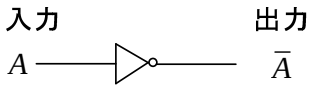
第 7 章　マイコン制御システム

はじめに論理回路と 2 進数のかんたんな話をしよう。

○ 論理回路

基本となる論理回路である NOT 回路，AND 回路, NAND 回路，OR 回路，NOR 回路について真理値表，ベン図を以下に示す。記号は MIL 規格(military standard)を用いている。ここでは高電位 H を 1，低電位 L を 0 に対応させた **H 駆動（正論理）**(Active High)を考える。高電位 H を 0，低電位 L を 1 に対応させた **L 駆動(負論理)**(Active Low)は後で述べる。

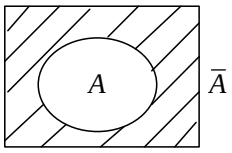
(1)　NOT 回路



論理回路

A	$\overline{A}$
0	1
1	0

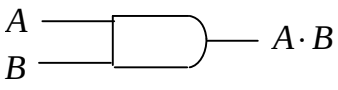
真理値表



ベン図

NOT とは否定の意味である。論理回路で A が入力，$\overline{A}$ が出力である。$\overline{A}$ は A の NOT を表わしている。真理値表は，入力と出力の関係を示す。変数には 0, 1 以外の値はない。**ベン図**はもともと集合についての関係を表わすものとして考案されたが，論理演算を視覚的に分かりやすく表現するものとしてしばしば用いられている。円の中が $A=1$，円の外が $A=0$ を意味する。斜線部は出力が 1 となる領域を表す。ベン図の円の中の A が 0 か 1 と考えてはいけない。NOT 回路を２つつなぐと元に戻る。

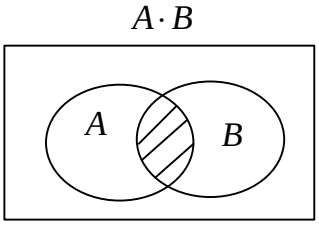
(2)　AND 回路



論理回路

A	B	$A\cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

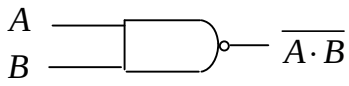
真理値表



ベン図

論理積回路とも言われる。出力は論理式で $A\cdot B$ と書き，掛け算の結果と一致するので分かりやすい。ベン図の斜線領域は出力 $A\cdot B$ が 1 となる領域を示す。

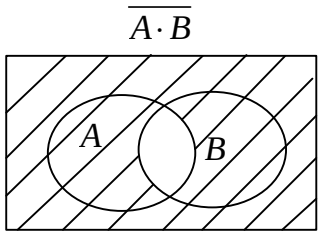
(3) NAND 回路



論理回路

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

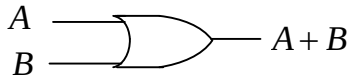
真理値表



ベン図

NAND は AND と NOT を組み合わせたものである。○印が NOT に相当する。$\overline{A \cdot B}$ は $A \cdot B$ の NOT である。NAND に NOT をつなぐと AND になる。

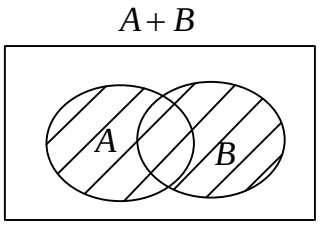
(4) OR 回路



論理回路

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

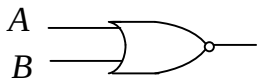
真理値表



ベン図

OR の出力は論理式で $A+B$ と表現される。どちらかの入力が 1 なら出力も 1 になる。

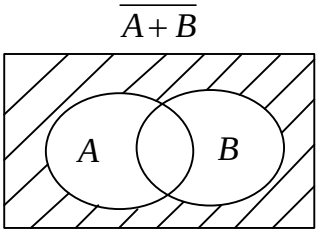
(5) NOR 回路



論理回路

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

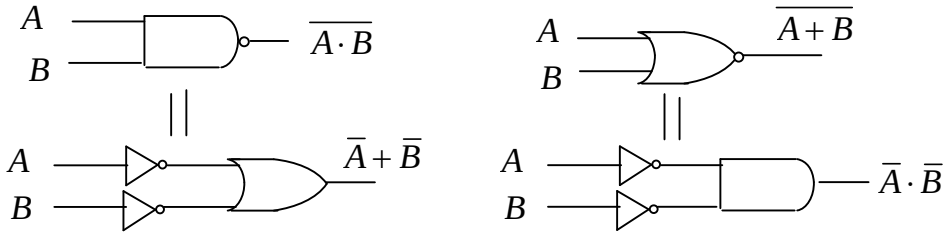
真理値表



ベン図

ド・モルガンの定理より成り立つ等価な論理回路を示す。形式的には反転の○を入力側に移すと AND は OR，OR は AND に変化する。

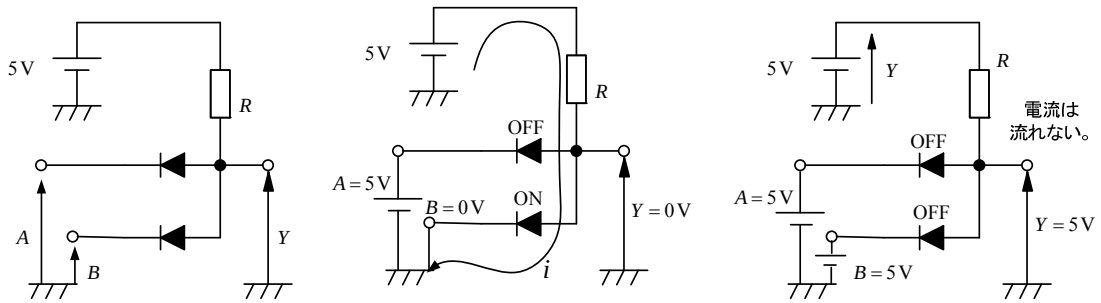
ド・モルガンの定理 ： $\overline{A \cdot B} = \overline{A} + \overline{B},\ \overline{A+B} = \overline{A} \cdot \overline{B}$



等価な論理回路（ド・モルガンの定理）

H 駆動(Active High)と **L 駆動**(Active low)について，簡単に述べておこう。ディジタル信号で，何かの動作をしようとするとき，その出力ピンの電圧が 5V で目的を果たす場合と 0V で目的を果たす場合がある。前者が H 駆動で後者が L 駆動である。例えば，5V 出力信号で発光ダイオードをつけるか，0V 信号で発光ダイオードをつけるかの違いである。

図の回路で具体的に考える。電圧 *A, B* が入力で，電圧 *Y* が出力である。図の様に *A* = 5V, *B* = 0V とすると，ダイオードを通って電流が流れ，*Y* = 0V となる(ダイオードの電圧を無視する)。*A* = 5V, *B* = 5V とすると，電流は流れず，抵抗の電圧は 0 だから *Y* = 5V となる。従って，H = 5V, L = 0V で書いた真理値表は図のようになる。これは，実際の電圧の表である。

