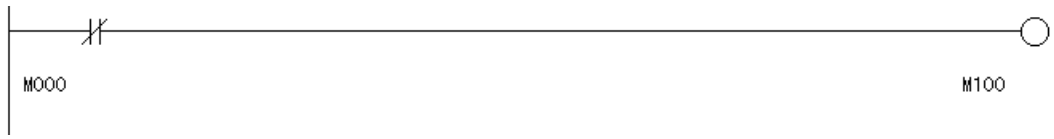


図2．3 B接点

この回路を電気回路的にみれば、M0がONのとき、このB接点は非導通となるため コイルM100は励磁されずOFFになります。逆に、M0がOFFのとき、このB接点は導通となるためコイルM100は励磁されONになります。M0を入力、M100を出力とするNOT回路と見なせます。

（3）コイル

コイルは、励磁（左母線と導通）されたときそのI／O番号の値がON（1）になり、励磁されないときそのI／O番号の値がOFF（0）になる素子です。したがって、ある番号のコイルがONになると、同じ番号のA接点は導通、B接点は非導通となります。それぞれのI／O番号について、コイルはラダー図の中に1つだけしか許されません。同じI／O番号のコイルが複数あると、ラダー図チェック時または変換実行時にエラーと判断されます。図2．4に、4つの接点とコイルが接続された回路を示します。

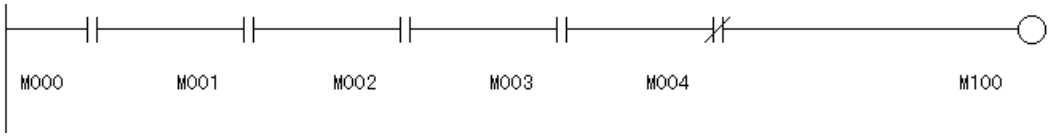


図2．4 コイル

この回路を電気回路的にみれば、M0、M1，M2，M3がON、かつM4がOFFのとき、コイルM100は励磁（左母線と導通）されONになります。これ以外のとき、コイルM100は励磁されずOFFになります。

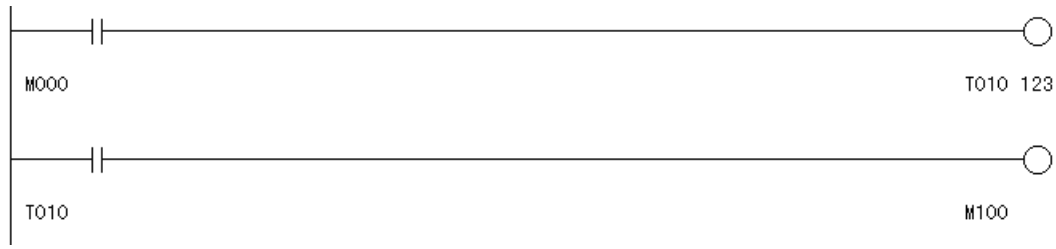
（4）タイマ（T）

タイマコイルへの入力がON（1）になると、タイムベース時間経過する毎にタイマの経過値がインクリメントされます。タイマの経過値が設定値に到達（すなわち、設定値×タイムベース時間が経過）するとタイマのコイルはONになります。逆に、タイマコイルへの入力がOFF（0）になると、ただちにそのタイマの接点はOFF（0）になり、経過値もゼロになります。

