TeC 教科書

Ver. 2.1.1

徳山工業高等専門学校
情報電子工学科

目 次

第1章　はじめに

1.1　この科目で学ぶこと

現代，コンピュータと呼ばれているものはスーパーコンピュータと呼ばれる高価で高性能なものから，マイコンと呼ばれ炊飯器やエアコンに組み込まれている小型のものまで，全て同じ原理に基づき動作しています.

この原理は，1946 年に米国の数学者フォン・ノイマン (Von Neumann) が提案したと言われています．現在のコンピュータの，ほぼ，全てのものは，ノイマンの提案した同じ原理を使用しており「ノイマン型コンピュータ」と呼ばれます.

「ノイマン型コンピュータ」が出現して既に 60 年以上の年月が経過しましたが，未だにこれを上回る「自動機械」の構成方法は発明されていません．だから，パソコンのような一般の人がコンピュータだと考えている装置だけでなく，炊飯器やエアコンの制御装置のようなコンピュータらしくない装置まで,「自動機械」が必要なところには全て「ノイマン型コンピュータ」が使用されているのです．恐らく，あと数十年は,「ノイマン型コンピュータ」の時代が続くでしょう.

この教科書では，教育用コンピュータ (TeC) を用いて，この「ノイマン型コンピュータ」の動作原理を，しっかり勉強できるようになっています．ここでしっかり勉強しておけば，将来，どんな新型のコンピュータに出会ったとしても，恐れることはありません．所詮,「ノイマン型コンピュータ」の一種ですから，皆さんはその正体を簡単に見抜くことができるはずです.

「ノイマン型コンピュータ」の原理をきちんと理解しておくことは，皆さんにとって，寿命の長いエンジニアになるため大事なステップとなります．しっかり，がんばって下さい.

1.2　勉強の進め方

この教科書は，高専や工業高校の 1 年生と 2 年生が，教育用コンピュータ TeC を教材に，ノイマン型コンピュータの基本を学ぶことを想定して作ってあります．1 章から 5 章の途中までを 1 年で学び，残りを 2 年生 (半年〜1 年間) で学びます.

1 年生では，(1) コンピュータの内部で情報を表現する方法，(2)TeC の組み立て，(3)TeC の基本的なプログラミングを学びます．2 年生では，TeC を用いた高度なプログラミングを学びます.

1.3　教材用コンピュータ

私達の身近にあるパーソナルコンピュータ (パソコン) や携帯電話のようなコンピュータシステムは，高度で複雑すぎて「ノイマン型コンピュータ」の原理を学ぶための教材としては適していません.

原理を学ぶのに適した単純で小さなコンピュータ (マイコン) を，専用に開発しました．このマイコンは TeC(Tokuyama Educational Computer：徳山高専教育用コンピュータ) と呼ばれます.

特徴は単純で小さなことです．単純で小さなことで，次のようなメリットがあります.

単純　ノイマン型コンピュータの本質的な部分だけを勉強しやすい．現在のパソコン等は，勉強するには難しすぎる.

小型　自宅に持ち帰り宿題ができる．授業時間外でも，納得がいくまで色々と試してみることができる.

第2章　情報の表現

この章では，コンピュータ内部でどのように情報が表現されているのか，その情報をどのような回路で扱うことができるのか，簡単に紹介します．

2.1　コンピュータ内部の情報表現

人は，情報を音声や文字，絵等で表現することができます．コンピュータも表面的には音声や文字，絵を扱うことができますが，コンピュータの内部は電子回路で構成されているので，最終的には電圧や電流で情報を表現せざるを得ません．

$$\text{情報の表現} = \left(\begin{array}{l} \text{人：音声，文字，絵,…} \\ \text{コンピュータ：電圧，電流} \end{array} \right)$$

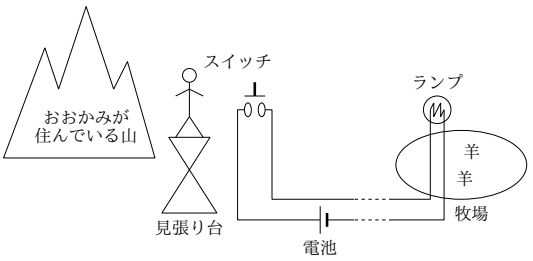
2.1.1　電気を用いた情報の表現

電圧や電流で情報表現する方法には，いろいろなアイデアがあります．しかし，現在のコンピュータは，電圧が**ある/ない (ON/OFF)** か，電流が**流れている/流れていない (ON/OFF)** のような，二つの状態だけを用いて情報を表現する方法を使っています．(デジタル・コンピュータ)

コンピュータの情報表現
　電圧が**ある/ない**
　電流が**流れる/ながれない**
　　↓
　ON と OFF の二つの状態だけを用いて
　情報を表現する．(回路が作りやすい)

次の例を見て下さい．電気の「ON/OFF」で「おおかみが来たか/来ていないか」のどちらの状態なのかを伝達できる「情報の表示装置」を実現することができます．電気の「ON/OFF」を用いて情報を表現することができました．

例 おおかみが来た情報表示装置
　ランプ点灯：おおかみが来た
　ランプ消灯：おおかみが来ていない



2.1.2　ビット

前の例では，ランプの「ON/OFF」を用いて「二つの状態のどちらなのか」を表しました．このような,「二つのどちらか」を表す情報が「情報の最小単位」になります．情報の最小単位のことを**「ビット (bit)」**と呼びます．

on/off のどちらか → 情報の最小単位 (ビット)

通常，ビットの値は「ON/OFF」ではなく,「1/0」で書きます．

$$\left(\begin{array}{lcl} \text{ON} & : & 1 \\ \text{OFF} & : & 0 \end{array} \right)$$

「おおかみが来た情報」は，ビットの値に次のように対応付けできます．

ビット値	おおかみが来た情報
0(off)	おおかみがきていない
1(on)	おおかみが来た !!!

2.1.3　より複雑な情報の表現

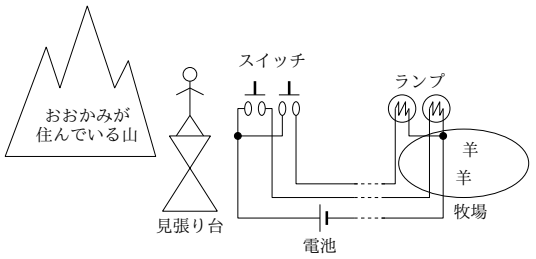
二つの状態では表現できない，もっと複雑な情報はどうやって表現したら良いでしょうか．

前の例で，やって来たおおかみの頭数により牧場の人が対応を変化させたい場合が考えられます．そのためには,「来たか/来なかったか」だけの情報だけでは十分ではなく，**「たくさん来たか」**を知らせる必要があります．それには，複数のビットを組み合わせて使用します．

以下に，複数のビットの組み合わせて表現した「拡張おおかみが来た情報」を示します．

ビット値	拡張おおかみが来た情報	
00	おおかみがきていない	平気
01	おおかみが 1 頭来た	戦う
10	おおかみが 2 頭来た	？
11	おおかみがたくさん来た	逃げる

例 拡張おおかみが来た情報表示装置



この例で見たように，2 ビットを用いれば 4 種類の情報を表現することができます．

一般に，n ビットを用いると 2^n 種類の情報を表現することができます．システム内で必要なビット数を決めて，それの組合せに意味付けをすれば，どんな情報だって表現できます．

ビット数	ビットの組合せ	組合せ数
1	0 1	2
2	00 01 10 11	4
3	000 001 010 011 100 101 110 111	8
…	…	
n		2^n

ビットだけでは情報の単位として小さすぎるので，4 ビットまとめたもの，8 ビットまとめたものにも名前があります．

「4 ビット」　＝　「1 ニブル」
「8 ビット」　＝　「1 バイト」

2.2　数値の表現

ビットの組合せを，どのように意味付けするかは，前節の例のように「システムが扱う必要のある情報」により，毎回，約束すればよいのです．しかし，どのコンピュータでも同じ方法で意味付けされている情報もあります．それの一つが数値の表現方法です．

2.2.1　2 進数

コンピュータの内部では，数値を 2 進数で表すのが普通です．我々が普段使用している 10 進数と，2 進数の特徴比較を次に示します．

10 進数と 2 進数の比較

10 進数の特徴
(1) 0 〜 9 の 10 種類の数字を使用する．
(2) 1 桁毎に 10 倍の重みをもつ．

2 進数の特徴
(1) 0 と 1 の 2 種類の数字を使用する．
(2) 1 桁毎に 2 倍の重みをもつ．

コンピュータの内部では，ビットが情報の表現に使用されています．そこで，ビット値の 0/1 をそのまま 2 進数の 1 桁と考えれば数値が表現できます．

例えば，4 ビット用いると 0 〜 15 の数が次のように表現できます．

ビット 3 (b_3)	ビット 2 (b_2)	ビット 1 (b_1)	ビット 0 (b_0)	意味
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

一般に n ビットで 0 〜 $2^n - 1$ の範囲の数を表現することができます．

2.2.2　2 進数と 10 進数の相互変換

前の 4 ビットの例なら，2 進数と 10 進数の対応を暗記することが可能です．しかし，8 ビットの場合ならどうでしょう ？

組合せは 256 もあり，とても暗記できそうにありません．対応を計算で求める必要があります．

1. 2 進から 10 進への変換

   2 進数の桁ごとの重みは，桁の番号を n とすると 2^n になります．

$$\begin{array}{cccc} b_3 & b_2 & b_1 & b_0 \\ 2^3 = 8 & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \end{array}$$

   2 進数の数値は，その桁の重みと桁の値を掛け合わせたものの合計です．例えば 2 進数の 1010_2 は，2^3 の桁が 1，2^2 の桁が 0，2^1 の桁が

1，2^0 の桁が 0 ですから，次のように計算できます.

$$\begin{aligned}1010_2 &= 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0\\ &= 1\times 8 + 0\times 4 + 1\times 2 + 0\times 1\\ &= 8 + 0 + 2 + 0\\ &= 10_{10}\end{aligned}$$

2. 10 進から 2 進への変換

次に，10 進数を 2 進数に変換する方法を考えます．ここで着目するのは桁の移動です.

10 進数では，値を 10 で割ると右に 1 桁移動します．2 進数では，2 で割ると右に 1 桁移動します．どちらの場合でも，割算をした時の余りは，最下位の桁からはみ出した数になります.

つまり，数値を 2 で割った時の余りは 2 進数を右に 1 桁移動したときはみ出してきた数を表しています.

そこで，2 で割る操作を繰り返しながらはみ出して来た数を記録すれば，もとの数を 2 進数で表したときの 0/1 の並びが分かります.

$$\begin{array}{rcl} & 12_{10} & = & 1100_2 \\ 1/2\downarrow & & & \\ & 6^{\cdots 0}_{10} & = & 0110^{\cdots 0}_{2} \\ 1/2\downarrow & & & \\ & 3^{\cdots 0}_{10} & = & 0011^{\cdots 0}_{2} \\ 1/2\downarrow & & & \\ & 1^{\cdots 1}_{10} & = & 0001^{\cdots 1}_{2} \\ 1/2\downarrow & & & \\ & 0^{\cdots 1}_{10} & = & 0000^{\cdots 1}_{2} \end{array}$$

計算方法と計算例

$$\begin{array}{l} 2)\ \underline{100} \\ 2)\ \ \underline{50}\cdots 0 \\ 2)\ \ \underline{25}\cdots 0 \\ 2)\ \ \underline{12}\cdots 1 \\ 2)\ \ \underline{6}\cdots 0 \\ 2)\ \ \underline{3}\cdots 0 \\ 2)\ \ \underline{1}\cdots 1 \\ \quad 0\cdots 1 \end{array}$$

余りを右から順に並べると 1100100_2

2.2.3 16 進数

2 進数は，桁数が多くなり書き表すのに不便です．そこで，2 進数 4 桁をまとめて 16 進数一桁で書き表します．9 より大きな数字が無いので，16 進数ではアルファベットを数字の代用にします.