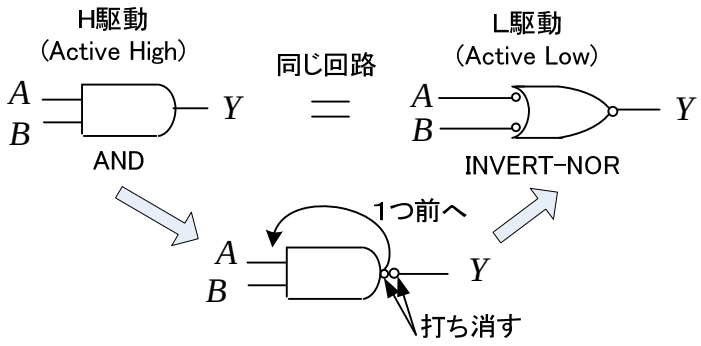


H 駆動時 AND 回路　　A=H, B=L の場合　　A=H, B=H の場合

A	*B*	*Y*
L	L	L
H	L	L
L	H	L
H	H	H

真理値表

L = 0V
H =5V



H 駆動(Active High)と L 駆動(Active low)の論理回路の書き方

（どちらで書いても間違いではないが目的で分けた方が設計者の意図が明確）

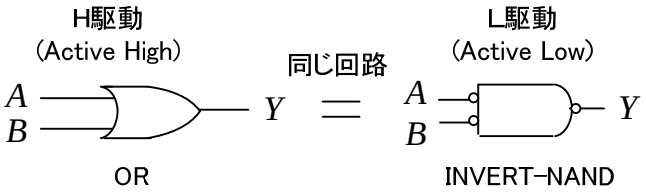
H 駆動とは Y = H のとき次段の素子を動作させるので，入力 A,B のどちらも H にならないと出力は出ないという見方である。よってこの回路は AND と考えられる。一方，**L 駆動**とは，出力が L になったら次段の素子を動作させるので，入力のどちらかが L なら動作するので，L に注目すると真理値表よりこの回路は OR と考えられる。そこで，設計者がどちらを考えているか記号を分けた方が判りやすい。AND の出力に反転の○を２つつけても等価で，そのうちの 1 つを入力側に移すとド・モルガンの定理より AND が OR の記号となり，INVERT-NOR の記号が得られる。両者の回路は全く同じで，論理記号にも矛盾はない。しかし書き方を変えることで，H 駆動と L 駆動のどちらで考えているかが判りやすく，L 駆動の場合 OR のイメージが得られる。

同様に H 駆動 OR の場合を考える。真理値表の L に着目した L 駆動の場合には，入力 A, B のどちらも L にならないと出力は L にならず駆動しないので AND と考えられる。よって L 駆動の場合には図の INVERT-NAND で書いた方が判りやすい。この L 駆動の論理回路は後述のマイコンの接続で利用する。

A	B	Y
L	L	L
H	L	H
L	H	H
H	H	H

真理値表

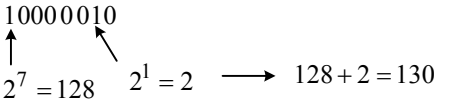
L = 0V
H =5V



○ 2 進数, 16 進数

通常我々は 10 進数を用いているが，コンピュータでは 2 進数や 16 進数が用いられている。10 進数の場合には，0 から 9 の数字を使うが，2 進数の場合には 0 と 1 しか使えない。16 進数では，0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F の 16 種の記号を使う。A は 10 進数の 10 に相当し，順に対応して F が 15 になる。10 進数の足し算で 10 以上になると桁上がりとなるが，2 進数では 2 以上，16 進数では 16 以上になると桁上がりする。2 進数と 16 進数は，よく対応しており，2 進数の 4 桁をまとめて 16 進数の 1 つの数値になる。

2 進数を 10 進数に直す場合は，各桁(けた)の重みを 2^n （2 桁目が $n=1$）として 2 進数の数値に掛けて加えるとよい。10 進数でも各桁は 10^n の重みを持っている。



10 進数を 2 進数に直すには， 2 で割り算を行って，余りを右側に書いて下から順に並べると良い。12（10 進数）＝1100(2 進数)

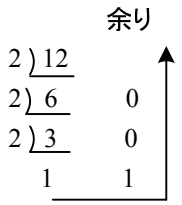


表 7-1 8ビットデータの表現

2 進数 （Binary）	16 進数 (**H**exa Decimal)	符号無し表現とみた場合の 10 進数の値 (Decimal)	符号付き表現とみた場合の 10 進数の値 (Decimal)
0000 0000	00	0	0
0000 0001	01	1	1
0000 0010	02	2	2
・	・	・	・
・	・	・	・
・	・	・	・
0111 1111	7F	127	127
1000 0000	80	128	-128
1000 0001	81	129	-127
1000 0010	82	130	-126
・	・	・	・
・	・	・	・
・	・	・	・
1111 1110	FE	254	-2
1111 1111	FF	255	-1

16 進数の場合，数値の後に H を付ける。例 10H（10 進数では 16）

2 進数の 1 桁を 1 **ビット**(bit)という。表は 8 桁なので，8 ビットである。8 ビットで表現できる数値は 0 から 255 の 256（ $2^8 = 256$ ）種類である。なお 8 ビットで負の数を表現しようとすると，表のように最上位ビットが 1 の場合を割り当てる。2 の補数表現と呼ばれるが，深入りはしないでおく。

○ 量子化

コンピュータで信号を処理するためには，A/D 変換器によって，センサから得られるアナログ量をディジタル量に変換する必要がある。A/D 変換器によるデータの取り込みに関しては，**量子化**の問題がある。

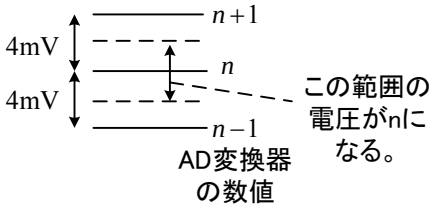
A/D 変換器は 8bit や 12bit などの限られた数値に変換される。例えば，8 ビットの A/D 変換器なら，センサで検出するデータを 0 から 255（または-128 から 127）の 256 種類に割り当てる必要がある。つまりある範囲をまとめて 1 つの数値に直さないといけない。これを量子化という。次の問題を考えよう。

問　A/D 変換器によって 0～1V 信号電圧を 8 ビットのディジタル信号に量子化する。量子化誤差の最大値に最も近いのはどれか。

(1) ±0.2mV　(2) ±2mV　(3) ±20mV　(4) ±200mV　(5) ±2V

(解) 8 ビットの分解能は $2^8 = 256$ (0 から 255 まで), $1/256 = 0.0039$ より，約 4mV である。つまり，4mV の幅でどれかの数値に割り当てる必要がある。

量子化するときに四捨五入するなら図の範囲の電圧が同じ数値 n (=0～255)になる。よって図より正解は(2)



A/D 変換の量子化

このように，ディジタル信号はアナログ信号より精度が劣化するが，一旦ディジタル量になると雑音の影響はほとんどなくなる。

○ コンピュータのソフトウェア

ソフトウェアは**基本ソフト**と**応用ソフト（アプリケーションソフト）**に分類される。基本ソフトには，**オペレーティングシステム** OS，**プログラミング言語**，ユーティリティープログラム，オブジェクトプログラムなどがある。応用ソフトは，ワープロソフト，エディタなど沢山ある。

オペレーティングシステム（OS）は, 応用ソフトが動くための土台と成るソフトで, ハードウェアに直接関係するソフトである。MS-DOS, Linux, UNIX, Android, Windows7 などがある。これらの OS 上でソフトウェアを開発すれば, メモリや入出力の番地は考えなくても良くなり, 開発がし易くなる。

プログラミング言語は，ソフトウェアを開発するためのものである。コンピュータを動かすには, 最終的には 1 と 0 の組み合わせとして, メモリに入れないといけない。しかし，これは人間にとって大変わかりにくい。そこで人間がわかり易い言葉すなわち言語が作られた（これを**高級プログラミング言語**という）。1 と 0 の組み合わせはコンピュータにとって理解できる唯一の言語で，これを**機械語**という。汎用言語の C, C++, 事務処理用言語 COBOL, 科学技術計算用言語 FORTRAN などは高級プログラミング言語である。これらの言語で書かれたプログラムは, 実行する前に**コンパイラ**（翻訳するプログラム）によって**コンパイル**して機械語に直す必要がある。BASIC は高級プログラミング言語であるが, 翻訳と実行を**インタプリタ**で行う。高級プログラミング言語に対して，**低級プログラミング言語**には先ほど述べた機械語の他に**アセンブリ言語**がある。ニーモニックコードで書くことで機械語に比べると人間に判りやすいが, CPU によって命令が異なりプログラム開発は大変である。アセンブリ言語を機械語に変換するソフトを**アセンブラ**という。

7.1　CPU

マイコンは**マイクロコンピュータ**の略称で，**CPU**(central processing unit)，メモリ，入出力装置から成る。本節では**8ビット**の CPU である **Z80**（**ぜっとはちまる**）について述べる。Z80 は一世を風靡（ふうび）した CPU であるが現在でも使われており，簡単で理解が容易なことからマイコンの原理を知るためには適している。