ラダー図上でタイマコイルを記述する場合、タイマアドレスと設定値（0～65535）を指定します。書式は次のとおりです。

タイマアドレス（T000～T07F）＋空白1文字＋設定値（0～65535）

図2．5に、タイマコイルとその接点の動作を示します。図中のtは、タイムベース時間を表します。



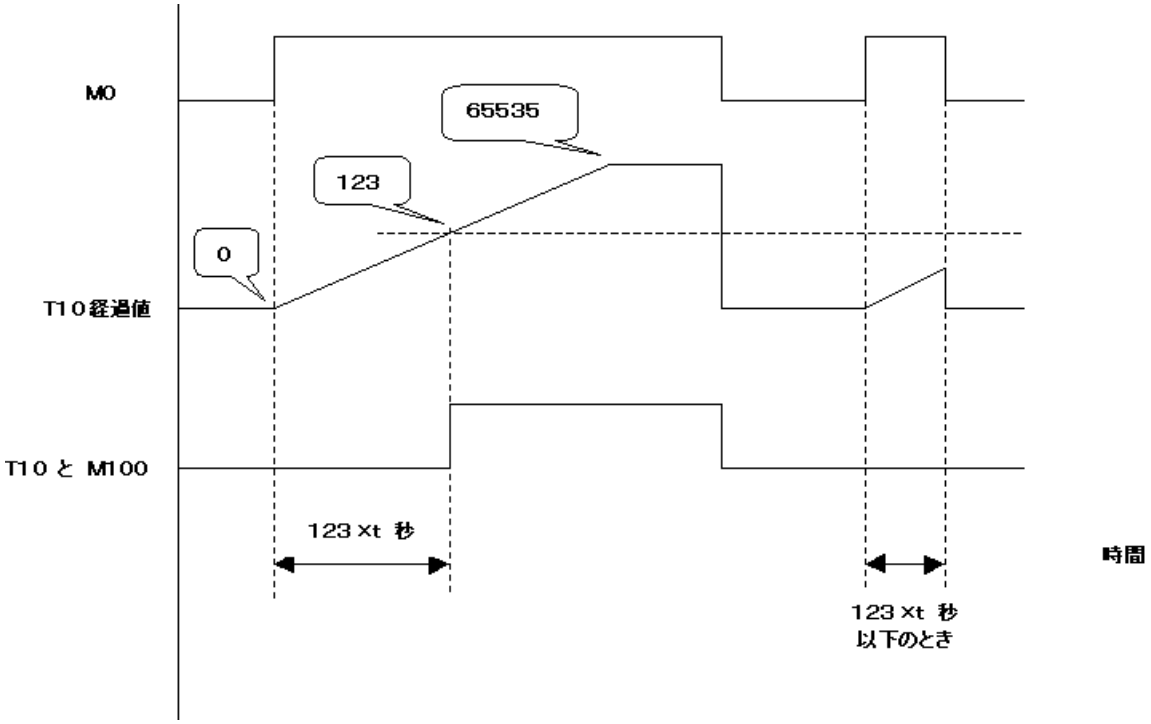


図2．5 タイマ動作

S0に200ms周期のクロックを与えれば、ONまたはOFFの時間は1周期の半分の100msなのでタイムベースは0．1sとな

り、0s～6553．5sまでの時間を計測できることになります。正確な時間が必要な場合は、外部プログラムで、毎スキャン毎に、S0を更新しますが、図2．6のようにラダー図上で、S10～S14（1スキャンタイム～10000スキャンタイム）のいずれかをS0のコイルに出力することで、タイムベースを書き込むことも可能です。

図2．6 ラダー図でタイムベースを与える方法

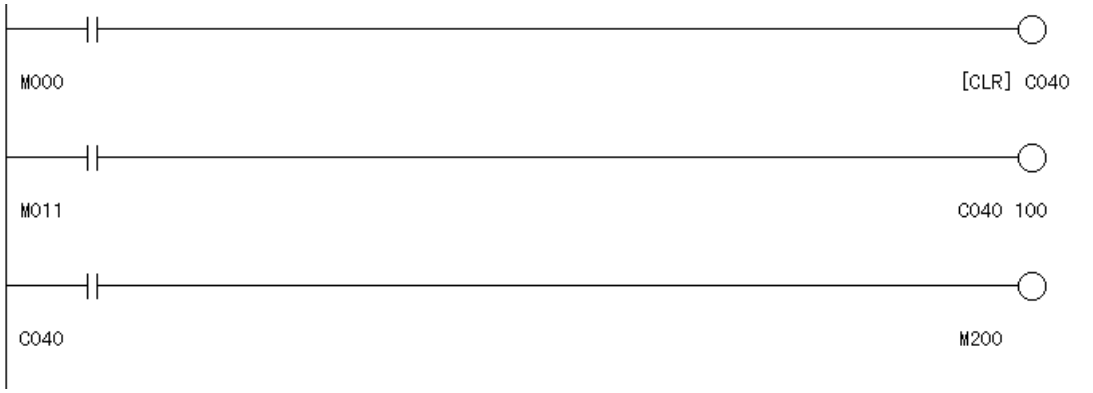
（5）カウンタ（C），CLRコイル

カウンタは計数動作をおこないます。カウンタコイルへの入力に立ち上がり入力（OFFからONへの変化）を与える毎にカウンタの経過値がインクリメントされます。カウンタの経過値が設定値に到達するとカウンタのコイルはONになります。

ラダー図上でカウンタコイルを記述する場合、カウンタアドレスと設定値（0～65535）を指定します。書式は次のとおりです。

カウンタアドレス（C000～C07F）＋空白1文字＋設定値（0～65535）

図2．7に、カウンタコイルとその接点の動作を示します。



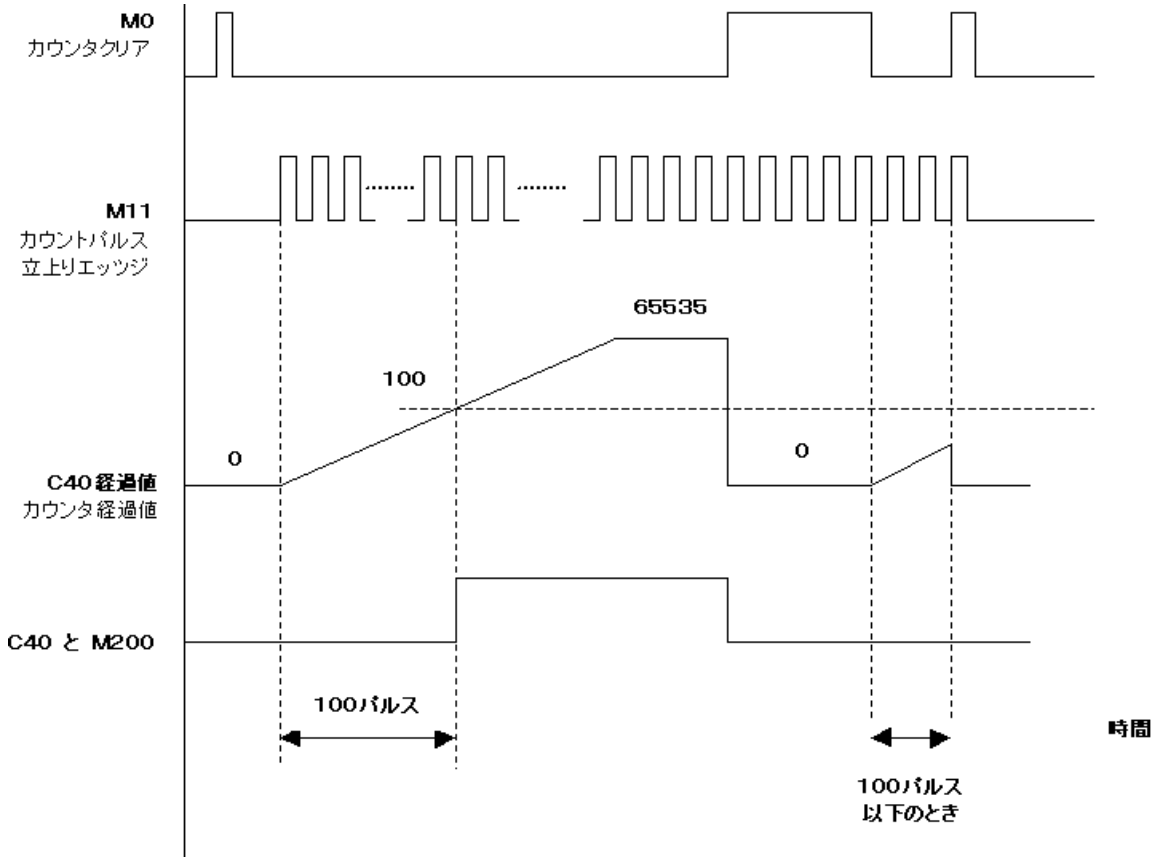


図2．7 カウンタ動作

カウンタにはカウンタクリア（[CLR]）が用意されています。カウンタクリアがON（1）になると（厳密には、そのスキャン中または

次のスキャンで同じカウンタ番号のコイルを最初に実行した時点で)、そのカウンタの接点はOFF(0)になり、経過値もゼロになります。カウンタとカウンタクリアコイルはペアで使用しますが、ラダー図中に並べて記述する必要はありません。