2 進数	16 進数	10 進数
0000_2	0_{16}	0_{10}
0001_2	1_{16}	1_{10}
0010_2	2_{16}	2_{10}
0011_2	3_{16}	3_{10}
0100_2	4_{16}	4_{10}
0101_2	5_{16}	5_{10}
0110_2	6_{16}	6_{10}
0111_2	7_{16}	7_{10}
1000_2	8_{16}	8_{10}
1001_2	9_{16}	9_{10}
1010_2	A_{16}	10_{10}
1011_2	B_{16}	11_{10}
1100_2	C_{16}	12_{10}
1101_2	D_{16}	13_{10}
1110_2	E_{16}	14_{10}
1111_2	F_{16}	15_{10}

n 進数の表記

日常生活では，数値を書き表すときは「いつも 10 進数」で書きます．しかし，コンピュータの世界では 2 進数，10 進数，16 進数を場合により使い分けます．そのため，何進数で書いてあるのか分からなくて困ることがあります.

そこで，上の表のように数値の右に小さな字で何進数かを書き加えます．ときには，数字の代わりに「2 進数='b'」,「16 進数='H'」を加えることもあります.

例えば,「01100100_2 を 01100100_b」,「64_{16} を 64H」のように書きます.

問題

1. 上の「2 進数，16 進数，10 進数対応表」を暗記しなさい.
2. 10 進数の 16，50，100，127，130 を，2 進数 (8 桁)，16 進数 (2 桁) で書き表しなさい.
3. 2 進数の 00011100，00111000，11100000 を 16 進数 (2 桁)，10 進数で書き表しなさい.
4. 16 進数の 1F, AA を 2 進数 (8 桁)，10 進数で書き表しなさい.

2.3　負数の表現

拡張おおかみが来た情報表示装置では,「表示装置の二つのビット (二つのランプ) を，あのように読む」ことを約束しました．つぎに，数値の場合は,「n 個のビットを 2 進数として読む」ことを約束しました.

今度は，負の数が必要になりました．そこで，ビットの新しい読み方を約束します．この節で出てくるビットは「符号付き数値」を表しています．以下では,「符号付き数値を表すビット」をどのように読むかを説明します.

2.3.1　符号付き絶対値表現

使用できるビットのうち一つを，符号を表すために使用します．これを「符号ビット」と呼ぶことにします．通常，符号ビットには最上位 (左端) のビットを使用します.

次の例のように 4 ビット使用して，$-7 \sim +7$ の範囲を表すことができます．この表現方法は分かりやすくて都合が良いのですが，実際に使われることはあまりありません.

符号付き絶対値表現 (4 ビット) の例	
-7	1111_2
-6	1110_2
-5	1101_2
...	...
-1	1001_2
-0	1000_2
$+0$	0000_2
$+1$	0001_2
...	...
$+5$	0101_2
$+6$	0110_2
$+7$	0111_2

2.3.2　補数表現

A から B を引いた数を，B の A に対する「補数」と言います．例えば，10 進数の世界で次のような例があります.

10 進数の世界で補数の例		
$A = 9$	9	6 は 3 の 9 に
$B = 3$	-3	対する補数
	6	
$A = 10$	10	7 は 3 の 10 に
$B = 3$	-3	対する補数
	7	
$A = 99$	99	57 は 42 の 99 に
$B = 42$	-42	対する補数
	57	
$A = 100$	100	58 は 42 の 100 に
$B = 42$	-42	対する補数
	58	

コンピュータの内部で使用される 2 進数の世界では,「1 の補数表現」と「2 の補数表現」の 2 種類があります．次の例に示すように $11...1_2$ に対する補数のことを「1 の補数」，$10...0_2$ に対する補数のことを「2 の補数」と呼びます.

2 進数の世界で補数の例		
$A = 1_2$	1_2	0_2 は 1_2 の
$B = 1_2$	-1_2	1_2 に対する
	0_2	補数
$A = 10_2$	10_2	01_2 は 01_2 の
$B = 01_2$	-01_2	10_2 に対す
	01_2	る補数
$A = 11_2$	11_2	10_2 は 01_2 の
$B = 01_2$	-01_2	11_2 に対す
	10_2	る補数
		(1 の補数)
$A = 100_2$	100_2	011_2 は 001_2 の
$B = 001_2$	-001_2	100_2 に対す
	011_2	る補数
		(2 の補数)

2.3.3 1の補数による負数の表現

補数で負の数を表現する方法について説明します．まず，最初は，4ビット2進数の1の補数を用いる例です．次の表は0 〜 7 の1の補数を計算したものです．

4ビット2進数の1補数	
0	$1111_2 - 0000_2 = 1111_2$
1	$1111_2 - 0001_2 = 1110_2$
2	$1111_2 - 0010_2 = 1101_2$
3	$1111_2 - 0011_2 = 1100_2$
4	$1111_2 - 0100_2 = 1011_2$
5	$1111_2 - 0101_2 = 1010_2$
6	$1111_2 - 0110_2 = 1001_2$
7	$1111_2 - 0111_2 = 1000_2$

「0001_2 の1に対する補数を -0001_2 を表現するために使用する.」,「0010_2 の1に対する補数を -0010_2 を表現するために使用する.」と約束すれば，次の表のように -7 〜 $+7$ の範囲の数を表現できます．

1の補数を用いた符号付き数値	
-7	1000_2
-6	1001_2
-5	1010_2
-4	1011_2
-3	1100_2
-2	1101_2
-1	1110_2
-0	1111_2
$+0$	0000_2
$+1$	0001_2
$+2$	0010_2
$+3$	0011_2
$+4$	0100_2
$+5$	0101_2
$+6$	0110_2
$+7$	0111_2

「1の補数を負の数を表現するために使用する.」と約束することにより，4ビットで -7 〜 $+7$ までの数を表現することができました．

1の補数を使用した方法も，実際に使われることはあまりありません．コンピュータの内部で実際に使用されるのは，次に説明する2の補数による表現です．

2.3.4 2の補数による負数の表現

まず最初は，4ビット2進数の2の補数を用いる例です．次の表は0 〜 8 の2の補数を計算したものです．5ビットで表現さている部分もありますが，四角で囲んだ4ビットに注目してください．

4ビット2進数の2補数			
0	$1\boxed{0000}_2$	$-\boxed{0000}_2 =$	$1\boxed{0000}_2$
1	$1\boxed{0000}_2$	$-\boxed{0001}_2 =$	$\boxed{1111}_2$
2	$1\boxed{0000}_2$	$-\boxed{0010}_2 =$	$\boxed{1110}_2$
3	$1\boxed{0000}_2$	$-\boxed{0011}_2 =$	$\boxed{1101}_2$
4	$1\boxed{0000}_2$	$-\boxed{0100}_2 =$	$\boxed{1100}_2$
5	$1\boxed{0000}_2$	$-\boxed{0101}_2 =$	$\boxed{1011}_2$
6	$1\boxed{0000}_2$	$-\boxed{0110}_2 =$	$\boxed{1010}_2$
7	$1\boxed{0000}_2$	$-\boxed{0111}_2 =$	$\boxed{1001}_2$
8	$1\boxed{0000}_2$	$-\boxed{1000}_2 =$	$\boxed{1000}_2$

「0001_2 の2に対する補数を -0001_2 を表現するために使用する.」,「0010_2 の2に対する補数を -0010_2 を表現するために使用する.」と約束すれば，次の表のように -8 〜 $+7$ の範囲の数を表現できます．

2の補数を用いた符号付き数値	
-8	1000_2
-7	1001_2
-6	1010_2
-5	1011_2
-4	1100_2
-3	1101_2
-2	1110_2
-1	1111_2
0	0000_2
1	0001_2
2	0010_2
3	0011_2
4	0100_2
5	0101_2
6	0110_2
7	0111_2

「2の補数を負の数を表現するために使用する.」と約束することにより，4ビットで -8 〜 $+7$ までの数を表現することができました．一般に，2の補数表現を用いたnビット符号付き2進数が表現できる数値の範囲は次の式で計算できます．

$$-2^{n-1} \ \sim \ 2^{n-1} - 1$$

2 の補数表現を用いると，2 進数の足し算・引き算が，正負のどちらの数でも同じ手順で計算できます (詳しくは後述).

「手順が同じ＝演算回路が同じ」ことになりますので，実際にコンピュータを製作する上では非常に都合がよく，現在のコンピュータは，ほとんどの機種で 2 の補数表現を採用しています.

2.3.5　2 の補数を求める手順

$-A$ を n ビット 2 の補数表現 (B) で表します．2 の補数の定義より，B は次のように計算できます.

$$\begin{aligned} B &= 2^n - A \\ &= (2^n - 1 - A) + 1 \\ &= (A \text{ の } 1 \text{ の補数}) + 1 \end{aligned} \tag{2.1}$$

「A の 1 の補数」は，A のビットの'0' と'1' を入れ換えた (ビット反転した) ものですので，簡単に求めることができます.

つまり,「A の 2 の補数」は「A をビット反転した後，1 を加える」ことにより求めることができます.

ビット反転した後，1 を加える.

2.3.6　2 の補数から元の数を求める手順

次に，逆変換について考えます．A の 2 の補数 B から A を求める手順です.

$$\begin{aligned} B &= 2^n - A \\ 0 &= 2^n - A - B \\ A &= 2^n - B \end{aligned} \tag{2.2}$$

(2.1) 式と (2.2) 式では，A と B が入れ換わっただけで，同じ型をしています．このことから，A から B を求める計算と，B から A を求める計算の手順が同じことが分かります.

2 の補数を求めるのと同じ計算で，2 の補数から元の数を求めることができます.

ビット反転した後，1 を加える.

2.4　2 進数の計算

ここでは，2 進数の和と差の計算方法を学びます.

2.4.1　正の数の計算

2 進数の計算も 10 進数と同じ要領です．桁上がりが 10 ではなく，2 で発生することに注意してください.

もちろん，2 進数で計算しても 10 進数で計算しても同じ計算結果になります.

2 進数と 10 進数の計算を比較

10 進数		2 進数		
	07		0111_2	(10 進の 7)
+	05	+	0101_2	(10 進の 5)
	12		1100_2	(10 進の 12)
	12		1100_2	(10 進の 12)
−	05	−	0101_2	(10 進の 5)
	07		0111_2	(10 進の 7)

何進数で計算しても同じ結果になる.

2.4.2　負の数を含む計算

2 の補数表現を用いると，正の数だけのときと同じ要領で負の数を含む計算ができます．ここでは和の計算を例に説明します.

正の数と負の数の和

正の数 (X) と負の数 $(-A)(A$ の 2 の補数 $(B))$ の和の計算を考えます.

$X + B$ の計算 $(X + (-A))$

1. 結果が負の場合 $(|A| > |X|)$
   解は $-(A - X)$ になるはず！

$$\begin{aligned} X + B &= X + (2^n - A) \\ &= 2^n - (X - A) \end{aligned} \tag{2.3}$$

(2.3) 式は，正解 $-(A - X)$ の 2 の補数表現になっている.