図 7-1 に Z80 の構成を示す（文献(4)）。これは他の **LSI**(Large Scale Integration：大規模集積回路)についても言えることであるが，Z80 は 5V 単一電源で動作し，**TTL**(transistor-transistor logic)コンパチブルである。図でⓉと記入されている端子は，“H”レベル(5V)と“L”レベル(0V)の他に線が切れた“高インピーダンス”状態を持つもので**トライステート**と呼ばれる。ピンの矢印は信号の伝わる向きを示す。制御線の○印やバー ‾ は**L駆動**（Active Low）を表すもので，“L”レベルのときにそのピンのもつ働きが有効となる。例えば，$\overline{RD}$ は読み込み動作をしようとするとき“L”レベルとなり，$\overline{INT}$ は“L”レベルになると割り込みを CPU に要求する。図で A，B，C，D，E，H，L，I の各**レジスタ**は 8 ビット， IX，IY，SP，PC の各レジスタは 16 ビットである。レジスタは CPU 内のメモリと思えばよい。表 7-2 に Z80 のピンの主な機能を示す。

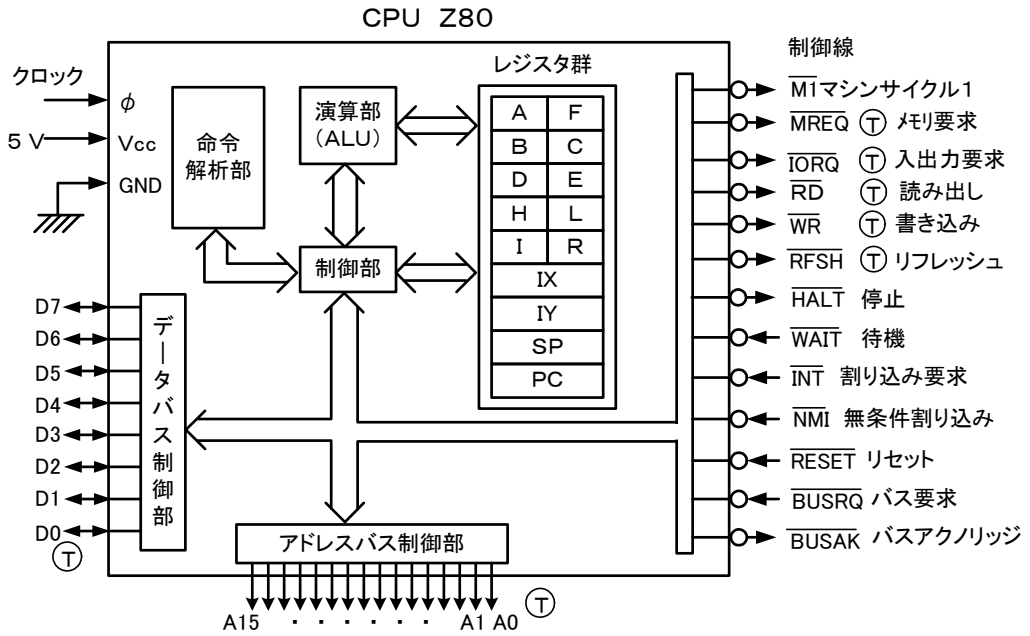


図 7-1　Z80 の構成

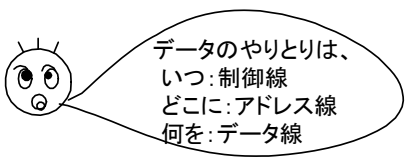


表 7-2　Z80 のピンの主な機能

ピ　ン	名　　前	機　　能
ϕ	Clock	このクロックパルスを基準に CPU が動作する。 Z80 は 2.5MHz 以下，Z80A は 4MHz 以下
A0～A15	Address bus	メモリや**入出力装置**（IO）の番地指定に用いる線である。
D0～D7	Data bus	メモリや IO と命令やデータをやりとりする線である。
$\overline{\text{MREQ}}$ $\overline{\text{IORQ}}$	Memory request IO request	CPU がメモリ(LD 命令)または IO (IN, OUT 命令)のどちらとつながるかを決める。
$\overline{\text{RD}}$ $\overline{\text{WR}}$	Read Write	CPU がメモリや IO からデータを読み込むのか，逆に書き込むのかを決める。
$\overline{\text{INT}}$	Interrupt request	CPU に割り込みを要求する。
$\overline{\text{RESET}}$	Reset	CPU を初期状態にする。

7.2　CPU とメモリ

(a)CPU とメモリの接続

半導体メモリは, CPU から直接読み出しと書き込みの両方が可能な **RAM**（Random Access Memory）と読み出し専用の **ROM**（Read Only Memory）に大別できる。RAM は電源を切るとデータが消えてしまうのが普通で，電源を入れたままでもデータが消える**ダイナミック RAM** もある。ダイナミック RAM でない RAM を**スタティック RAM** という。ROM には，素子の製造中にデータを固定してしまい，後で変更できないマスク ROM，紫外線を照射することにより何度もデータの書き込みが可能な EPROM などがある。ここでは仕組みを理解する目的でメモリ容量 8K バイトの EPROM と 8K バイトのスタティック RAM を用いる（容量は小さいが原理を理解することが目的である）。

図 7-2 に Z80 とメモリの接続例を示す。**アドレス線** A0～A15（16 本）で指定できる番地の数は，$2^{16}=2^6\times2^{10}=64\times1024=65536=64K$ である。コンピュータの分野では K（キロ）が 1024 を意味することがある。メモリの各番地は 8 ビット構成であり，**1バイト＝8ビット**であるから，図の 8K バイト(64K ビット)メモリの番地の数は 8K である。8K の番地を指定するアドレス線の数は，$2^{13}=2^3\times2^{10}=8K$ であるから A0～A12 までの 13 本で良く，残り A13～A15 は**アドレスレコーダ** 74LS138 に接続してメモリ IC の選択に用いる（メモリを何個もつなぐ場合で，最近はこのようなケースは少ないだろう）。74LS138 を図のように接続すると，表 7-3 に示すように，A13～A15 に対し，Y0～Y7 のいずれかの端子が“L”となる（他は“H”）。

ROM は，$\overline{\text{CE}}$ **(chip enable)**が“L”，$\overline{\text{OE}}$ **(output enable)**が“L”ならばアドレス信号（A0～A12）の番地のデータを，データバス（D0～D7）に出力する。CPU はそれを読み込むことになる。**RAM** は，$\overline{\text{CE}}$ が“L”，$\overline{\text{WR}}$ (RAM)が“L”ならば，$\overline{\text{OE}}$ の状態に無関係に，アドレス信号の示す番地に，データバスの内容を記憶する。データの読み出しは，$\overline{\text{CE}}$ が“L”，$\overline{\text{OE}}$ が“L”，$\overline{\text{WR}}$ (RAM)を“H”とする。アドレスデコーダの出力と $\overline{\text{CE}}$ の接続から，図の回路の**メモリマップ**は表 7-4 のようになる。A0～A12 までの 13 本(13 ビット)あることに注意せよ。