(6)SET, RESコイル

[SET](セット)は、励磁(左母線と導通)されたときそのI/O番号の値がON(1)になり、励磁されないときそのI/O番号の値が変化しない(以前の値を保持する)素子です。

[RES](リセット)は、励磁されたときそのI/O番号の値がOFF(0)になり、励磁されないときそのI/O番号の値が変化しない(以前の値を保持する)素子です。

同一I/O番号に対して[SET]と[RES]の両方を使用して、セット入力、リセット入力のある記憶回路を構成することができます。セット、リセットは工程歩進制御の状態を記憶するのによく使われます。

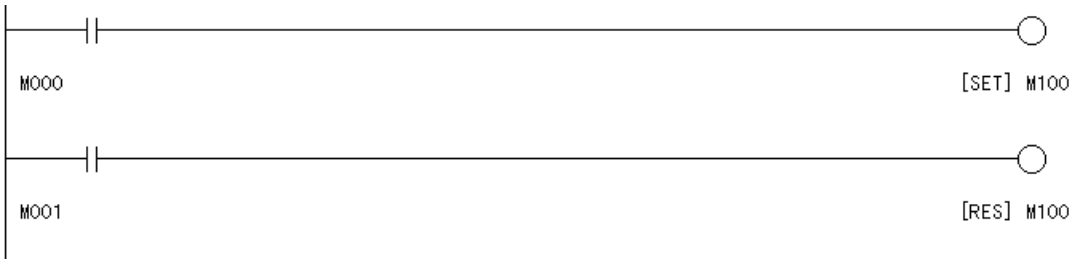


図2. 8 SET, RESコイル

この回路では、M0がONのとき、M100がONになります。また、M1がONのとき、M100がOFFになります。M0とM1の両方がOFFのときは、M100は以前の状態と保持(記憶)したままです。

(7)MCS, MCR

[MCS](マスタコントロールセット)は、ラダー図が連続した複数の回路にわたり共通の接点を使用しているときに、この共通の接点を1つにまとめて記述するために使用します。この共通の接点と[MCS]を接続した回路を記述すると、以降の回路ではこの共通の接点を省略して記述できます。この影響は、[MCR](マスタコントロールリセット)が出現するまで続きます。

ラダー図上で、[MCS]と[MCR]は対で使用されます。[MCS]と[MCR]が1:1に対応していなかったり、対応が入れ子構造になっていると、ラダー図チェック時または変換実行時にエラーと判断されます。[MCR]は、左母線から(途中に接点を入れずに)直結してコイルを記述します。