例　$3 + (-5) = -2$ (4 ビット 2 の補数表現)

3 を 2 進数に変換すると 0011_2
-5 を 2 進数に変換すると 1011_2

「和を計算する」
$0011_2 + 1011_2 = 1110_2 = -2_{10}$

2. 結果が正の場合 ($|X| \geq |A|$)
解は $(X - A)$ になるはず！

$$\begin{aligned} X + B &= X + (2^n - A) \\ &= 2^n + (X - A) \qquad (2.4) \\ &= X - A \end{aligned}$$

(2.4) 式の 2^n は，桁あふれにより結果に残らないので，正解 $(X - A)$ となる．

例　$5 + (-3) = 2$ (4 ビット 2 の補数表現)

5 を 2 進数に変換すると 0101_2
-3 を 2 進数に変換すると 1101_2

「和を計算する」
$0101_2 + 1101_2 = \boxed{1}0010_2 = 2_{10}$
(5 ビット目の 1 は桁あふれで消滅)

負の数と負の数の和

負の数 $(-A_1)$(2 の補数 (B_1)) と
負の数 $(-A_2)$(2 の補数 (B_2)) の和
$B_1 + B_2$ の計算 $((-A_1) + (-A_2))$
解は $-(A_1 + A_2)$ になるはず！

$$\begin{aligned} B_1 + B_2 &= (2^n - A_1) + (2^n - A_2) \\ &= 2^n + (2^n - (A_1 + A_2)) \qquad (2.5) \\ &= 2^n - (A_1 + A_2) \qquad (2.6) \end{aligned}$$

(2.5) 式の最初の 2^n は，桁あふれにより消滅する．(2.6) 式は正解 $(-(A_1 + A_2))$ の 2 の補数表現になっている．

例　$(-1) + (-3) = 2$ (4 ビット 2 の補数表現)

-1 を 2 進数に変換すると 1111_2
-3 を 2 進数に変換すると 1101_2

「和を計算する」
$1111_2 + 1101_2 = \boxed{1}1100_2 = -4_{10}$
(5 ビット目の 1 は桁あふれで消滅)

以上のように，2 の補数表現を使用すると負の数を含んだ足し算が簡単に計算できます．ここでは省略しますが，引き算も同様に正負の区別をすることなく計算できます．

MSB と LSB

「2 の補数表現を用いると，最上位ビットが 1 なら負の数と分かります.」のように最上位ビット と言う言葉をよく使います.

「最上位ビット」と言うのは長いので，これを英語にして (Most Significant Bit) 頭文字を取り MSB と略すことが良くあります．憶えておいて下さい.

同様に「最下位ビット」のことは英語で「Least Significant Bit」なので，略して LSB と呼びます．これも，憶えておきましょう．

問題

1. 次の数を「2 の補数表現を用いた 4 ビット符号付き 2 進数」で書き表しなさい.
3_{10}，-3_{10}，5_{10}，-5_{10}，6_{10}，-6_{10}

2. 次の数を「2 の補数表現を用いた 8 ビット符号付き 2 進数」で書き表しなさい.
3_{10}，-3_{10}，8_{10}，-8_{10}，
30_{10}，-30_{10}，100_{10}，-100_{10}

3. 次の「2 の補数表現を用いた 4 ビット符号付き 2 進数」を，10 進数で書き表しなさい.
1100_2，1011_2，0100_2，1101_2

4. 次の 2 進数は「2 の補数表現を用いた符号付き 2 進数」です．空欄を埋めなさい.

1)
$$\begin{array}{rr} & 0011\ 1100_2 \\ + & 0010\ 1101_2 \\ \hline & \square_2 \end{array} \rightarrow \begin{array}{rr} & \square_{10} \\ + & \square_{10} \\ \hline + & \square_{10} \end{array}$$

2)
$$\begin{array}{rr} & 0110\ 0100_2 \\ + & 1000\ 0001_2 \\ \hline & \square_2 \end{array} \rightarrow \begin{array}{rr} & \square_{10} \\ + & \square_{10} \\ \hline + & \square_{10} \end{array}$$

3)
$$\begin{array}{rr} & 1110\ 0100_2 \\ + & 0100\ 0001_2 \\ \hline & \square_2 \end{array} \rightarrow \begin{array}{rr} & \square_{10} \\ + & \square_{10} \\ \hline + & \square_{10} \end{array}$$

4)
$$\begin{array}{rr} & 1110\ 0100_2 \\ + & 1100\ 0001_2 \\ \hline & \square_2 \end{array} \rightarrow \begin{array}{rr} & \square_{10} \\ + & \square_{10} \\ \hline + & \square_{10} \end{array}$$

2.5　小数の表現

この節では，小数を 2 進数で表現する方法を紹介します.

2.5.1　2 進数による小数の表現

小数を含む数を，コンピュータ内部で，2 進数を用いて表現する方法は何種類かあります．ここでは，その中でも最も基本的な，固定小数点形式を紹介します.

これまでの 2 進数は，暗黙のうちに小数点が右端にあると考えました．小数点の位置を右端以外だと約束すれば，小数を含む数の表現も可能になります.

例　4 ビットで小数点が 2 桁目なら

$00.00_2 = 0.0_{10}$
$00.01_2 = 0.25_{10}$
$00.10_2 = 0.5_{10}$
$00.11_2 = 0.75_{10}$
$01.00_2 = 1.0_{10}$
$01.01_2 = 1.25_{10}$
$01.10_2 = 1.5_{10}$
$01.11_2 = 1.75_{10}$
$10.00_2 = 2.0_{10}$
...
$11.11_2 = 3.75_{10}$

小数点を中心に，左は 1 桁毎に 2 倍，右は 1 桁毎に 1/2 倍になります．(10 進数では，左は 1 桁毎に 10 倍，右は 1 桁毎に 1/10 倍になりましたね．)

2.5.2　10 進数との相互変換

1. 2 進数から 10 進数への変換
   整数の場合と同様に桁の重みを加算すれば 10 進数に変換できます.

   例　2 進 10 進変換

   $10.01_2 = 2 + 1/4 = 2.25_{10}$

2. 10 進数から 2 進数への変換
   整数の場合は，1/2 倍することで右にシフト (桁移動) し，小数点を横切った 0/1 を判定しました．(小数点を 1 が横切ると，余りが出ていました．)

   小数点以下の数値は，2 倍することで左にシフトし，小数点を横切った 0/1 を判定すれば 2 進数に変換できます.

例 小数を表す 10 進数から 2 進数への変換

2 進数	×2 は	10 進数
0.101_2	左シフ	0.625
↙	トと同じ	× 2
1.010_2		$\underline{1}.250$

2 進数	×2 は	10 進数
0.010_2	左シフ	0.250
↙	トと同じ	× 2
0.100_2		$\underline{0}.500$

2 進数	×2 は	10 進数
0.100_2	左シフ	0.500
↙	トと同じ	× 2
1.000_2		$\underline{1}.000$

10 進数で計算したとき，小数点を横切って整数部に出てきた数を小数点の右に順番に並べる.

$$0.\underline{101}_2$$

3. 整数部と小数部の両方がある場合
   整数部分と小数部分を分離して，別々に計算します.

問題

1. 次の 2 進数を 10 進数に変換しなさい.
   (a) 0.1001_2
   (b) 0.0101_2
   (c) 11.11_2
2. 次の 10 進数を 2 進数に変換しなさい.
   (a) 0.0625_{10}
   (b) 0.1875_{10}
   (c) 0.4375_{10}
3. 次の 10 進数を 2 進数に変換しなさい．(難しい問題)
   (a) 0.8_{10}
   (b) 0.7_{10}

2.6 文字の表現

この節では，文字を 2 進数で表現する方法を紹介します．(文字情報を表しているビットの読み方を約束する．)

2.6.1 文字コード

文字の場合，数値のような規則性を期待することはできません．そこで，使用する文字の一覧表を作成し，1 文字毎に対応する 2 進数 (コード) を決めます．この一覧表のことを「文字コード表」と呼びます．文字コード表を各自 (コンピュータメーカ等) が勝手に定義すると，コンピュータの間でのデータ交換に不便です．そこで，規格として制定してあります．

2.6.2 ASCII コード

ASCII(American Standard Code for Information Interchange) コードは，1963 年にアメリカ規格協会 (ANSI) が定めた情報交換用の文字コードです．英字，数字，記号等が含まれます．現在のパーソナルコンピュータ等で使用される文字コードは，ASCII コードが基本になっています．

ASCII 文字コード表

(下位4ビット) \ (上位4ビット)	0	1	2	3	4	5	6	7
0		DE(SP)	0	@	P	‘	p	
1	SH	D1	!	1	A	Q	a	q
2	SX	D2	"	2	B	R	b	r
3	EX	D3	#	3	C	S	c	s
4	ET	D4	$	4	D	T	d	t
5	EQ	NK	%	5	E	U	e	u
6	AK	SN	&	6	F	V	f	v
7	BL	EB	’	7	G	W	g	w
8	BS	CN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SB	*	:	J	Z	j	z
B	HM	EC	+	;	K	[	k	{
C	CL	→	,	<	L	\	l	|
D	CR	←	-	=	M	]	m	}
E	SO	↑	.	>	N	^	n	~
F	SI	↓	/	?	O	_	o	DEL

文字コード表で 00H 〜 1FH と 7FH は，機能 (改行等) を表す特殊な文字になっています．20H(SP) は空白を表す文字です．

ASCII 文字コードは 7 ビットで表現されます．しかし，コンピュータの内部では 1 バイト (8 ビット) 単位の方が扱いやすいので，最上位に 0_2 を補って 8 ビットで扱うことがほとんどです．

2.6.3 JIS 文字コード

日本では JIS(Japan Industrial Standard：日本工業規格) の一部として，8 ビットコード (英数記号＋カナ) と，16 ビットコード (英数記号＋カナ＋ひらがな＋カタカナ＋漢字...) が定められています．

JIS 8 ビットコードは，ASCII コードに半角カタカナを追加したものです．記号，数字，英字の部分は同じ並びになっています．

2.7 補助単位

長さや重さを書くとき，1,000m を 1km，1,000g を 1kg のように書きました．この k のような記号を補助単位と呼びます．

コンピュータの世界でよく使用される補助単位には，k(キロ=10^3)，M(メガ=10^6)，G(ギガ=10^9)，T(テラ=10^{12}) 等があります．(1,000 倍毎に補助単位があります．) パソコンのカタログに「CPU のクロックは 1GHz」,「バスクロックは 800MHz」のような記述を良く見かけますね．

記憶容量を表す場合の補助単位は，$1,000$ の代わりに $2^{10} = 1,024$ を使用します．つまり，$1kB = 2^{10}B = 1,024B$，$1MB = 2^{20}B = 1,048,576B$ となります．(「B」はバイトを表す．)

「キャシュメモリの容量は 512kB」,「メインメモリの容量は 512MB」,「H.D.D. の容量は 180GB」のように表示されている場合は記憶容量を表しているので，$k = 2^{10}$ で表示されています．

補助単位まとめ

通常の値	記号	読み方	記憶容量の値
10^3	k	キロ	2^{10}
10^6	M	メガ	2^{20}
10^9	G	ギガ	2^{30}
10^{12}	T	テラ	2^{40}