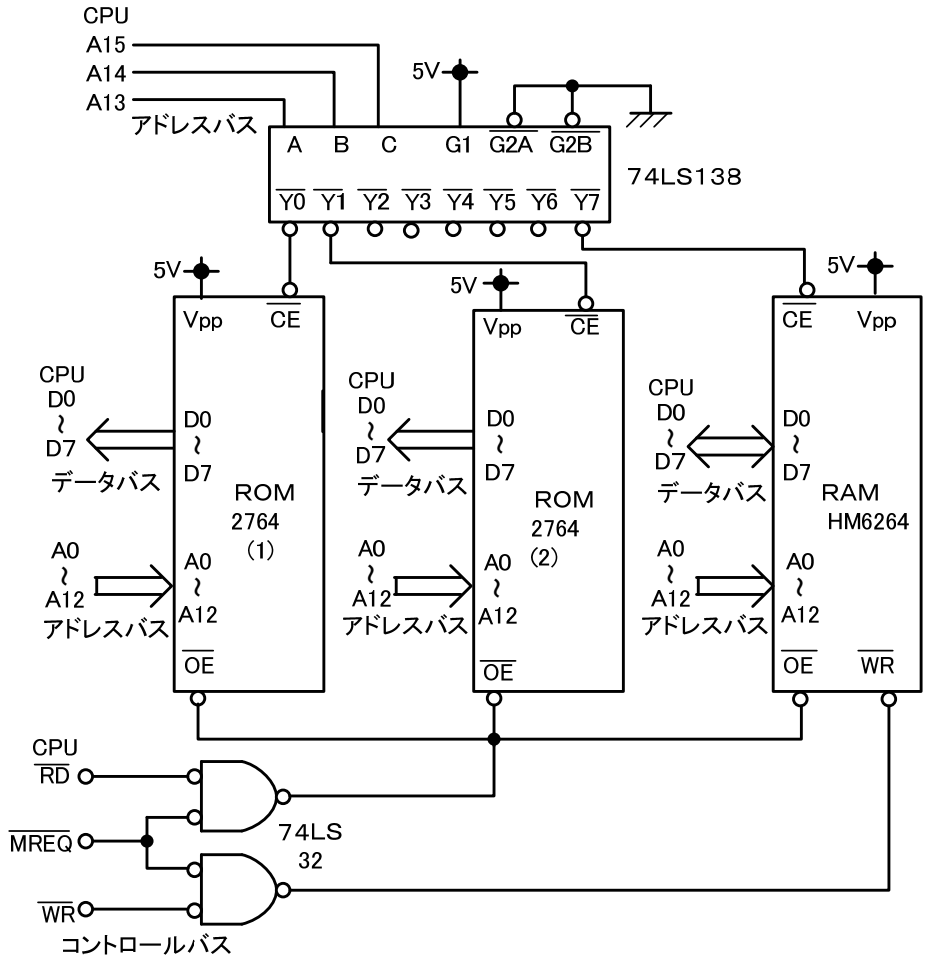


図 7-2　Z80 とメモリの接続

表 7-3　74LS138 の入出力表

A15	A14	A13	“L”となるピン
0	0	0	$\overline{Y0}$
0	0	1	$\overline{Y1}$
0	1	0	$\overline{Y2}$
0	1	1	$\overline{Y3}$
1	0	0	$\overline{Y4}$
1	0	1	$\overline{Y5}$
1	1	0	$\overline{Y6}$
1	1	1	$\overline{Y7}$

G1＝“H”,$\overline{G2A}$＝$\overline{G2B}$＝“L”のとき

表 7-4　メモリマップ

番　地	メ　モ　リ
０００００Ｈ ： １ＦＦＦＨ	ROM(1)
２０００Ｈ ： ３ＦＦＦＨ	ROM(2)
：	空　き
Ｅ０００Ｈ ： ＦＦＦＦＨ	RAM

(b)プログラムの実行

CPU の基本的な動作は，メモリに保存されている命令を次々に読み出して実行することにある。ここでは簡単なプログラムを CPU が実行してゆく過程を説明する。図 7-3 にプログラムの実行を説明するためのハードウェアの構成とメモリの内容の一例を示す。CPU とメモリの接続は図 7-2 と同じものとする。

1．リセットスイッチを押して，$\overline{\text{RESET}}$ が“L”になると，CPU は CPU 内の**プログラムカウンタ(PC)**を **0** にする。

2．リセットスイッチを切って，$\overline{\text{RESET}}$ が“H”になると，PC が示す 0 番地の命令を**フェッチ**（fetch：行って取って来る）し，解析する。（**フェッチサイクル**＝4 クロック）

① CPU は PC の値をアドレスバスに出力する。A0＝A1＝・・・＝A15＝0 なので，ROM(1)の 0 番地が指定される。その後 **CPU は PC に1を加え，PC＝1とする。**

② CPU の $\overline{\text{MREQ}}$，$\overline{\text{RD}}$ が共に“L”となる。その結果，ROM(1)の $\overline{\text{OE}}$ が“L”となり（L 駆動の考え方が役立つ），ROM はデータバス上に 0 番地の値 3AH を出力する。

③ CPU はデータバス上の 3AH を読み込み，この命令を解析する。この結果，CPU は次の番地（1 番地）のデータを下位番地，更に次の番地（2 番地）のデータを上位番地としたメモリからデーを読み込み，A レジスタに入れる命令であることを知る。（表 7-5 参照）。

3．読み出したいデータが入っている番地をメモリから読み込む。

メモリリードサイクル（3 クロック）×2＝6 クロック

① CPU は PC の値をアドレスバスに出力する。ROM（1）の 1 番地が指定される。その後，**PC＝2**となる。

② ２の②と同様にして，ROM（1）はデータバス上に 1 番地のデータ 00H を出力する。

③ CPU はデータバス上の 00H を読み込む。

④ ①～③と同様にして 2 番地のデータ 20H を読み込む。**PC＝3**となる。

4．CPU は読み出したデータの入っている番地 2000H が分ったので，ROM（2）からこれを読み込む。この結果 10H が，レジスタ A に入る。

5．CPU は PC の値をアドレスバス上に出力し，次の命令のフェッチサイクルが始まる。２と同様にして，CPU はメモリより 32H を読み込み，それを解析して，A レジスタのデータを次の番地（4 番地）のデータを下位番地，更に次の番地（5 番地）のデータを上位番地としてメモリへ書き込む命令であることを知る。**PC＝4**となる。

6．CPU はデータを書き込む番地をメモリから読み出す。３と同様にして，4 番地の 00H，5 番地の E0H を読み込む。**PC＝6**となる。

7．CPU は E000H 番地に A レジスタの値（10H）を書き込む。

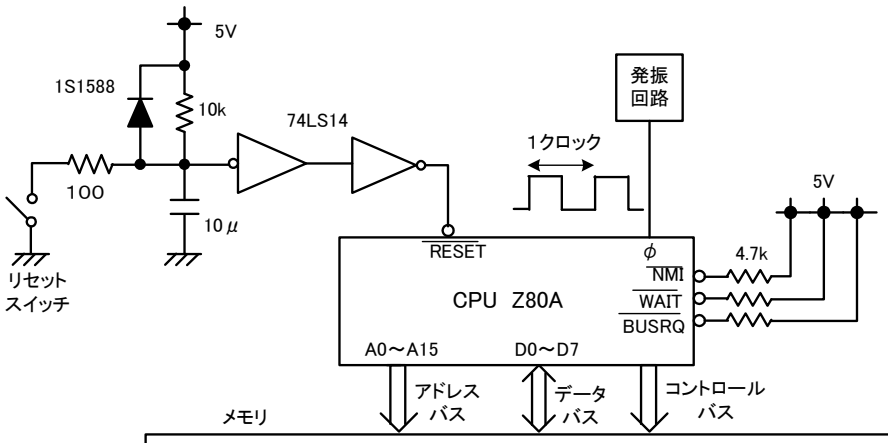
（**メモリライトサイクル**＝3 クロック）

① CPU はアドレスバスに E000H を，データバスに A レジスタの値を出力する。RAM の E000H 番地が選定される。

② CPU の $\overline{\text{MREQ}}$ ， $\overline{\text{WR}}$ が共に“L”となり，その結果 RAM の $\overline{\text{WR}}$ が“L”となる。

③ RAM はデータバスの値を E000H 番地へ保存する。

以下，同様にして次々と命令が実行されることになる。正確にはメーカが公表したタイミング図により各信号の変化が定まっているが，プログラムの基本的な実行過程は理解できたであろう。簡単に言うと，CPU にある **PC の値を自動的に1つずつ増やして，それが示す番地の命令を読み出し実行しているのである。**上述の例でも判るように，命令の実行はいくつかのサイクルから構成されている。それには，①フェッチサイクル　②メモリリードサイクル　③メモリライトサイクル　④IO リードサイクル　⑤IO ライトサイクル　⑥割り込み応答サイクルなどがある。フェッチサイクルはどの命令にも必要である。