図2. 9に、共通の接点(M0、M1、M2の直列接続)を3回路にわたり使用している回路を示します。

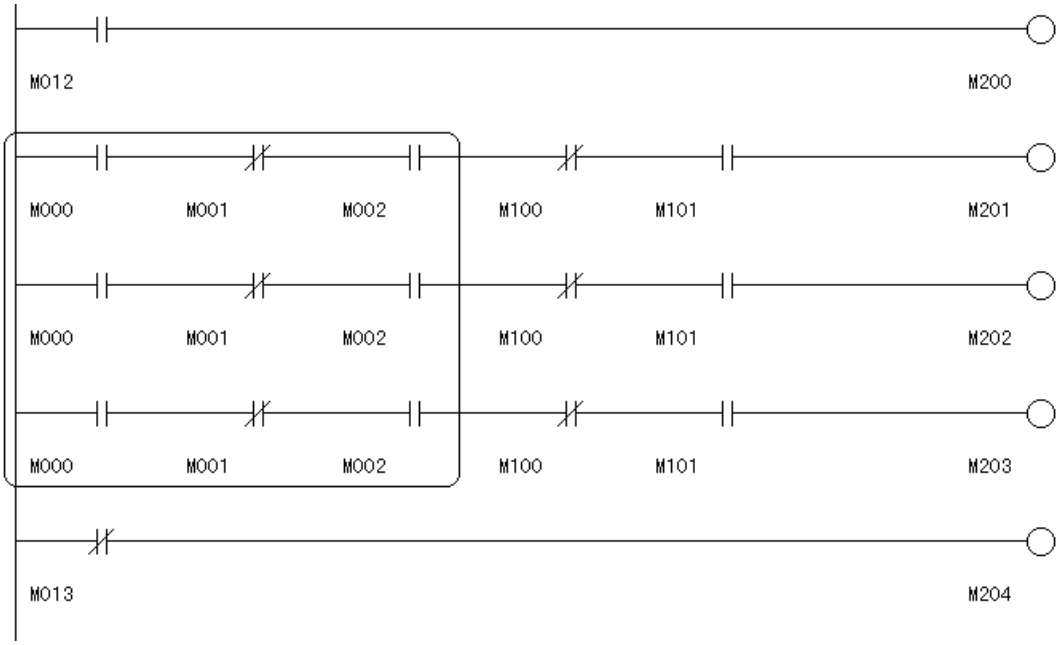


図2. 9 共通の接点(M0、M1、M2の直列接続)を3回路にわたり使用している回路

この回路は、MCS，MCRを使って、図2．10のように書き換えることができます。

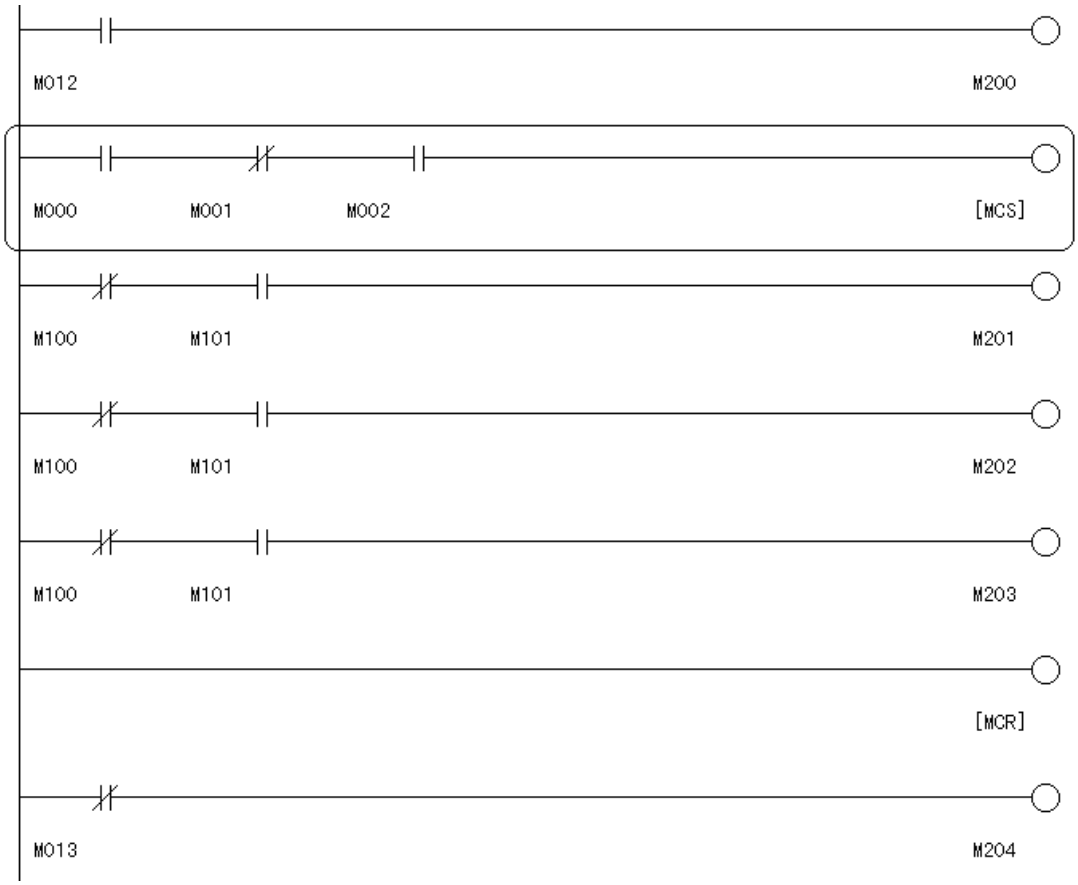


図2．10MCS，MCRを使って図2．9の回路を書き換えた回路

第3章 システムへの組込

ここでは、LD Cv！を使って、ターゲットマシンのI／Oポートを制御するプログラムを開発する方法を説明します。

ラダー図をZ80アセンブリ語に変換する場合と、C言語に変換する場合で、異なる部分があります。

■Z80アセンブリ言語に変換する場合

（1）手順1：入出力ポートとラダー図上のMとの対応を決める

まず、ターゲットマシンの入出力ポートの各ビット毎に、ラダー図上でどのI／O番号にするかを決めます。I／O番号は、M000～M7FFの2056点の中から適当に決めます。この段階では、単に対応を決めただけで、LD Cv！上で何か作業を行うわけではありません。ここでは、表3．1のようにI／O番号を割付けることにしました。

表3．1 入出力ポートとラダー図上のMとの対応

ターゲットマシンの入出力ポート	ラダー図上でのI／O番号
I／Oアドレス2DH番地のビット0～ビット7	M000～M00F
I／Oアドレス2FH番地のビット0～ビット7	M100～M10F

（2）手順2：目的とする制御をラダー図で記述

LD Cv！の編集機能を使って、目的とする制御のラダー図を作成します。

たとえば、ターゲットマシンの入力ポート（2DHのビット2）に入ってきた信号を、ビット反転して、出力ポート（2FHのビット7）に出力したければ、先程決めた表3．1の対応によって、M002の接点とM107のコイルを使って、図3．1のラダー図を作成します。ビット反転させるので、M002の接点は、b接点にします。