2.8　コンピュータの基本回路

前の節までで，コンピュータ内部での情報の表現方法を勉強しました．コンピュータの内部では，どんな情報も電気の「ON/OFF」で表現しているのでしたね．

この節では，電気の「ON/OFF」を使用して，計算や記憶をする回路を勉強します．

2.8.1　論理演算と論理回路

論理 (Yes/No，真/偽，True/False) を対象とする演算 (計算) のことを 論理演算 と呼びます．論理の「真/偽」をビットの「1/0」に対応させることにより，論理も電気の「ON/OFF」で表現することができます．

ここでは，論理演算と，論理演算をする回路を紹介します．論理演算をする回路のことを 論理回路 と呼びます．ここで紹介する論理回路が，コンピュータを構成する基本回路になります．

2.8.2　基本的な論理回路

基本的な論理回路を 4 種類，組合せてできたものを 2 種類，紹介します．

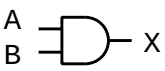
1. 論理積 (AND)
   二つのビットを入力し，両方が 1 のときだけ出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

ANDの真理値表

$X = A \cdot B$

ANDの論理式



ANDの回路記号

2. 論理和 (OR)
   二つのビットを入力し，どちらかが 1 のとき出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

ORの真理値表

$X = A + B$

ORの論理式

ORの回路記号

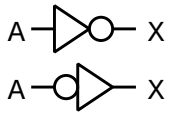
3. 否定 (NOT)
   一つのビットを入力し，入力と逆の論理を出力する論理演算です．

入力	出力
A	X
0	1
1	0

NOTの真理値表

$X = \overline{A}$

NOTの論理式



NOTの回路記号

4. 排他的論理和 (XOR)
   二つのビットを入力し，二つが異なるとき出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

XORの真理値表

$X = A \oplus B$

XORの論理式

XORの回路記号

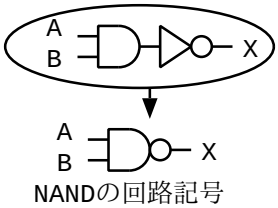
5. NAND
   否定 (NOT) と論理積 (AND) を組み合わせた演算です．

入力		出力
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

NANDの真理値表

$X = \overline{A \cdot B}$

NANDの論理式



NANDの回路記号

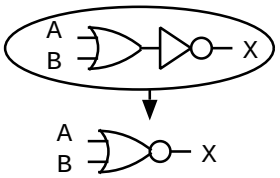
6. NOR
否定 (NOT) と論理和 (OR) を組み合わせた演算です.

入力		出力
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

NORの真理値表

$$X = \overline{A + B}$$

NORの論理式



NORの回路記号

2.8.3 演算回路

「基本的な論理回路」を組み合わせることにより, 足し算／引き算等のより複雑な演算を行う回路を実現できます.

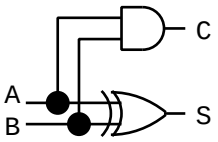
1. 半加算器
例えば, 1 ビットの足し算回路は次の例のように実現できます. この例では A, B の 2 ビットを入力し, 和 (S) と上の桁への桁上がり (C) を出力する回路を示しています. このような回路のことを「半加算器」と呼びます.

例 1 ビット半加算器

```
  A     0     1     0     1
+ B   + 0   + 0   + 1   + 1
-----------------------------
C S   0 0   0 1   0 1   1 0
```

入力		出力	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

半加算器の真理値表



半加算器の回路図

2. 全加算器
半加算器は桁上がりを出力しますが, 自分自身は下の桁からの桁上がりを入力することができません. 桁上がりの入力を備えた 1 ビット足し算器を「全加算器」と呼びます.

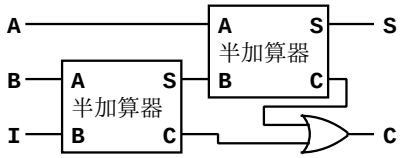
次の例のように, 全加算器は半加算器を組み合わせて実現することができます.

例 1 ビット全加算器

入力			出力	
A	B	I	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A:計算の入力
B:計算の入力
I:桁上がりの入力

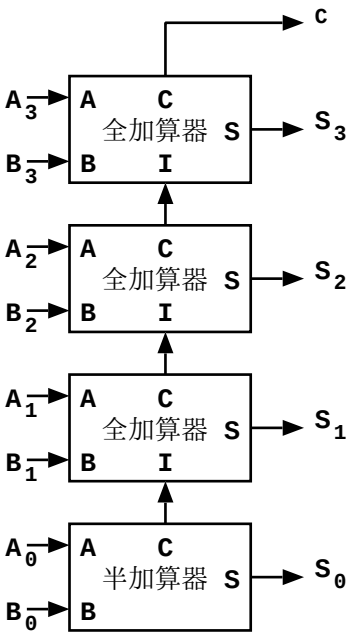
全加算器の真理値表



全加算器の回路図

3. n ビット加算器
半加算器と全加算器を組み合わせることで, 任意ビットの加算器を実現することができます. 次に 4 ビット加算器の例を示します.

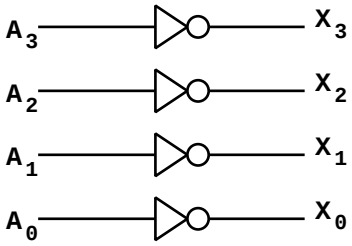
例 4 ビット加算器



4ビット加算器の回路図

4. n ビット 1 の補数器
   1 の補数を作る回路です．1 の補数はビット反転によってできるので，NOT でできます．次に，4 ビットの例を示します．簡単ですね．

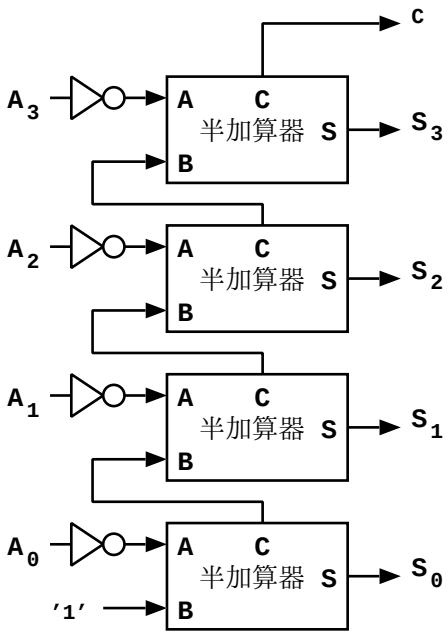
例 4 ビット 1 の補数器



4ビット1の補数器の回路図

5. n ビット 2 の補数器
   2 の補数を作る回路です．2 の補数は，1 の補数に 1 足すとできるので，1 の補数器の出力に半加算器を組み合わせるとできます．

例 4 ビット 2 の補数器



4ビット2の補数器の回路図

演算回路を数種類紹介しました．ここに紹介しませんでしたが，引き算回路等も同様に製作可能です．

2.8.4　記憶回路

「基本的な論理回路」を組み合わせることにより，記憶機能を実現することもできます．ここでは，最も簡単な RS-FF(RS フリップフロップ) を紹介します．

RS-FF は S(Set) と R(Reset) の二つの入力と，状態 $(Q, \bar{Q})$ の出力を持つ回路です．S=0,R=1 を入力することにより，回路はリセットされ出力 Q は 0 になります．S=1,R=0 を入力することにより，回路はセットされ出力 Q は 1 になります．$\bar{Q}$ には，常に Q の否定が出力されます．

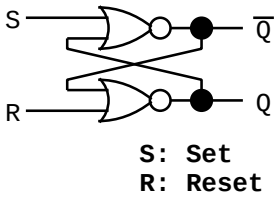
S=0,R=0 を入力すると，回路は直前の値を「記憶」します．S=1,R=1 は入力してはいけない組合せです．

RS-FF は，回路をリセット／セットした後で「記憶」状態にすることにより,「値を記憶する回路 (＝記憶回路)」として働きます．

RS-FF

入力		出力
S	R	Q
0	0	記憶
0	1	0
1	0	1
1	1	禁止

RS-FFの動作



RS-FFの回路

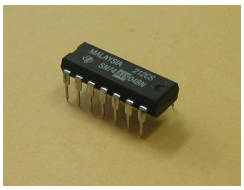
論理回路素子

回路を製作するには，基本的な論理回路を内蔵した集積回路 (論理 IC) を用いることができます．

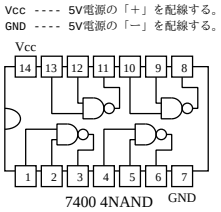
様々な論理 IC が販売されていますが，ここでは，74 シリーズのものを紹介します．下の例に示したのは，NAND ゲートを四つ内蔵した 7400 と呼ばれる IC です．同様な IC で，AND，OR，XOR，NOT 等の回路を内蔵したものがあります．

このような IC 同士を配線して組み合わせることにより，本文で紹介した演算回路や記憶回路を実現することができます．

論理 IC



外観



論理ICの内部

第3章　マイコンの組み立て

この章では，TeC の組み立て方法を説明します．ハンダ付けの練習も含めて，5 日で組み立てます．何年間も使用するマイコンなので，慎重に組み立てて下さい．

3.1　ハンダ付け練習

マイコン本体を組み立てる前に，ハンダ付けの練習をします．練習は，キットに含まれる小さな基板 (図 3.1) を使用して行います．この基板には，マイコンに 1 秒毎に信号を入力するための回路を搭載します．

ハンダ付けの練習専用ではなく，将来，キットの機能の一部になりますので，手抜きをしないでしっかり組み立てて下さい．

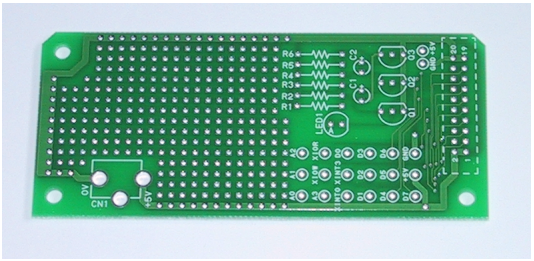


図 3.1: ハンダ付け練習用基板

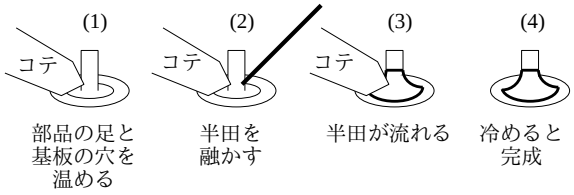


図 3.2: ハンダ付け手順

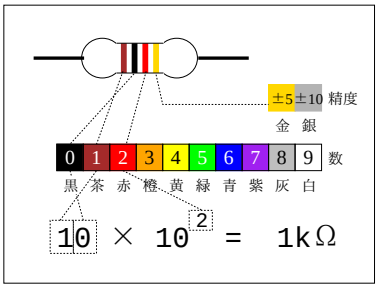
表 3.1: ハンダ付け練習に使用する部品

記号	名前	説明
R1,R3	抵抗器	1k Ω (茶黒赤金)
R2	抵抗器	10k Ω (茶黒橙金)
R6	抵抗器	4.7k Ω (黄紫赤金)
R4,R5	抵抗器	100k Ω (茶黒黄金)
C1,C2	コンデンサ	$10\mu F$
Q1,Q2,Q3	トランジスタ	電流を増幅する部品
LED1	ランプ	LED
CN2	コネクタ	マイコン本体と接続