番地（１６ビット）	データ（８ビット）	メモリ
０Ｈ	３ＡＨ	
１Ｈ	００Ｈ	
２Ｈ	２０Ｈ	
３Ｈ	３２Ｈ	ＲＯＭ
４Ｈ	００Ｈ	（１）
５Ｈ	Ｅ０Ｈ	
：	：	
１ＦＦＦＨ	：	
２０００Ｈ	１０Ｈ	
２００１Ｈ	ＦＦＨ	ＲＯＭ
：	：	（２）
３ＦＦＦＨ	：	
		空　き
Ｅ０００Ｈ	：	
：	：	ＲＡＭ
ＦＦＦＦＨ	：	

図 7-3　プログラムの実行（データは一例）

7.3　アセンブリ言語

16 進数で表現される**機械語**は人間には理解しにくいので，ニーモニックという機械語に対応した英記号を使って作った**アセンブリ言語**を用いる。**アセンブリ言語**で書かれたプログラムを機械語に自動的に置き換えるプログラムは**アセンブラ**と呼ばれる。表 7-5 に Z80 の命令を示す。

フラグが変化する**加算命令**（ADD）と**減算命令**（SUB）について説明しておく。表 7-1 は 8 ビットデータの表現法とそれを**符号無し表現**とみた場合，または**符号付表現**（2 の補数表現）とみた場合の 10 進数での値を示している。表 7-1 で，8 ビットデータを符号無し表現とみるか，それとも符号付表現とみるかはプログラマー自身が決めることで，加算命令や減算命令で両者が区別して実行される訳ではない。ただし，CPU 内にある **F レジスタの値（フラグ）**が有用な情報を提供してくれる。例えば，正の数だけを対象とした符号無し表現としてデータを扱う場合，加算，減算の結果，桁上り（255 を超えるとき），桁下り（負の数になるとき）があると，**キャリフラグ**（C フラグ）が 1 になり，桁上り，桁下りが無いときは 0 となる。符号付表現としてデータを扱う場合，加算，減算の結果-128～+127 以外の数を表現する必要が生じたとき**オーバフローフラグ**（V フラグ）が 1 となり，範囲内であれば 0 となる。従ってフラグの変化を調べながら演算を行う必要がある。

表 7-5　Z80 の命令

ニーモニック	機械語 （16 進数）	ｸﾛｯｸ ｻｲｸﾙ	機能説明
LD　A, n	3E　n	7	8 ﾋﾞｯﾄ定数 n を A ﾚｼﾞｽﾀに入れる。 A←n
LD　A, B	78	4	B ﾚｼﾞｽﾀの値を A ﾚｼﾞｽﾀへ入れる。 B は不変　　A←B
LD　A, (lm)	3A　m　l	13	lm 番地のﾃﾞｰﾀを A ﾚｼﾞｽﾀへ入れる。 A←(lm)
LD　(lm), A	32　m　l	13	(lm)←A
ADD　A, B	80	4	A←A+B　ﾌﾗｸﾞ変化あり
SUB　B	90	4	A←A-B　ﾌﾗｸﾞ変化あり
IN　A, (n)	DB　n	11	n 番地(8 ﾋﾞｯﾄ)の IO ﾎﾟｰﾄから A ﾚｼﾞｽﾀへ読み込む。
OUT　(n), (A)	D3　n	11	A ﾚｼﾞｽﾀの内容を n 番地の IO ﾎﾟｰﾄへ出力する。
PUSH　BC	C5	11	BC ﾚｼﾞｽﾀの値をｽﾀｯｶへ転送する。 (SP-1)←B(上位) (SP-2)←C(下位) SP←SP-2
POP　BC	C1	10	ｽﾀｯｶの値を BC ﾚｼﾞｽﾀへ転送する。 C(下位)←(SP) B(上位)←(SP+1) SP←SP+2

JP　lm	C3　m　l	10	lm 番地へｼﾞｬﾝﾌﾟする。 PC←lm
CALL lm	CD　m　l	17	ｻﾌﾞﾙｰﾁﾝｺｰﾙ PC をｽﾀｯｶへ PUSH し PC←lm
RET	C9	10	ｻﾌﾞﾙｰﾁﾝからのﾘﾀｰﾝ PC へｽﾀｯｶより POP
EI	FB	4	割り込み許可
DI	F3	4	割り込み禁止

LD 命令では $\overline{\text{MREQ}}$ が“L”，IN,OUT 命令では $\overline{\text{IORQ}}$ が“L”になる。

最近の 32 ビットなどの CPU では，アセンブリ言語のかわりに **C 言語**でプログラムを書くので便利である。また変数が浮動小数点演算できるのでフラグも気にしないですむ。

7.4　マイコンの構成

CPU を何かの目的に使うとき，必ずメモリ以外からのデータの入出力を必要とする。

（a）汎用入出力 LSI　8255

例えば，8 ビットデータにより 8 個の発光ダイオードを点滅する場合を考えてみよう。この 8 ビットのデータはデータバスから出力することになるが，データバスは CPU とメモリがデータをやりとりする場合にも使われるので，データバスに直接発光ダイオードをつなぐとどちらのデータか区別できず誤動作することになる。また，一度出力したデータは次のデータが来るまで保持する（**ラッチ**する）ことが望ましい。更に，発光ダイオードだけではなく他の周辺装置とデータのやりとりをしたい場合もあり，データバスをうまく使い分ける必要がある。このような場合, 3 つの 8 ビット入出力ポートをもつ 8255 が利用できる。図 7-4 に Z80 と 8255 の接続例を示す。図において，デコーダ 74LS138 で 8255 をはじめ後述する各周辺 LSI を選択する。図より, アドレスバスの下位 8 ビットが E0H～E3H のとき $\overline{\text{CS}}$ が“L”となり，8255 が動作する。CPU と 8255 の A, B, C ポート及び **CW（コントロールワード）レジスタ**のいずれとを接続するかは，A0 と A1 の信号で決る。また，データの入出力の区別は 8255 の $\overline{\text{RD}}$，$\overline{\text{WR}}$ 信号で行う。表 7-6 に 8255 の基本機能を示す。

8255 には，モード 0，モード 1，モード 2 の使い方があるが，単なる IO ポートとして使う場合にはモード 0 で使用する。図 7-5 にモード 0 での CW レジスタへの書き込み形式を示す。モード 0 において，出力ポートとして使うときデータは保持されるが，入力ポートとして使うときは保持されない。図 7-4 の回路で，ポート B のスイッチがオンのとき，各ビットに対するポート A の発光ダイオードを点燈するプログラムを示す。

① LD　　A,　10000010B　　; モード 0 で A，C ポート出力。B ポート入力。
② OUT　(0E3H),　A　　; CW レジスタへの書き込み
③ IN　　A,　(0E1H)　　; B ポートからデータの読み込み
④ CPL　　; A レジスタのビット反転（0→1，1→0）
⑤ OUT　(0E0H),　A　　; A ポートへ出力（発光ダイオード点燈）

②の　OUT　（0E3H），A　（機械語　D3　E3）の実行過程を以下に説明する。

１．フェッチサイクル（4 クロック）

CPU はメモリより D3H を読み込み，命令を解析する。

２．メモリリードサイクル（3 クロック）

CPU はメモリより E3H を読み込む。

３．IO ライトサイクル（4 クロック）

① CPU はアドレスバスの下位 8 ビットに E3H を出力する。これにより，8255 が動作する。

② CPU はデータバスに A レジスタの内容を出力する。

③ CPU の $\overline{\text{IORQ}}$，$\overline{\text{WR}}$ が共に“L”となり，その結果 8255 の $\overline{\text{WR}}$ が“L”となり，8255 はデータバスの内容を CW レジスタへ取り込む。

（注意）IN または OUT 命令では，$\overline{\text{IORQ}}$ 端子が“L”になるので，E3H,E0H 番地は同じ番地がメモリにあっても構わない。メモリとは LD 命令を使い，$\overline{\text{MREQ}}$ が“L”になる。