図3．1の2行目のENDは、ラダー図の末尾に入れる規則と考えてください。

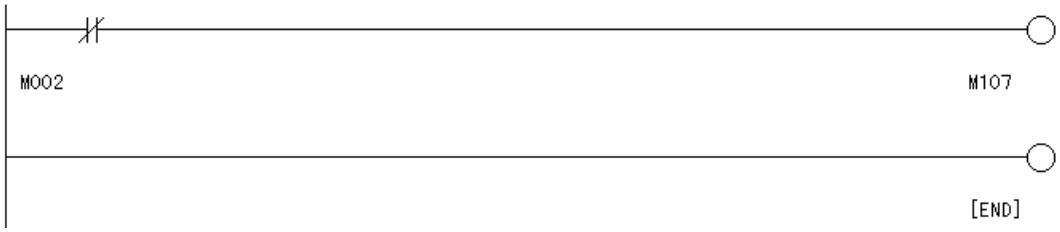


図3．1 入力ポート（2DHのビット2）に入ってきた信号を、反転して出力ポート（2FHのビット7）に出力

（3）手順3：変換条件を設定、変換実行する

次に、作成したラダー図を、どの言語に変換するかを決めます。

変換条件ウィンドウで、変換言語にZ80アセンブリ言語を選択し、また使用するアセンブラにあわせ、出力コードの細かい仕様を指定します。（図3．2）

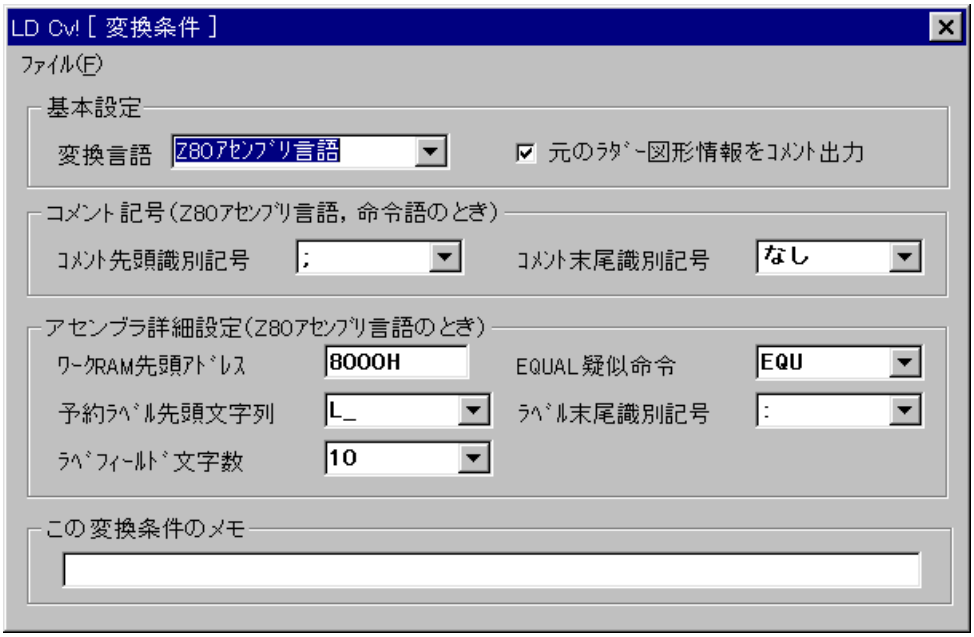


図3．2 変換条件ウィンドウ

Z80アセンブリ言語の場合、ターゲットマシンのハードウェアにあわせ、RAMのうち4KバイトをLD Cv！のコードが使用するので、この情報を、LD Cv！に指定します。ここでは、8000H番地～8FFF 番地を使用することにしました。表3．2に、変換条件ウィンドウ上の設定項目を示します。

表3．2 変換条件ウィンドウ上の設定項目

設定項目	機能
変換言語	ラダー図を、どの言語プログラムに変換するかを‘Z80アセンブリ言語’と‘C言語’と‘命令語’から選択します。
元のラダー図情報をコメント出力	この項目にチェックマークを付けた場合、出力ファイルにラダー図（図形情報）を残します。 このラダー図は、文字を使って疑似的に表現された文字列で、アセンブラ、コンパイラとってはコメント行に見えます。ラダー図は、等幅フォントのとき、正しく表示されます。
コメント先頭識別記号	使用するアセンブラの仕様に基づき、コメントの先頭に付ける記号を選択または入力します。記号‘；’と‘COMMENT’と‘／＊’は選択肢に登録済みで、これ以外の記号も設定可能です。
コメント末尾識別記号	使用するアセンブラの仕様に基づき、コメントの末尾に付ける記号を選択または入力します。記号‘＊／’と‘なし’は選択肢に登録済みで、これ以外の記号も設定可能です。
ワークRAM先頭アドレス	LD Cv！が生成するアセンブリ言語プログラムが使用するワークRAMの先頭アドレスを4000HやF000Hなどのように入力します。
EQUAL疑似命令	アセンブラの仕様に基づき、右の文字列を左のラベルに置き換えるアセンブラ疑似命令（EQU疑似命令）を選択または入力します。文字列‘EQU’と‘equ’は選択肢に登録済みで、これ以外の文字列も設定可能です。
予約ラベル先頭文字列	この文字列を指定すると、LD Cv！が生成するアセンブリ言語プログラムのラベルは、必ずこの文字列から始まります。文字列‘L_’と‘なし’は選択肢に登録済みで、これ以外の文字列も設定可能です。
ラベル末尾識別記号	使用するアセンブラの仕様に基づき、ラベルの末尾に付ける記号を選択または入力します。記号‘：’と‘なし’は選択肢に登録済みで、これ以外の記号も設定可能です。
ラベルフィールド文字数	ラベルフィールドの文字数を8～16から選択します。ラベルフィールドの文字数は、次のとおりでなければなりません。 ラベルフィールド文字数＞予約ラベル先頭文字列文字数＋5＋ラベル末尾識別記号文字数 この条件を満たさない場合[ファイル:このウィンドウを閉じる]メニューを実行してもウィンドウを閉じることができません。
この変換条件のメモ	この変換条件のメモです。変換動作には影響を与えません。

実際に変換によって、LD Cv！が生成したコードをプログラム3．1に示します。

まず、■色の部分で、ワークRAMのアドレスをEQU疑似命令で定義しています。ここに現れている8000Hは、変換条件ウィンドウで定義したワークRAM先頭アドレスです。

次に、■色の部分は、L__ENTRYというラベルで始まるスキャンループ開始前の処理があります。このL__ENTRYを実行する前に、Z80の電源投入からZ80システムの初期設定（スタックポインタの設定、周辺デバイス（または内蔵レジスタ）の初期化を行う必要があるため、これらのプログラムは、プログラマが自分で作成する必要があります。