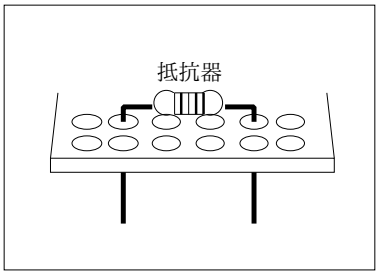
3.1.1　部品の取り付け

表 3.1 の部品を基板に取り付けます．ハンダ付けの要領を図 3.2 に示します．上手なハンダ付けができると，ハンダが富士山型になります．基板上の表示を良く確かめて部品の取り付け位置を確認し，以下の順序で部品をハンダ付けしてください．

1. 抵抗器
   図 3.3 のような外観です．抵抗値は，部品のカラーコードで見分けます．正しい抵抗値の抵抗器を選んで，足を曲げて基板に差し込んで下さい．浮き上がりの無いように，しっかり差し込んで下さい．



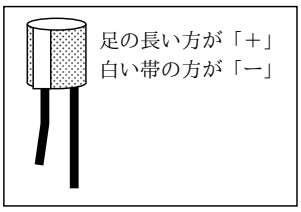
(1) カラーコードの読み方



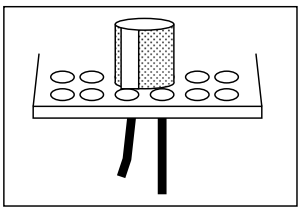
(2) 実装方法

図 3.3: 抵抗器

2. コンデンサ
   コンデンサは図 3.4 ような外観の部品です．コンデンサには極性があります．基板の上にある「＋」のマークと，コンデンサの極性を一致させて実装してください．



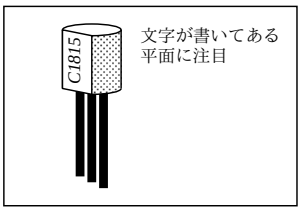
(1)極性の見分けかた



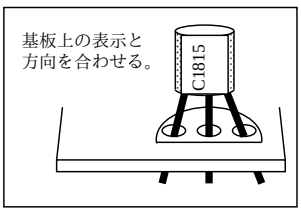
(2)実装方法

図 3.4: コンデンサ

3. トランジスタ
   トランジスタは図 3.5 ような外観の部品です．トランジスタにも方向があります．基板の表示とトランジスタを上から見た形が，同じになるように実装してください．足を折らないように気をつけながら，なるべく低く実装してください．



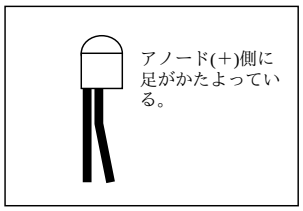
(1)極性の見分けかた



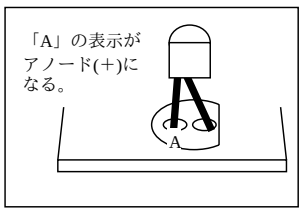
(2)実装方法

図 3.5: トランジスタ

4. ランプ
   ランプには，LED と呼ばれる部品を使用します．LED にも極性があります．図 3.6 のように，方向を間違えないように実装してください．足を折らないように気をつけながら，なるべく低く実装してください．



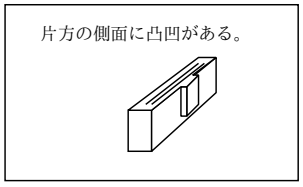
(1)極性の見分けかた



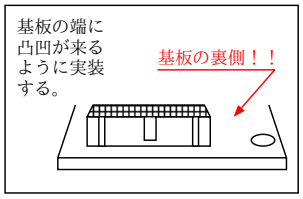
(2)実装方法

図 3.6: ランプ (LED)

5. コネクタ
   マイコン本体と，この基板を接続するためのコネクタです．図 3.7 のように，方向を間違えないように実装してください．この部品だけは基板の裏側 に実装しますので，間違えないように注意してください．



(1)方向の見分けかた



(2)実装方法

図 3.7: コネクタ

3.1.2 動作テスト

完成品のマイコンに接続して動作テストをします．1 秒程度の周期でランプが点滅を繰り返せば OK です．

3.2 組み立て 1 日目

本体の組み立てを 4 日に分けて行います．組み立ては，基板に部品をハンダ付けする作業です．ハンダ付けする場所は，基板の上に白い文字で書いてある記号で判断します．ハンダ付けの順序は，次のような基準で決めます．

1. 背の低い部品
2. 基板中心の部品
3. 壊れにくい部品

初日は，(1) 抵抗器，(2)IC(IC ソケット)，(3) 集合抵抗器を基板にハンダ付けして取り付けます．

3.2.1 準備

ハンダ付けする時，基板が安定するように基板に仮の足を付けます．図 3.8 のように，本体基板の四隅に白色のスペーサを取り付けて下さい．

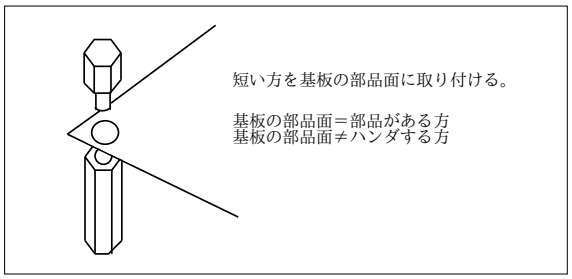


図 3.8: 仮の足の取り付け

3.2.2 抵抗器の取り付け

表 3.2 の抵抗器を基板にハンダ付けします．抵抗器には向きはありませんが，カラーコードの読みやすさを考えて方向を統一する方が良いでしょう．

表 3.2: 抵抗器

記号	カラーコード	説明
R1	赤黒茶金	470 Ω 1/4W カーボン
R2	赤黄黒黒茶	240 Ω 1/4W 金属皮膜
R3	橙白黒黒茶	390 Ω 1/4W 金属皮膜
R4	赤黄黒黒茶	240 Ω 1/4W 金属皮膜
R5	赤黄黒黒茶	240 Ω 1/4W 金属皮膜
R6	茶黒橙金	10k Ω 1/4W カーボン

3.2.3 IC(ソケット) の取り付け

表 3.3 の部品を基板にハンダ付けします．IC には向きがありますので，間違えないように注意して下さい．図 3.9 の説明を良く読んで，**「正しい部品を正しい方向で」**取り付けて下さい．この時，図 3.10 のように**「IC の足を修正して」**基板の穴に差し込んで下さい．

部品が傾いたり，浮き上がったりしないようにするには，IC の足 2ヵ所程度を**「仮にハンダ付けし，問題が無いか確認して」**から他の足をハンダ付けすると良いでしょう．足の多い部品を間違ってハンダ付けしてしまうと，取り外すのが絶望的に大変ですので，くれぐれも間違えないように注意して下さい．

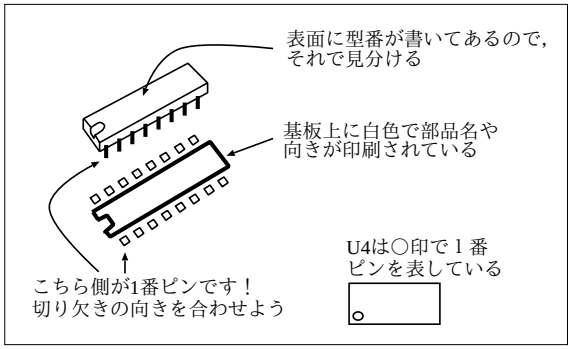


図 3.9: IC 取付の説明図

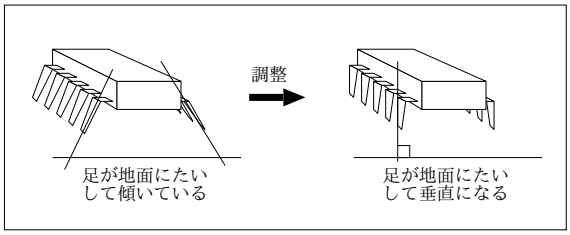


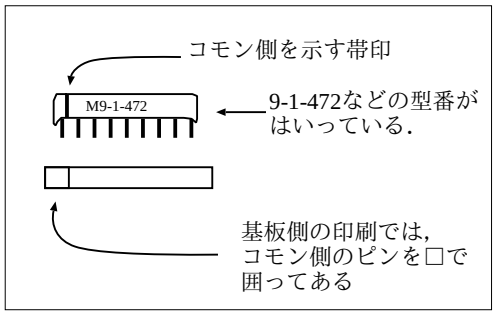
図 3.10: IC 足の調整方法

表 3.3: IC(ソケット)

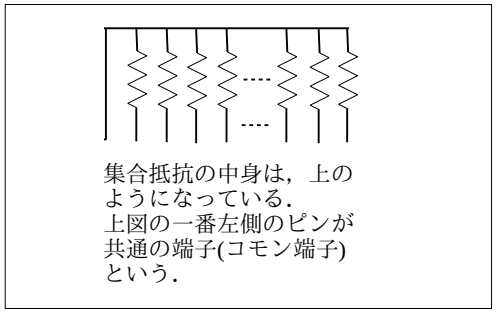
記号	型番	説明
U4	TL7705	電源監視 IC
U5	なし	IC ソケット (後で IC を差し込む)
U6	MAX232N	パソコンとの通信用 IC
U7	74ALS540	74 シリーズ論理 IC
U8	74ALS540	74 シリーズ論理 IC
U9	74ALS04	74 シリーズ論理 IC
U10	74ALS138	74 シリーズ論理 IC

3.2.4　集合抵抗器の取り付け

図 3.11 の集合抵抗器を取り付けます．この部品にも向きがありますので，注意して下さい．IC と同じ要領で，傾きや浮き上がりが無いよう注意深くハンダ付けしましょう．



(1)方向の見分け方



(2)内部の構造

図 3.11: 集合抵抗器

表 3.4: 集合抵抗器

記号	型番	説明
RA1	M5-1-332	3.3k Ω (4 素子)
RA2	M5-1-472	4.7k Ω (4 素子)
RA3	M9-1-472	4.7k Ω (8 素子)
RA4	M9-1-472	4.7k Ω (8 素子)
RA5	M5-1-332	3.3k Ω (4 素子)
RA6	M5-1-332	3.3k Ω (4 素子)
RA7	M5-1-201	470 Ω (4 素子)
RA8	M9-1-201	470 Ω (8 素子)
RA9	M9-1-201	470 Ω (8 素子)
RA10	M9-1-201	470 Ω (8 素子)

3.3　組み立て 2 日目

2 日目の作業を説明します．1 日目と同様に，基板の四隅に白色スペーサを取り付けて作業をします．

3.3.1　積層セラミック・コンデンサの取り付け

表 3.5 の部品をハンダ付けします．数がとても多い (15 個) のですが，全て同じ部品です．向きはありませんが，完成したとき表面の文字が読みやすい方向に取り付けると良いでしょう．図 3.12 を良く見て，部品を間違えないようにしてください．

表 3.5: 積層セラミック・コンデンサ

記号	型番	説明
C3,C6,C9,C11,C16〜C26	104	$0.1\mu F$

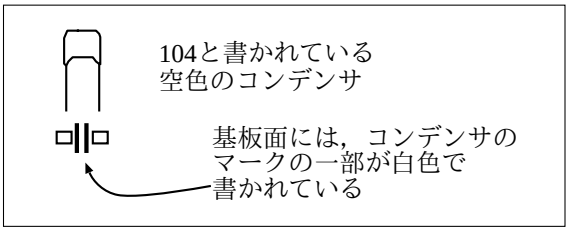


図 3.12: 積層セラミック・コンデンサ

3.3.2 タンタル・コンデンサの取り付け

表 3.6 に書いてある 7 個の部品をハンダ付けします．今度の部品は向きがありますので，気をつけて下さい．図 3.13 に説明してあるように，長い方の足が「＋」です．