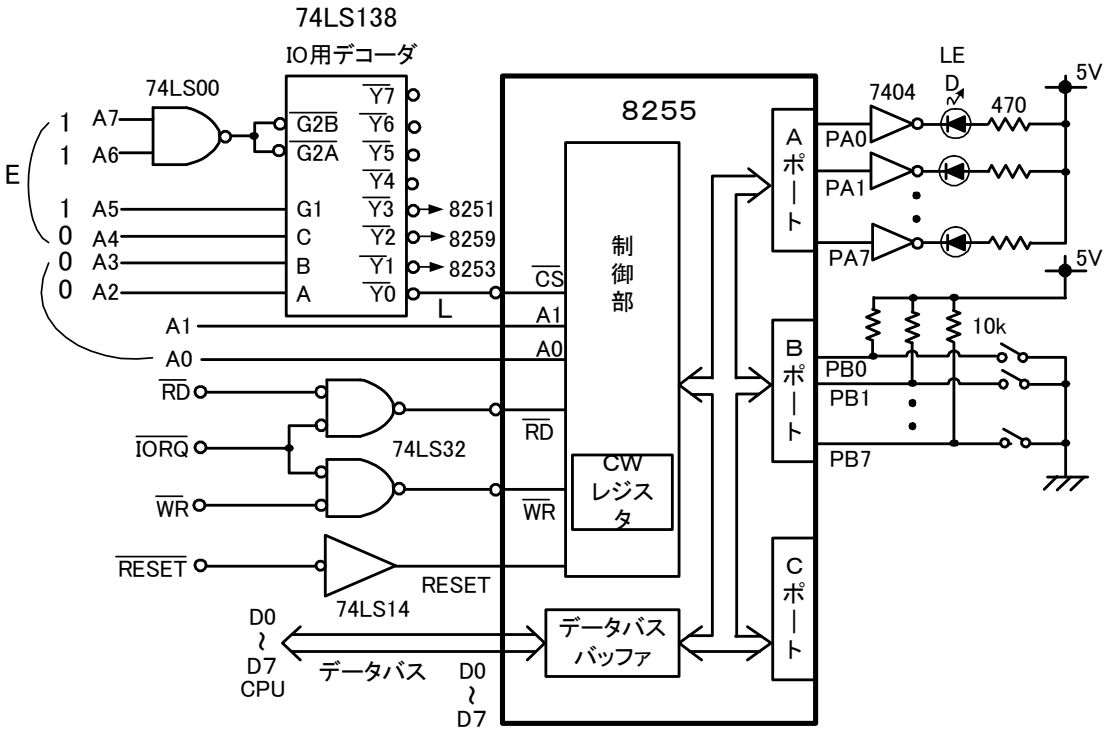


図 7-4　Z80 と 8255 の接続

表 7-6　8255 の基本機能

$\overline{CS}$	A１	A０	番　地	$\overline{RD}$	$\overline{WR}$	機能（データの流れ）
L	0	0	E０H	L	H	CPU←ポートA
L	0	1	E１H	L	H	CPU←ポートB
L	1	0	E２H	L	H	CPU←ポートC
L	0	0	E０H	H	L	CPU→ポートA
L	0	1	E１H	H	L	CPU→ポートB
L	1	0	E２H	H	L	CPU→ポートC
L	1	1	E３H	H	L	CPU→CW レジスタ
H	X	X	E0H～E3H 以外	X	X	ﾃﾞｰﾀﾊﾞｽは高ｲﾝﾋﾟｰﾀﾞﾝｽ状態

Xは“H”または“L”

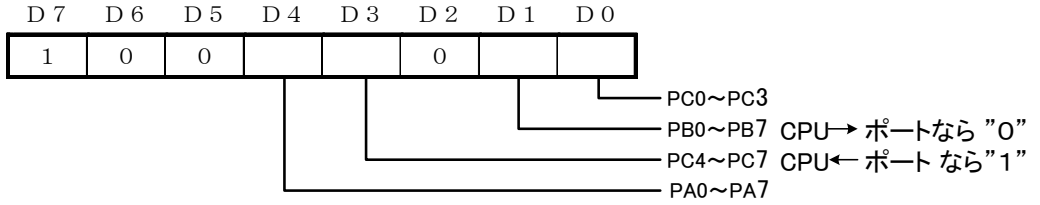


図 7-5　モード 0 における CW レジスタへの書き込み形式

（b）割り込み制御用 LSI　8259

割り込みの話をする前に，まず**サブルーチン**（または関数）について述べる。同じ手順のプログラム（データは異なっていてもよい）を複数回用いるときには，そのプログラムをサブルーチンにして，必要に応じて呼び出すことにすれば，全体のプログラムが短くなり，また，情報の流れが見やすくなる。図 7-6 (a)と(b)は全く等価なプログラムであるが，(b)の方が処理 A をサブルーチンにした分だけ短くなっている。(b)図のプログラム実行過程を説明する。

① 処理 1 を実行する。

② CALL　SUB を実行する。（表 7-5 参照）

処理 2 の先頭番地が自動的に**プログラムカウンタ（PC）**に入るが，これを**スタックポインタ（SP）**の示す番地のメモリへしまう。これは，PUSH　PC に相当する命令（実際には無い）を実行したことになる。次に，ラベル名 SUB の所すなわち処理 A の先頭番地へジャンプする。PC と SP は CPU の中のレジスタである（図 7-1 見よ）。

③ 処理 A を実行する。

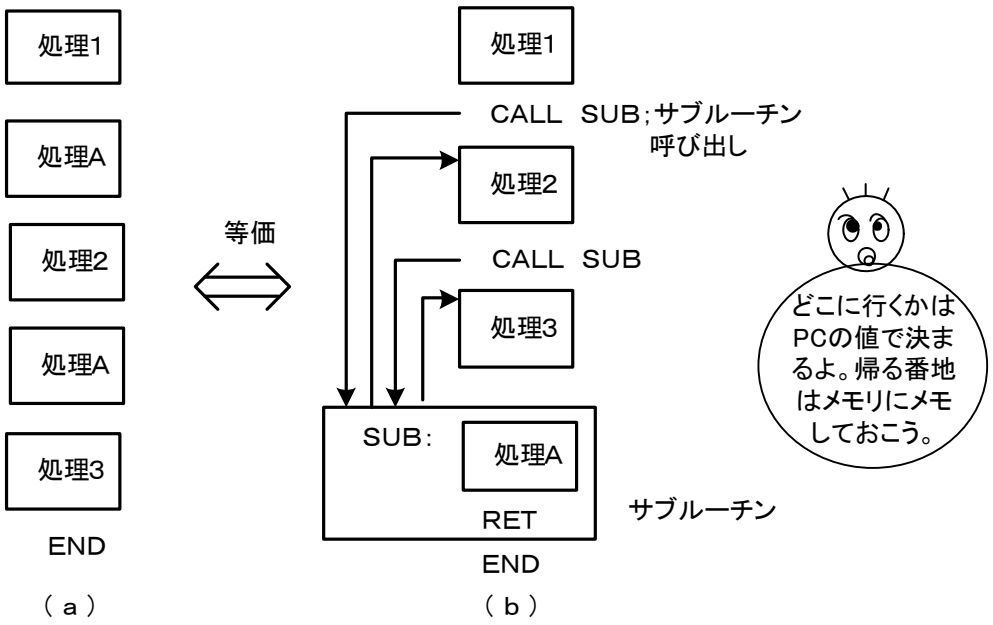


図 7-6　サブルーチン

④　RET 命令を実行する。（表 7-5 参照）

SP の示す番地のデータを PC にしまう。これは POP　PC に相当する命令（実際には無い）が実行されることになる。この結果，処理 2 の先頭番地へジャンプする。

⑤　処理 2 を実行する。

⑥　CALL　SUB を実行する。

⑦　処理 A を実行する。

⑧　RET を実行する。

⑨　処理 3 を実行する。

以上のことから，(a) のプログラムと（b）のプログラムは等価であることがわかる。スタックポインタ（SP）が番地を管理しているメモリ（RAM）の領域は**スタッカ**と呼ばれる。サブルーチンコールでは戻り番地を自動的にスタッカへ退避し, RET 命令でこれを PC に入れて元のプログラムへ復帰している。PUSH, POP の動作の詳細は, 表 7-5 を参照されたい。PUSH，POP 命令やサブルーチンを用いる場合には必ずプログラムの最初で SP のイニシャライズ（初期設定）を行う必要がある。