次の、■色の部分は、スキャン実行ループです。このループで、ラダー図の制御動作が繰り返し実行されます。このループの中でサブルーチンL__EXECをコールしていますが、これがラダー図をZ80アセンブリ言語に変換したサブルーチンをコールしている部分です。これらの制御は、■色の部分で定義したRAMに対しておこなわれるため、このままでは、外部の入出力ポートを無関係です。このため内部のRAMと外部入出力ポートとの間でデータ転送する必要があります。これが、入出力リフレッシュサブルーチンで、スキャン実行ループの中で、サブルーチンコールしています。これらの入出力リフレッシュサブルーチンも、プログラマが自分で作成する必要があります。

その次の、■色の部分は、ラダー図をZ80アセンブリ言語に変換されたサブルーチンです。このサブルーチンは、先ほどのスキャン実行ループの中で、サブルーチンコールされています。

プログラム3．1 ラダー図をZ80アセンブリ言語に変換した結果（LD CV！の出力結果）

```
;(LD Cv! Version 1.5 出力結果 - Z80 ｱｾﾝﾌﾞﾘ言語)
;*****************************************************************
; ﾜｰｸ RAM ｱﾄﾞﾚｽ定義
;*****************************************************************
L_RAM       EQU 8000H ;ﾜｰｸ RAM ｴﾘｱ(4K ﾊﾞｲﾄ)の先頭ｱﾄﾞﾚｽ
L_M0        EQU 8000H ;M ｴﾘｱ(M0-M7FF)
L_S0        EQU 8000H+0800H ;S ｴﾘｱ(S0-S07F)
```

```
L_TO        EQU 8000H+0900H ;T ｴﾘｱ(T0-T07F)
L_CO        EQU 8000H+0B00H ;C ｴﾘｱ(C0-C07F)
L_WK        EQU 8000H+0D00H ;作業ｴﾘｱおよび未使用ｴﾘｱ
;*****************************************************************
; ﾒｲﾝﾌﾟﾛｸﾞﾗﾑ
;*****************************************************************
;*** 電源投入直後のｲﾆｼｬﾗｲｽﾞ後、このﾌﾟﾛｸﾞﾗﾑに飛び込むと運転を開始する ***
L_ENTRY:
            LD A, 0             ;*** ﾜｰｸ RAM ｴﾘｱをｾﾞﾛｸﾘｱ ***
            LD (L_RAM),A
            LD BC,0FFFH
            LD HL,L_RAM
            LD DE,L_RAM+1
            LDIR
            LD HL,L_SO+1EH      ;*** S1E(運転 1 ｽｷｬﾝ ON)の処理 ***
            SET 0,(HL)
            LD HL,L_SO+1FH      ;*** S1F(常時 ON)の処理 ***
            SET 0,(HL)
;*** ﾗﾀﾞｰ図実行ﾙｰﾌﾟ(無限ﾙｰﾌﾟ) ***
L_LOOP:
            CALL L_INPREF       ;*** 入力ﾘﾌﾚｯｼｭ(外部ｻﾌﾞﾙｰﾁﾝ) ***
            LD A,(L_WK)         ;*** SO(ﾀｲﾏ基準ｸﾛｯｸ)の処理 ***
            LD (L_WK+1),A
            LD A,(L_SO)
            LD (L_WK),A
            RES 0,A             ;*** 導通状態ﾚｼﾞｽﾀを OFF ***
            SET 0,B             ;*** 母線状態ﾚｼﾞｽﾀを ON ***
            LD C,0              ;*** 導通状態ｽﾀｯｸをｾﾞﾛｸﾘｱ ***
            CALL L_EXEC         ;*** ﾗﾀﾞｰ図 1 ｽｷｬﾝ実行 ***
            CALL L_OUTREF       ;*** 出力ﾘﾌﾚｯｼｭ(外部ｻﾌﾞﾙｰﾁﾝ) ***
            JR L_LOOP           ;*** ﾗﾀﾞｰ図実行ﾙｰﾌﾟ先頭ﾍｼﾞｬﾝﾌﾟ ***
;*****************************************************************
; ﾗﾀﾞｰ図 1 ｽｷｬﾝ実行ｻﾌﾞﾙｰﾁﾝ
;*****************************************************************
L_EXEC:
;       |
;     0+---|/|-------------------------------------------------------------------------------------( )
;       |M002                                                                                 M107
;       |
;[LDNOT M002]
            RRA
            RL C
            LD A,(L_MO+002H)
            CPL
;[OUT M107]
            AND B
            LD (L_MO+107H),A
;       |
;     1+--------------------------------------------------------------------------------------( )
;       |                                                                                   [END]
;       |
;[END]
            CALL L_END
            RET
```

（以下省略）

(4)手順4:ハードウェアに合わせたコードを作成

LD Cv！が生成したプログラム以外に最低限必要なプログラムは表3．3になります。これらは、プログラマが作る必要があります。電源投入直後の初期設定プログラムは、LD Cv！でなくても、一般にマイクロコンピュータに必要なプログラムです。

表3．3 LD Cv！が生成したプログラム以外に最低限必要なプログラム

プログラム	仕様
電源投入直後の 初期設定プログラム	Z80CPUが、電源投入によりスタートしてから、スタックポインタを設定し、周辺LSI（または内蔵レジスタ）を初期設定した後、指定ラベルへジャンプするまでのプログラム。ラベル名は、予約ラベル先頭文字列＋文字列ENTRY、に決められている。図3．2の変換条件なら、ラベル名は、L__ENTRYとなる。
入力リフレッシュ サブルーチン	I／O番号割付で決めたターゲットマシンの入力ポートのデータを読出し、そのデータを、対応するMの番号のワークRAMに書込むサブルーチン。Mの番号とターゲットマシンのワークRAMの対応は、図3．3による。サブルーチン名は、予約ラベル先頭文字列＋文字列INREF、に決められている。図3．2の変換条件なら、サブルーチン名は、L__INREFとなる。 タイマの基準クロックをS0に与える場合は、この入力リフレッシュサブルーチンの中で、一定時間ごとに定期的にビット反転するデータを、S0に対応するワークRAMに書込むと良い。ただし、ラダー図でタイマの基準クロックを生成する場合は、このサブルーチン内でS0を更新する必要は無い。
出力リフレッシュ サブルーチン	I／O番号割付で決めたターゲットマシンの出力ポートに対応するMの番号のワークRAMからデータを読出し、そのデータを、ターゲットマシンの出力ポートに書込むサブルーチン。Mの番号とターゲットマシンのワークRAMの対応は、図3．3による。サブルーチン名は、予約ラベル先頭文字列＋文字列INREF、に決められている。図2．2の変換条件なら、サブルーチン名は、L__OUTREFとなる。