表 3.6: タンタル・コンデンサ

記号	型番	説明
C1,C4,C7,C12〜C15	1/25	$1\mu F$

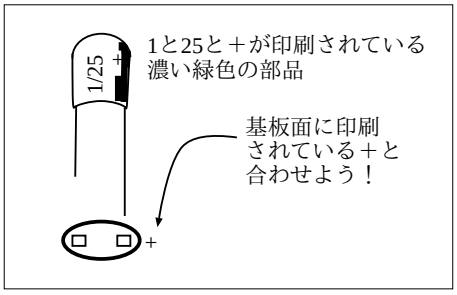


図 3.13: タンタル・コンデンサ

3.3.3 水晶発振器の取り付け

表 3.7 の部品をハンダ付けします (1 個だけです)．今度の部品も向きがありますので，気をつけて下さい．図 3.14 に説明してあるように，基板の印刷と同じ向きにします．

表 3.7: 水晶発振器

記号	型番	説明
X1	2.457KSS 8F;JXO-5S	2.4576MHz

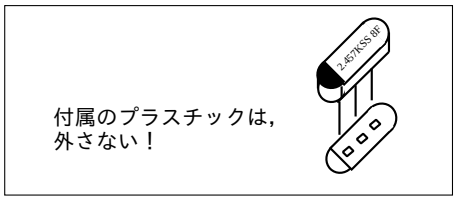


図 3.14: 水晶発振器

3.3.4 ジャンパの取り付け

表 3.8 の部品をハンダ付けします (1 個だけです)．この部品は二つの部分からできています．小さくて壊れやすいので，組み立てた状態でハンダ付けしてください．図 3.15 のように，上下を間違えないように組み立てて下さい．

表 3.8: ジャンパ

記号	型番	説明
J1	なし	ジャンパー

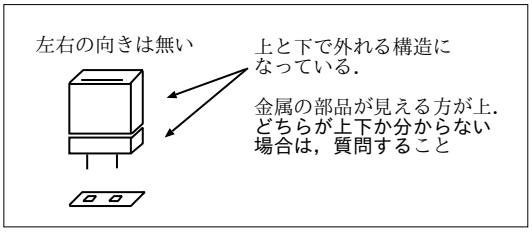


図 3.15: ジャンパ

3.3.5 電源 IC の取り付け

表 3.9 の部品をハンダ付けします．図 3.16 を良く見て，向きを間違えないように取り付けて下さい．

表 3.9: 電源 IC

記号	型番	説明
U2,U3	LM317	安定化電源を作る IC

図 3.16: 電源 IC

3.3.6　スピーカの取り付け

表 3.10 の部品をハンダ付けします．向きがあります．長い足が「＋」です．図 3.17 を良く見て，向きを間違えないように取り付けて下さい．

表 3.10: スピーカ

記号	型番	説明
BZ1	T3a3	スピーカ

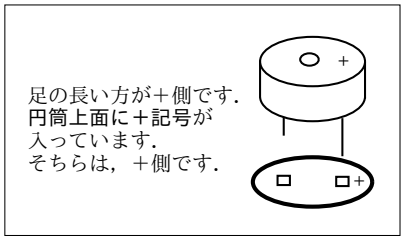


図 3.17: スピーカ

3.3.7　コネクタの取り付け

表 3.11 の部品をハンダ付けします．この部品にも向きがあります．図 3.18 を良く見て，向きを間違えないように取り付けて下さい．

表 3.11: コネクタ

記号	型番	説明
CN2	なし	小さい 10 ピンのコネクタ
CN3	なし	大きい 20 ピンのコネクタ

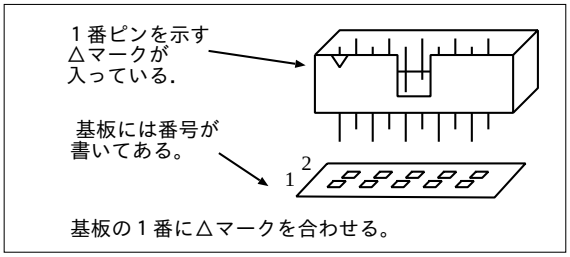


図 3.18: コネクタ

3.4　組み立て 3 日目

3 日目の作業を説明します．今日から背の高い部品が多くなりますので，基板の四隅に白色スペーサをこれまでとは**上下逆に**取り付けて下さい．(図 3.19)

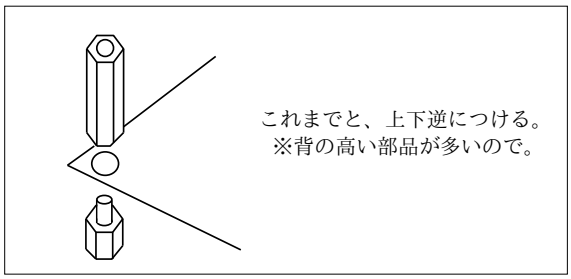


図 3.19: 仮の足を上下逆につける

3.4.1　電解コンデンサの取り付け

表 3.12 の部品をハンダ付けします．部品の形状は図 3.20 のようなものです．大きなものが三つ，小さなものが一つあります．この部品も，向きがありますので取り付け方向に注意が必要です．長い足の方が「＋」になります．図 3.20 を良く見て，部品を間違えないようにしてください．

表 3.12: 電解コンデンサ

記号	型番	説明
C10	$25V10\mu F$	$10\mu F$
C2,C5,C8	$25V100\mu F$	$100\mu F$

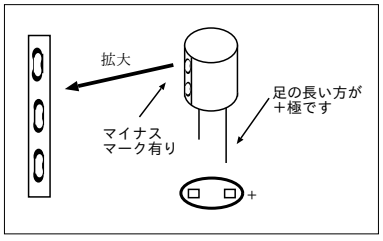


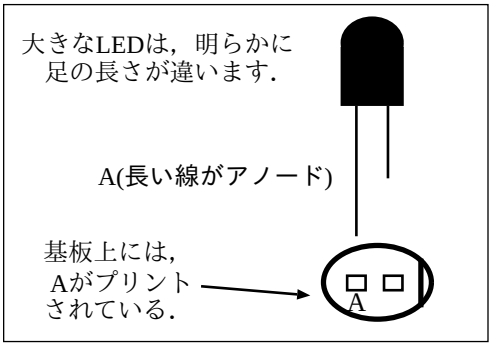
図 3.20: 電解コンデンサ

3.4.2 LED(ランプ) の取り付け

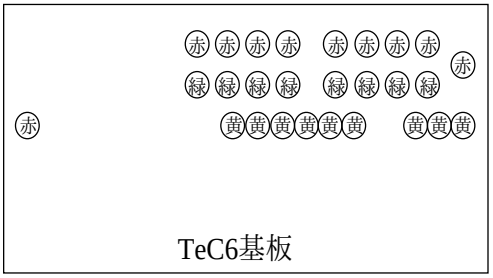
LED(ランプ) は，マイコンを使用するときいつも注視する部品です．傾いていたりすると，とても気になりますので綺麗に取り付けて下さい．LED は，赤，緑，黄色の 3 種類があります．LED をハンダ付けする要領は次の通りです．

1. 同じ色を一斉に**アノード**だけハンダ付けする．(黄→緑→赤の順に中心から取り付ける．)
2. LED が垂直になっているか確認する．(垂直になっていない場合は，再度温めて修正する．)
3. LED が奥までささっているか確認する．
4. カソードをハンダ付けする．
5. リード線を切る．

アノード (+)，カソード (−) の見分け方は図 3.21 (1) の通りです．方向を間違えないようにしてください．3 種類の LED は，図 3.21 (2) のように配置してください．



(1) 方向の見方



(2) LEDの配置

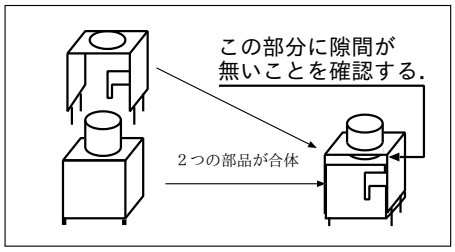
図 3.21: LED の取り付け

3.4.3 スイッチの取り付け

スイッチの種類

スイッチには，**トグルスイッチ**と**プッシュスイッチ**の 2 種類があります．トグルスイッチは上下に倒すタイプのスイッチ，プッシュスイッチは押しボタンになったスイッチです (図 3.22 (1) 参照)．どちらのスイッチも二つの部品が組み合わさっています．ハンダ付けする前に部品の間にすき間が無いか確認して，もしも，すき間があるときはしっかり組み合わせて下さい (図 3.22 (2) 参照)．

(1) スイッチの種類



(2) スイッチの構造

図 3.22: スイッチ

スイッチのハンダ付け

スイッチは，マイコンを操作するときいつも触る部品です．スイッチが傾いていたりすると操作性が悪くなりますので，まっすぐになるよう慎重にハンダ付けしてください．また，スイッチは温まりにくいので，ハンダ付けするときは少し余計に熱するようにしてください．向きを間違えると，基板の穴に足が入りません．間違える心配は無いので安心してください．スイッチをハンダ付けするときの手順は以下の通りです．

1. 足を穴にしっかり差し込む．
2. ハンダ面に 7 本の足が均等に出ていることを確認する．
3. 足のうち 1 本をハンダ付けする．

4. スイッチが傾いていないか確認する．(傾いていた場合は，温め直して修正する．)

5. 他の足をハンダ付けする．

2 種類のスイッチは，図 3.23 のように間違えないように配置してください．

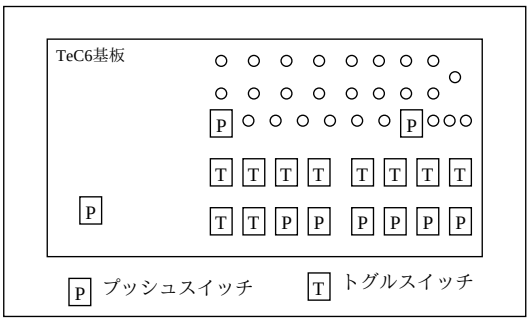


図 3.23: スイッチの配置

3.4.4　電源コネクタの取り付け

電源コネクタ (図 3.24 (1)) は，基板の左上の大きな三つの穴 (図 3.24 (2)) にハンダ付けします．このコネクタの取り付けでは，失敗するとひどい火傷をする恐れがあります．以下の手順を良く読んで慎重に作業してください．

1. 穴にしっかり差し込む．

2. 穴とコネクタの端子をハンダゴテで十分熱する．

3. 穴が塞がるまで，ハンダをどんどん融かし込む．

4. **十分に冷めるのを待つ．**

5. 部品が傾いていないか確認する．

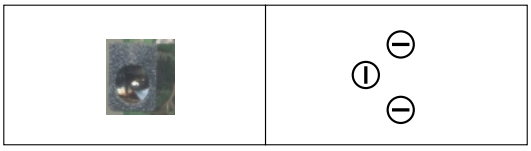


図 3.24: 電源コネクタ

3.5　組み立て 4 日目

4 日目の作業を説明します．今日でマイコンが (やっと ？) 完成します．今日も背の高い部品が多くなりますので，基板の四隅に白色スペーサを**上下逆** (3 日目と同じ方向) に取り付けて下さい．(図 3.19)