次に割り込みについて述べる。サブルーチンはプログラム実行中に CALL 命令があると実行されるが，外部からのハード的な信号（割り込み信号）によってもサブルーチン（**割り込み処理プログラム**）を実行することができる。これを**割り込み**（Interrupt）という。CPU が同時に実行できる命令は 1 つであるが，割り込みを使うことで複数の異なる処理を時分割で実現できる。ここでは割り込み制御用 LSI として 8259 を取り上げる。

図 7-7 に Z80 と 8259 の接続例を示す。マイコンの外部で作られた割り込み要求の信号が 8259 の IR0 ピンに入力されるとする。8259 にはコントロールワード(CW)レジスタがあり，CPU に実行して欲しい割り込み処理プログラムが置かれているメモリの先頭番地を予め設定しておく。この初期設定が終わった後で，EI 命令をどこかで実行して，Z80 が割り込み受け付け可能な状態にする。EI 命令を実行していないと CPU は割り込みを受け付けない。

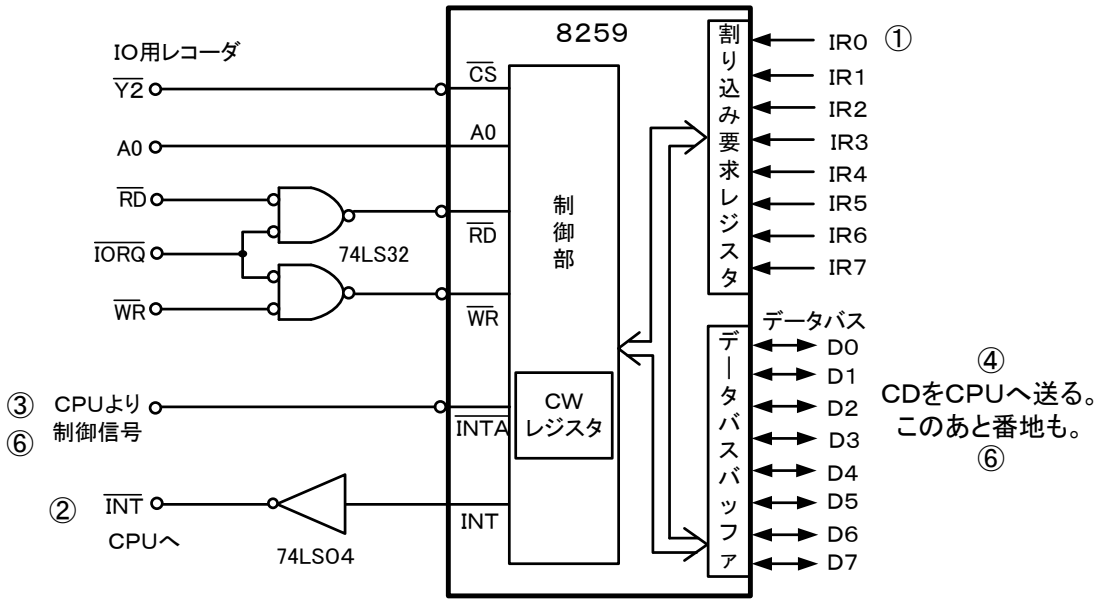


図 7-7　Z80 と 8259 の接続

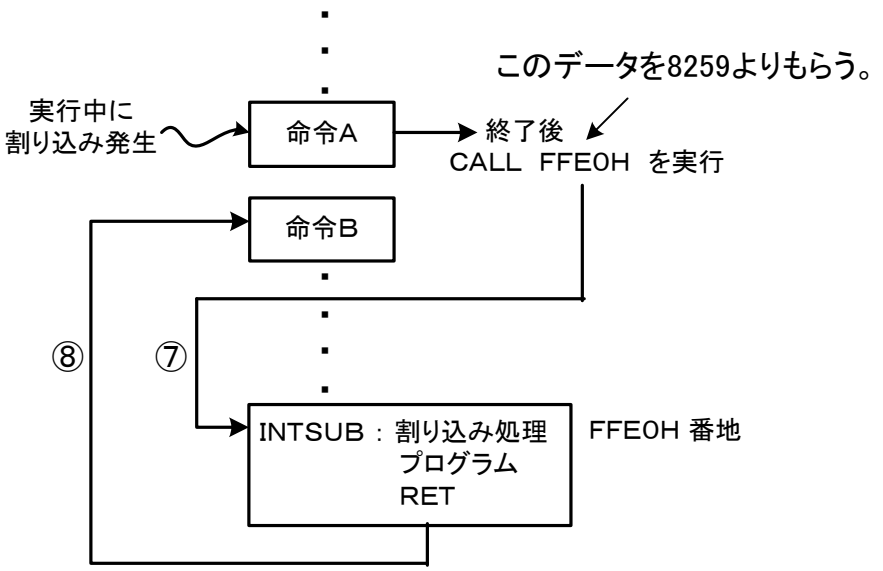


図 7-8　割り込み発生時に CPU が実行するプログラムの順序

図 7-8 のプログラムを用いて割り込みシーケンス説明する。初期設定で IR0 に対しては，FFE0H 番地が 8259 の CW の中に書いてあるとする。①～⑧は図中の番号に対応する。

① 命令 A を実行中に 8259 の IR0 端子が外部回路により“L”から“H”へ変化する。
② 8259 は CPU に $\overline{\text{INT}}$ 信号を出力する。“割り込みが来たよ”と知らせる。
③ EI 命令後とすると CPU は割り込みを受け付け，命令 A の実行の後 $\overline{\text{M1}}$ と $\overline{\text{IORQ}}$ を共に“L”にする。これらの信号によって，8259 の $\overline{\text{INTA}}$ に“L”信号が入力される（そのような論理回路が必要)。
④ 8259 はこの $\overline{\text{INTA}}$ 信号を受けて, データバスに CALL 命令(マシン語 CDH)を出力する。
⑤ CPU は CDH を読み込むと，CALL 命令と判るので，次いでジャンプ先の番地を読み込むためのメモリリードサイクルとなる。
⑥ CPU の $\overline{\text{RD}}$ が“L”となるので，$\overline{\text{INTA}}$ が再び“L”となる（そのような論理回路が必要)。この結果，8259 はジャンプ先番地の下位 E0H をデータバスに出力し，これを CPU が読み込む。同様にして，CPU はジャンプ先番地の上位 FFH を読み込む。これで 3 バイトの CALL 命令が読み終わった。
⑦ CPU は CALL　FFE0H を実行する。すなわち，命令 B の格納されている番地をスタッカへしまい，FFE0H へジャンプする。この結果 FFE0H から始まる割り込み処理プログラムが実行される。
⑧ 割り込み処理プログラムの最後の RET 命令で，スタッカを POP して中断していた処理プログラム（命令 B）へ戻り実行する。

以上のように割り込みはハード的に引き起こされたサブルーチンコールである。8259 は CPU へ割り込み処理プログラムが置かれているジャンプ先を知らせる役目がある。また，IR1～IR7 には別の割り込み処理プログラムの先頭番地を割り当てておけば，いろいろな処理が可能となる。同時に割り込みが入ることもあるので，8259 で優先順位を決めておくことができる。

7.5　割り込みを利用したマイコン制御システム

割り込みを利用したディジタル制御システムの例を図 7-9 に示す。*RL* 回路の電流を制御する最も簡単なマイコン制御システムである。第 1 章では，電源電圧は自由に変えられることを前提としたが，実際にはトランジスタを電子スイッチとして使ってゲート信号でオン，オフし，周期 T ごとの平均値を変える方法が用いられる（文献(20)）。つまり，トランジスタ Q がオンしたら $v = E$, Q がオフしたらダイオード D を通って電流が流れ $v = 0$ となる。ダイオード D がないと Q をオフした瞬間にコイルのスパイク電圧でトランジスタが破壊される。以上により，周期 T ごとの v の平均値は図 7-10 より

$$\overline{v} = \frac{T_{on}}{T} E \tag{7-1}$$

となる（ダイオードは理想的と仮定）。T_{on} をマイコンで計算し，ゲート信号発生器でその長さのパルスを作り，トランジスタのゲートに加えれば T_{on} 期間 Q をオンできる。