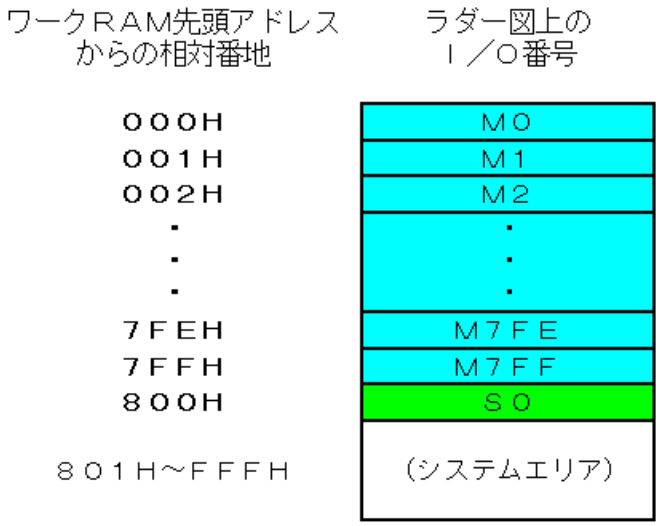


図3．3 ラダー図上のI／O番号と、ターゲットマシンのワークRAMとの対応

（各RAMアドレスの最下位ビットがI／O番号に対応）

以上のプログラムを、アセンブル・リンク・ROM化します。

■C言語に変換した場合

LD Cv！の変換条件ウィンドウ上で、変換言語に「C言語」を選択して変換実行をおこなった場合、ラダー図はC言語で記述された関数plcに変換されます。C言語に変換した場合は、そのCPUのCコンパイラがあれば、ターゲットマシンがZ80である必要はありません。

この関数plcには、ラダー図で書かれた制御ロジック以外に、M，S，T，Cの変数エリアの確保、この関数plcに与えるコマンドの処理がすべて含まれます。

プログラマは、この関数に、コマンドを与えて、I／Oのリード・ライト、ラダー図の実行（コマンド1回実行につき1スキャン）をおこなわせます。Z80アセンブリ言語に変換した場合と異なり、C言語の場合、スキャン実行ループのプログラムコードを生成しません。したがって、プログラマは、スキャン実行ループのプログラムを自分で作成する必要があります。

表3．4に、関数plcの各コマンド機能を示します。

表2．4 関数plcのコマンド機能

Writeコマンド	Readコマンド
■書式　plc('W', prm1, prm2) ■パラメータ prm1 = MのI／O番号(0x000～0x7FF) prm2 = 書込値(0または1) ■戻り値　0 ■動作 仮想PLCのMエリアのパラメータで指定したI／O番号(0～7FF)の 1 点に、パラメータで指定したデータを書き込みます。 ■使用例 M0に1を書き込み、M7Fに0を書込みます。 plc('W',0x0,1) plc('W',0x7f,0)	■書式　plc('R', prm1, 0) ■パラメータ prm1 = MのI／O番号 (0x000～0x7FF) ■戻り値　読出値(0または1) ■動作 仮想PLCのMエリアのパラメータで指定したI／O番号(0～7FF)の 1 点から、データを読み出し、その値(0または1)を返します。 ■使用例 M100とM1FFから値を読出し、a, bに代入します。 a = ldcv('R',0x100,0) ; b = ldcv('R',0x1ff,0) ;
Scanコマンド	**Initコマンド**
■書式　plc('S', 0, 0) ■戻り値　0 ■動作 ラダー図の論理動作を、1スキャン実行します。 ■使用例 ラダー図の論理動作を、1スキャン実行します。 ldcv('S',0,0)	■書式　plc('I', 0, 0) ■戻り値　0 ■動作 仮想PLC内部のメモリを初期化します。 このコマンドは、電源投入からScanコマンドを実行するまでの間に実行してください。 ■使用例 仮想PLC内部のメモリを初期化します。 ldcv('I',0,0)

以下、C言語に変換した場合の開発手順を示します。

(1)手順1:入出力ポートとラダー図上のMとの対応を決める

この手順は、Z80アセンブリ言語に変換した場合と同じです。

(2)手順2:目的とする制御をラダー図で記述

この手順も、Z80アセンブリ言語に変換した場合と同じです。

(3)手順3:変換条件を設定、変換実行する

作成したラダー図をC言語に変換するため、変換条件設定ウィンドウで、C言語を選択します。

実際に変換によって、LD Cv！が生成したコードをプログラム3．2に示します。

まず、■色の部分で、関数名、引数名と、それぞれの型を宣言しています。

次に、■色の部分で、変数を、staticで宣言しています。これは、スキャンのために関数plcをコールした場合、前回のスキャン結果を使う必要があるため、意図的にstatic変数を使っています。