3.5.1　D-SUB コネクタの取り付け

パソコンと通信するために使用する D-SUB コネクタを取り付けます．この部品は，ねじ止めした状態でハンダ付けします．手順は次の通りです．

1. 穴にしっかり差し込む．

2. ワッシャをはさみ，ナットでねじ止めする．(図 3.25 参照)

3. 工具でナットをしっかりしめつける．

4. ピンをハンダ付けする．

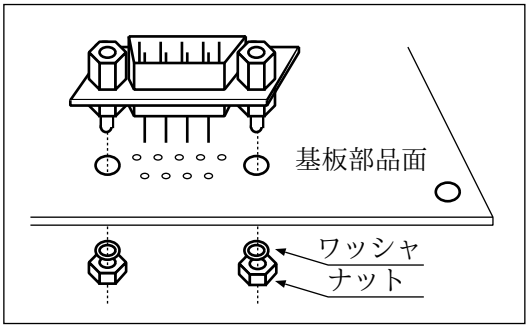


図 3.25: D-SUB コネクタ

3.5.2 ゴム足の取り付け

基板に足をつけます．足は，両面テープの付いた黒い円盤状の部品です．足を取り付ける場所は，図 3.26 の通りです．まず，アルコールで周辺をきれいに拭いてから，鉄板とシールを剥がして，両面テープで指定の場所に貼り付けて下さい．

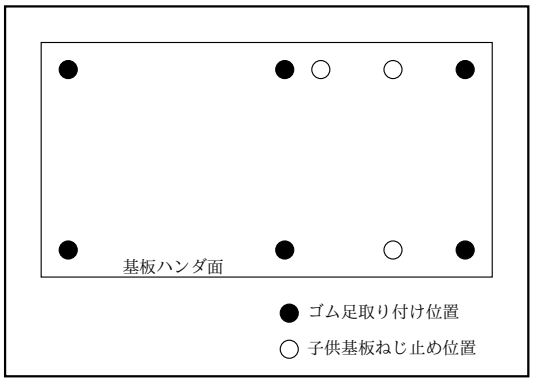


図 3.26: ゴム足の取り付け位置

3.5.3 ROM の取り付け

マイコンの設計データが格納された ROM を取り付けます．万一，設計にバグが見付かったときに備え，ROM だけは交換可能になっています．ROM も取り付け方向が決まっている部品です．図 3.27 のように IC ソケットの切り欠きと，ROM の切り欠きが同じ方向になるように，基板中央の IC ソケットに取り付けて下さい．

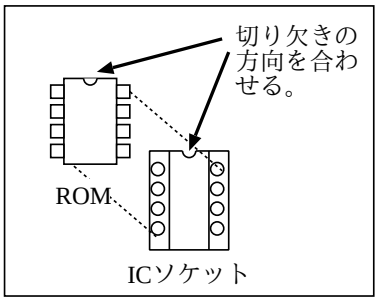


図 3.27: ROM の取り付け方向

3.5.4 プッシュスイッチの頭の取り付け

最後に，プッシュスイッチに頭を取り付けます．図 3.28 を良く見て取り付けて下さい．

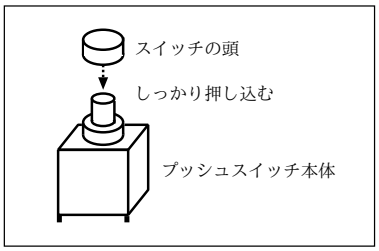


図 3.28: プッシュスイッチの頭の取り付け方

3.5.5 命令表の貼り付け

ケースのフタ (外側に組み合わされる方です．) の裏に，命令表を貼り付けて下さい．

3.6 完成

ご苦労さまでした．これでマイコンが完成しました．動作確認をします．次章「マイコンの操作」に従い，操作できるか試して下さい．

第4章　マイコンの操作

この章では，マイコンの操作方法を学習します．併せてマイコンの内部にある，レジスタや記憶装置も憶えましょう．

4.1　各部の名称

図 4.1 に，マイコン各部の名称 (または役割り) を示します．各部の役割りは次の通りです．

1. コンソールパネル
   マイコンにプログラムを入力したり，マイコンの内部を観察したり，プログラムの実行を指示したりするために使用する部分です．データの入力や表示は 2 進数を用います．本マイコンの顔です．
2. マイコン本体
   正方形の LSI に，マイコンのほとんどの機能が入っています．
3. ランプ駆動回路
   マイコン本体は，ランプ (LED) に流すのに十分な電流を扱うことができません．駆動回路はマイコン本体の出した信号を増幅して，ランプを光らせます．
4. 電源回路
   電源コネクタから供給される DC 5V から，マイコン本体が必要とする DC 3.3V と DC 2.5V の電源を作ります．電源コネクタには付属の AC アダプタを接続します．付属の AC アダプタ以外を接続しないで下さい．
5. 通信機能
   パソコンと通信するための機能を提供します．マイコンとパソコンの RS-232C コネクタ同士を，クロスケーブルで接続すると通信できるようになります．
6. リセットスイッチ
   マイコンをリセットするための押しボタンスイッチです．
7. スピーカー
   コンソールパネルを操作したとき，操作を確認するための音を出します．また，マイコンに入力したプログラムで音を出すこともできます．
8. 拡張コネクタ
   マイコンの外部に追加の回路を拡張するために使用します．
9. JTAG コネクタ
   マイコン本体の回路を変更するとき使用します．

4.2　コンソールパネル

図 4.2 にコンソールパネルの様子を示します．この図を見ながら以下を読んで下さい．

4.2.1　コンソールパネルの構成

コンソールパネルは，以下のランプやスイッチにより構成されています．コンソールパネルが，マイコンボードの約半分の面積を使用していますが，これは，操作性を考慮した結果です．これ以上，小さくするとスイッチの間隔が狭くなり過ぎ，操作性が悪くなります．

1. アドレスランプ
   最上段の 8 個の赤ランプはアドレスランプです．アドレスランプは，主記憶を操作するとき，操作対象となる主記憶の番地を表示します．
2. データランプ
   上から 2 段目の 8 個の緑ランプは，選択されたレジスタまたは主記憶の内容を表示します．
3. ロータリースイッチ
   上から 3 段目の 6 個の黄色ランプと左右の押しボタンスイッチは，ロータリースイッチの代用です．通常のロータリースイッチは背が高く，ビデオカセットケースにマイコンを収めるための障害になるのでこのようにしました．

   ランプのうち一つが点灯し，レジスタ (G0，G1，G2，SP，PC) または主記憶 (MM) のどれが選

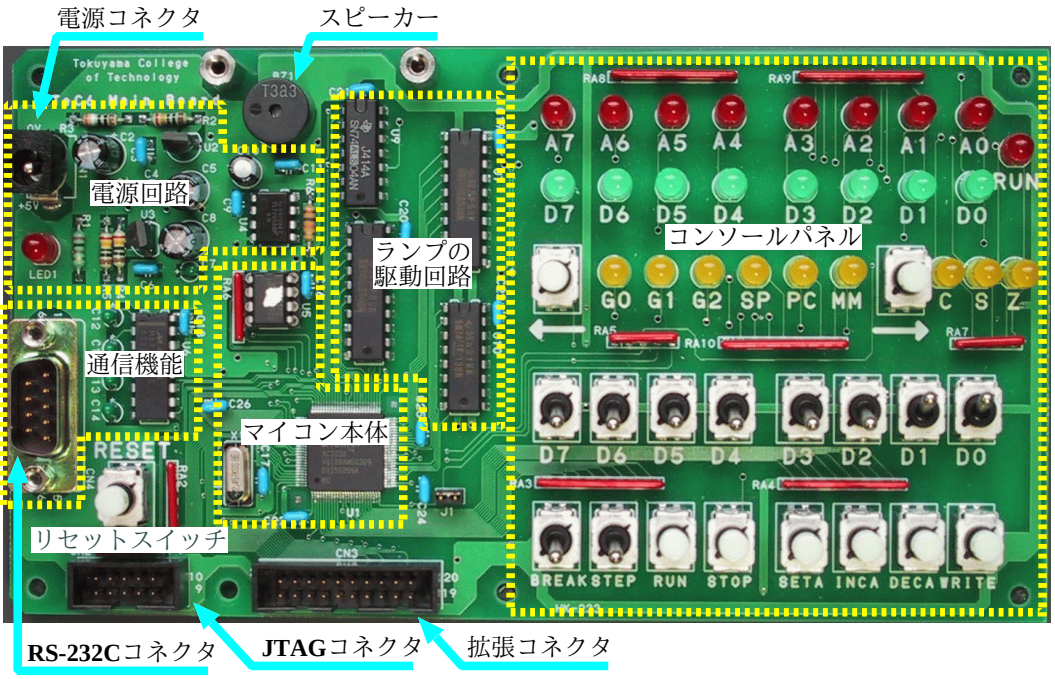


図 4.1: 各部の名称

択されているか表示します．左右の押しボタンスイッチ (←，→) によって，点灯するランプを変更することができます．ロータリースイッチで選択されたものの内容が，データランプに表示されます．

4. データスイッチ
   8 個のトグルスイッチは，データやアドレスの値を入力するためのものです．

5. 機能スイッチ
   一番下の列の 8 個のスイッチには，様々な機能が割り当てられています．左から順に，**ブレークポイントモード** (BREAK)，**ステップ実行モード** (STEP)，**プログラム実行** (RUN)，**プログラム停止** (STOP)，**アドレスセット** (SETA)，**アドレスを進める** (INCA)，**アドレスを戻す** (DECA)，**書き込み** (WRITE) の機能を持ちます．押しボタンスイッチは，長押しでリピート機能が働きます．

6. 実行ランプ
   右上の RUN ランプは，CPU がプログラムを実行中であることを表します．CPU が正しくない命令を実行した場合は点滅して異常を知らせます．

7. フラグランプ
   実行ランプ下の三つの黄色ランプ C(Carry)，S(Sign)，Z(Zero) は，フラグの値を表示します．

4.2.2　コンソールパネルの操作方法

コンソールパネルから，プログラムを実行させたり実行を途中で止めたりすることができます．また，図 4.3 に示す TeC 内部の情報をアクセスすることができます．TeC の内部には 3 ビットのフラグ，8 ビットのレジスタが 5 個，8 ビット× 256 のメモリが内蔵されています．

コンソールパネルのランプが，レジスタやフラグの値を表示します．また，データを入力するときはトグルスイッチで値を表現します．ランプは点灯している状態が 2 進数の'1'，消灯している状態が 2 進数の'0' を表現しています．また，トグルスイッチ (上下に倒すスイッチ) は，上に倒した状態が 2 進数の'1'，下に倒した状態が 2 進数の'0' に対応します．ここでは，コンソールパネルの操作方法を目的別に説明します．