割り込みの利用について述べる。ゲート信号は周期 T ごとにオン電圧とオフ電圧としてトランジスタに加える（電圧は低いが波形は v と同じ）。このオン期間の長さで平均電圧 $\overline{v}$ が決るので，周期 T の間制御の計算は 1 回でよい。何故なら数回計算してもオン期間は周期 T ごとに 1 回しか変えられないからである。よって検出する電流も周期 T ごとに 1 回としよう。周期 T はゲート信号発生器で決まるので，T ごとにマイコンに割り込みをかける。図 7-11 に CPU が実行する制御プログラムの構成を示す。**メインプログラム**は起動時に実行され，初期値の設定（一度行うだけでよい）を行ったあと接続するパソコンとの通信を行う無限ループに入る。これは装置の電源がオフされるまで実行され続ける。**割り込み処理プログラム**はゲート信号発生器からのパルスで周期 T ごとに実行される。**割り込み処理プログラム**の中に A/D 変換器よる電流検出，電流制御演算（例 PI 制御），ゲート信号発生器への T_{on} 出力などを書いておく。CPU は同時に 2 つのプログラムを実行することはできず，割り込み処理プログラムが動いていない間だけメインプログラムが動く。

サンプリング周期 T は 100μs 程度に短くできるから，連続して v が変わる（連続系）と考えることもある。v がサンプリング周期 T ごとに階段状に変化すると仮定すれば零次ホールドがある場合のディジタル制御系として解析できる。

最近のマイコンは，図の点線で囲んだものに更に A/D 変換器までを内蔵したシングルチップになっているものも多い。安価でコンパクトさらに高速大容量へと進化している。本稿で述べたことはそのような場合でも基本的に変わることはない。

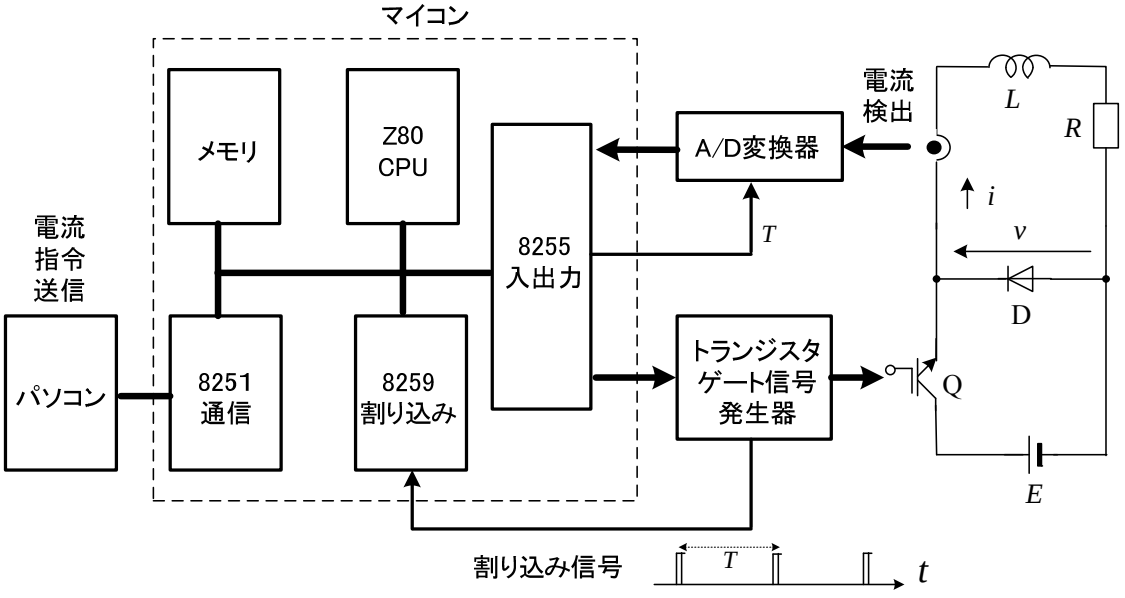


図 7-9　割り込みを利用したディジタル制御システム

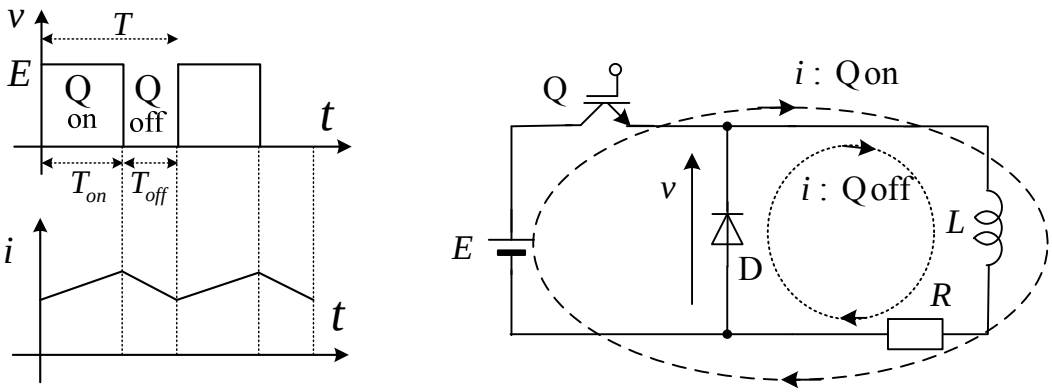
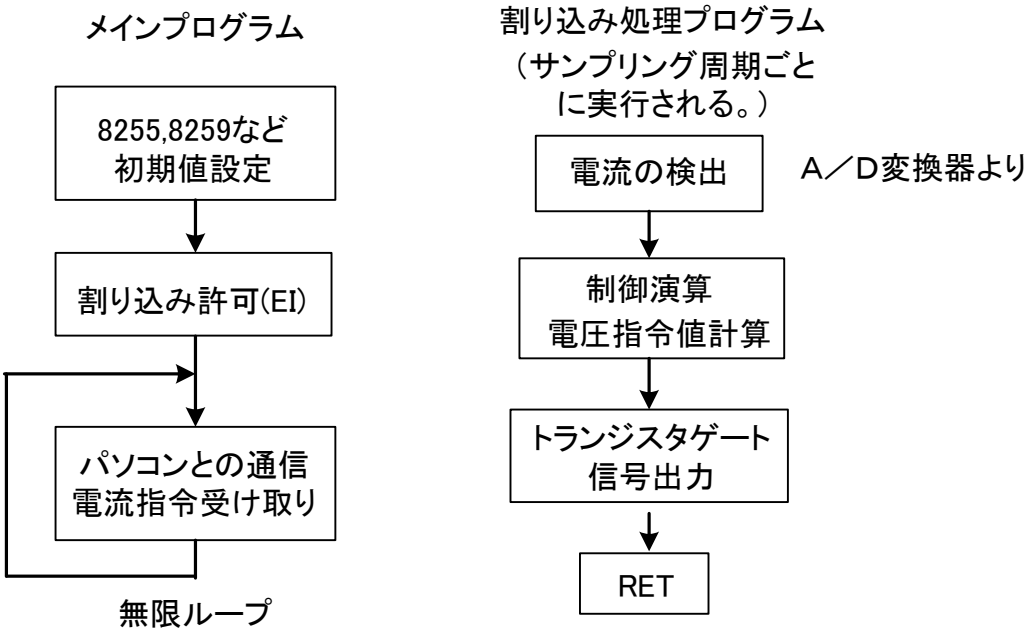


図 7-10　*RL* 負荷の電圧と電流



＊　これらのプログラムはパソコンで作りアセンブルしてマイコンのメモリに入れる。

図 7-11　CPU が実行する制御プログラムの構成

[問題 7-1]　表 7-4 のメモリマップを導出せよ。

[問題 7-2]　ソフトウェア（プログラム）が CPU で実行される仕組みを説明せよ。

[問題 7-3]　キャリフラグ（C フラグ）とオーバフローフラグ（V フラグ）は何故必要か説明せよ。

[問題 7-4]　8255 の働きを説明せよ。

[問題 7-5]　割り込みとは何か。何故必要か。

[問題 7-6]　8259 の働きを説明せよ。