■色の部分は、swich文で、4つのコマンドの解析をおこなっています。

■色の部分は、S(スキャン)コマンドの処理内容で、ラダー図の制御ロジックがC言語に変換されています。

プログラム3．2 ラダー図をC言語に変換した結果(LD Cv！の出力結果)

```
/* (LD Cv! Version 1.5 出力結果 - C 言語) */

unsigned char plc(unsigned char cmd,unsigned int prm1,unsigned int prm2)
{
    static unsigned char m[0x800],s[0x80],t[0x80],c[0x80];
    static unsigned char rg,mc,stk;
    static unsigned int tk[0x80],ck[0x80];
    static unsigned char cf[0x80],cl[0x80];
    static unsigned char s0_old,s0_new;
    static unsigned int cnt_10,cnt_100,cnt_1000,cnt_10000;
```

```
    unsigned int i;

    switch(cmd){
    case 'R':
        return(m[prm1] & 0x01);
    case 'W':
        m[prm1] = prm2 & 0x01;
        return(0);
    case 'I':
        for (i = 0; i < 0x800; i++){m[i] = 0;};
        for (i = 0; i <  0x80; i++){s[i] = 0;};
        for (i = 0; i <  0x80; i++){t[i] = 0; tk[i] = 0;};
        for (i = 0; i <  0x80; i++){c[i] = 0; ck[i] = 0; cf[i] = 0; cl[i] = 0;};
        s[0x1e] = 0x01; s[0x1f] = 0x01;
        cnt_10 = 0; cnt_100 = 0; cnt_1000 = 0; cnt_10000 = 0;
        return(0);
    case 'S':
        s0_old = s0_new; s0_new = s[0];
        rg = 0x00; mc = 0x01; stk = 0x00;

        /* User Program */

/*     |                                                                              */
/*    0+----|/|-------------------------------------------------------------------()  */
/*     |M002                                                              M107        */
/*     |                                                                              */

        /* LDNOT M002 */

        stk = stk << 1;
        stk = stk | (rg & 0x01);
        rg = ~m[0x002];


        /* OUT M107 */

        m[0x107] = rg & mc & 0x01;

/*     |                                                                              */
/*    1+-------------------------------------------------------------------------()  */
/*     |                                                                  [END]       */
/*     |                                                                              */
```

```
        /* END */

        s[0x10] = ~s[0x10] & 0x01;
        cnt_10++    ; cnt_10    = cnt_10 % 10;
        cnt_100++   ; cnt_100   = cnt_100 % 100;
        cnt_1000++  ; cnt_1000  = cnt_1000 % 1000;
        cnt_10000++ ; cnt_10000 = cnt_10000 % 10000;
        if (cnt_10    == 0){s[0x11] = ~s[0x11] & 0x01;};
        if (cnt_100   == 0){s[0x12] = ~s[0x12] & 0x01;};
        if (cnt_1000  == 0){s[0x13] = ~s[0x13] & 0x01;};
        if (cnt_10000 == 0){s[0x14] = ~s[0x14] & 0x01;};
        s[0x1e] = 0x00;
        s[0x1f] = 0x01;

        return(0);
    }
}
```

（4）手順4：スキャン実行ループのコードと、ハードウェアに合わせたコードを作成

C言語に変換した場合、スキャン実行ループを含んだメインルーチンは生成しません。したがって、メインルーチンは、プログラマが作成する必要があります。このメインルーチンは、図3．4のフローチャートのように作成します。

このフローチャートに記載してあるInit、Write、Scan、Readコマンドは、関数plcに引数を与えて実行することで、コマンドが実行されます。

なお、フローチャートを見てわかるとおり、電源投入直後の初期設定プログラムをプログラマが作成する必要があります。また、入力リフレッシュ処理で入力ポートからの読出をおこなう関数が、出力リフレッシュ処理で出力ポートへの書込をおこなう関数が必要となります。これらの関数は、コンパイラに付属している関数を使うか、無ければアセンブラでプログラマが作成する必要があります。

LD Cv！が生成した関数plcと、このフローチャートのプログラムを、コンパイル・リンク・ROM化します。

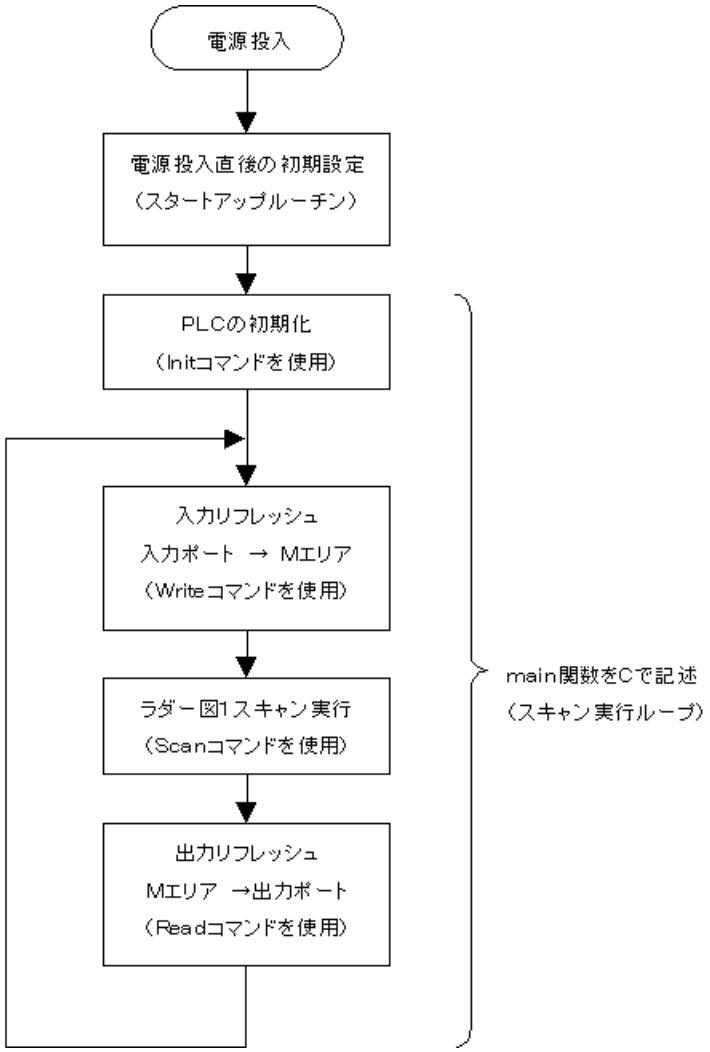


図3．4 C言語に変換した場合のメインルーチン(プログラマが作成する)

－ 以上 －