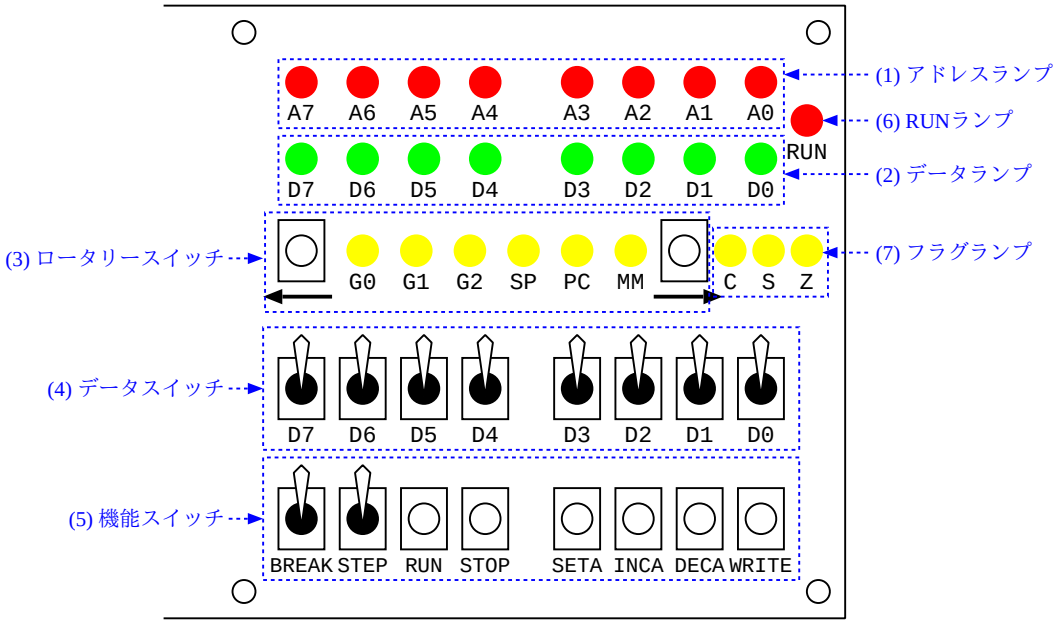


図 4.2: コンソールパネル

フラグの表示

フラグの状態は C，S，Z の三つのランプに常時表示されています．フラグの値を表示するために何も操作をする必要はありません．フラグの値をコンソールパネルから変更する方法は用意されていません．

レジスタの表示と書き換え

五つのレジスタ (G0，G1，G2，SP，PC) を表示する方法と，書き換える方法を説明します．レジスタの値は，緑色のデータランプに表示されます．8 個のランプで 8 ビットの情報を表現します．どのレジスタをデータランプに表示するかは，ロータリースイッチで選択します．

レジスタの値を書き換える手順は，次の通りです．

1. ロータリースイッチを操作し，書き換えたいレジスタの値が表示される状態にする．
2. 書き込むデータをデータスイッチにセットする．
3. **WRITE スイッチ**を押す．

主記憶 (メモリ) の表示と書き換え

図 4.3 を見ると分かるように，メモリには 00_{16} ～ FF_{16} (2 進数で表現すると 0000 0000$_2$ ～ 1111 1111$_2$) の番地 (アドレス) が付けられています．メモリへ対する表示や書き換え等の操作は，1 アドレス (8 ビット) 毎に行いますので，操作対象となるメモリのアドレスを指定した上で行う必要があります．メモリ・アドレスは，**アドレスランプ**にセットします．アドレスランプにアドレスをセットする方法は次の通りです．

1. ロータリースイッチを MM(Main Memory) に合わせる．
2. アドレスをデータスイッチにセットする．
3. **SETA(Set Address) スイッチ**を押す．
   (データスイッチにセットした値がアドレスランプに表示される．)
4. アドレスを次の番地に進めたいときは，**INCA(Increment Address) スイッチ**を押す．
5. アドレスを前の番地に戻したいときは，**DECA(Decrement Address) スイッチ**を押す．

フラグ		計算結果を表す目印
レジスタ	G0,G1,G2	計算対象となるデータや 計算結果を一時的に置く場所
	SP	難しいので省略
	PC	実行するプログラムのアドレス
主記憶		プログラムや変数を置く場所

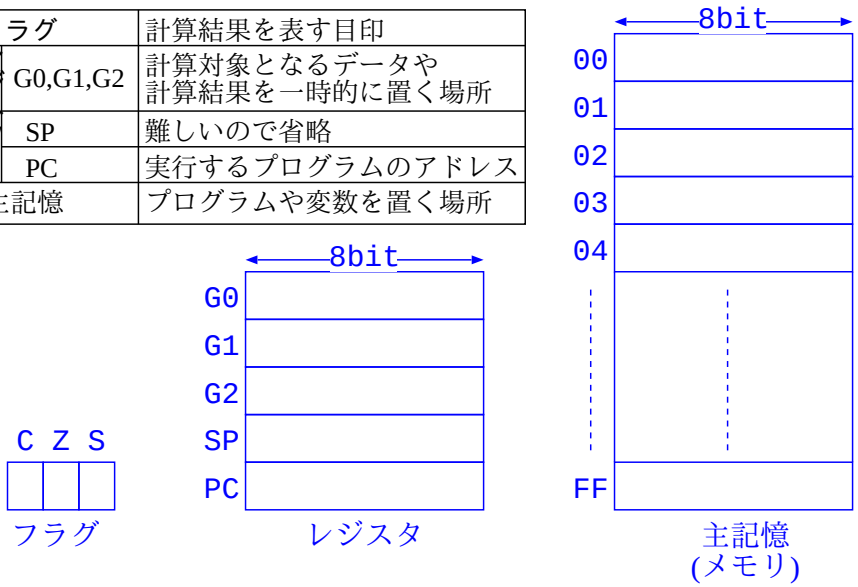


図 4.3: TeC の内部

アドレスの指定方法が分かったところで，メモリ内容を表示する手順を説明します．

1. ロータリースイッチを MM(Main Memory) に合わせる．
2. 操作の対象となるアドレスをアドレスランプで指定する．
3. データランプに該当メモリ番地の内容が表示される．
4. 別の番地の内容を表示したいときは，SETA，INCA，DECA 等のスイッチで，アドレスランプの表示を変更する．

次に，メモリ内容を書き換える手順を説明します．

1. 上の手順に従い，書き換えたいメモリ番地の内容がデータランプに表示された状態にする．
2. データスイッチに書き込みたい値をセットする．
3. **WRITE スイッチ**を押す．
4. メモリにデータが書き込まれる．

なお，**WRITE スイッチ**を押したとき，アドレスが自動的に次の番地に変化します．これは，連続したデータの書き込みに便利だからです．

プログラムの実行

ノイマン型のコンピュータは，**「プログラム内蔵方式 (ストアードプログラム方式)」**を採用しているのが特徴です．「プログラム内蔵方式」とは，データだけでなくプログラムもメモリに記憶する方式です．TeC もプログラム内蔵方式ですので，メモリにデータのようにプログラムを書き込んで実行します．つまり，コンソールパネルからデータを書き込む方法を知っていれば，プログラムを書き込むこともできます．

ノイマン型コンピュータのもう一つの特徴として，**「逐次実行」**があげられます．「逐次実行」とは，メモリに記憶したプログラムの命令をアドレス順に一つ一つ順番に実行することを言います．次に実行する命令のアドレスは，PC レジスタが記憶しています．

TeC でプログラムを実行する手順は，次の通りです．

1. プログラムをメモリに書き込む．
2. プログラムの開始番地を PC に書き込む．
3. **BREAK，STEP スイッチ**を下に倒す．
4. **RUN スイッチ**を押す．(RUN ランプ点灯)
5. プログラムの実行終了．(RUN ランプ消灯)

次に，とても短いプログラムの打ち込みと実行の例を示します．

例 最も簡単なプログラムの実行

下は，何もしない命令 (NO 命令) を三つ実行した後，停止命令 (HALT 命令) を実行して停止するプログラムです．このプログラムを打ち込んで実行します．

最も簡単なプログラムのリスト

番地	命令	命令の種類
00_{16}	00_{16}	NO
01_{16}	00_{16}	NO
02_{16}	00_{16}	NO
03_{16}	FF_{16}	HALT

プログラム打ち込み手順

1. ロータリースイッチを MM に合わせる．
2. アドレスランプに 00_{16} をセットする．
3. データスイッチに 00_{16} をセットする．
4. WRITE スイッチを 3 回押し，メモリ 00_{16} 番地〜02_{16} 番地の 3 番地に 00_{16} を書き込む．
5. データスイッチに FF_{16} をセットする．
6. WRITE スイッチを押し，メモリ 03_{16} 番地に FF_{16} を書き込む．

プログラム実行手順

1. ロータリースイッチを PC に合わせる．
2. データスイッチに 00_{16} をセットする．
3. WRITE スイッチを押し，PC を 00_{16} にする．
4. BREAK，STEP スイッチを下に倒す．
5. RUN スイッチを押し実行を開始する．
6. 一瞬でプログラム実行が終了する．
7. PC は，HALT 命令実行後なので，その次の番地 04_{16} を指している．
8. 何もしない命令と，停止命令しか実行していないので，PC 以外のレジスタ，フラグ，メモリに変化はない．

プログラムのステップ実行

プログラム中の命令を，1 命令ずつ実行することを**ステップ実行**と言います．作ったプログラムが予定通りに動かないとき，原因を調べるために使用します．(このような作業を**デバッグ**と言います．) ステップ実行の手順は次の通りです．

1. プログラムをメモリに書き込む．
2. プログラムの開始番地を PC レジスタに書き込む．
3. **STEP スイッチ**が上に倒れていることを確認する．(倒れていなかった場合は倒す．)
4. **RUN スイッチ**を押す．
5. PC にセットしてあった番地の 1 命令が実行され，プログラムが停止する．
6. レジスタやメモリの状態が予想通りか確認する．
7. 確認が終わったら RUN スイッチを押して，次の命令を実行させる．
8. プログラムの終わりまで以上の操作を繰り返す．

次に，ステップ実行の例を示します．

例 プログラムのステップ実行

前の例と同じプログラムを，ステップ実行モードで実行してみます．

プログラムの実行手順

1. 前の例と同様に，プログラムを打ち込む．
2. STEP スイッチを上に倒して (ステップ実行モードにして)，前の例と同様にプログラムの実行を開始する．
3. ステップ実行モードなので，00_{16} 番地の命令だけを実行し停止する．このとき，PC は次命令の番地 01_{16} を指している．
4. 再度 RUN スイッチを押すと，01_{16} 番地の命令を実行し，PC が 02_{16} を指して停止する．
5. 以後，RUN スイッチを押す毎に，1 命令実行しては停止する．

ブレークポイントを使用した実行

デバッグをするときに目的の命令まで一気に進んでから，ステップ実行をしたい場合があります．そのとき，一気に目的の命令まで進むのにブレークポイントを使用できます．ブレークポイントとは，プログラムの実行を停止する場所 (アドレス) のことです．次のような手順になります．

1. プログラムをメモリに書き込む．
2. プログラムの開始番地を PC レジスタに書き込む．
3. **BREAK スイッチが上**に **STEP スイッチが下**に倒れていることを確認する．
   (倒れていなかった場合は倒す．)
4. データスイッチに，プログラム実行を停止したい命令のアドレスをセットする．
5. **RUN スイッチ**を押す．
6. データスイッチで指定したアドレスの命令を**実行後**，停止する．

例 プログラムのブレークポイントモードでの実行
前の例と同じプログラムを，ブレークポイントモードで実行してみます．

プログラムの実行手順

1. 前の例と同様にプログラムを打ち込む．
2. 実行を停止したい命令のアドレス (今回は，02_{16}) をデータスイッチにセットする．
3. STEP スイッチを**下**に，BREAK スイッチを**上**に倒して (ブレークポイントモードにして)，前の例と同様にプログラムの実行を開始する．
4. ブレークポイントモードなので 02_{16} 番地の命令を実行したら停止する．このとき，PC は次命令の番地 03_{16} を指している．
5. 再度 RUN スイッチを押すと 03_{16} 番地からプログラムの実行を再開する．
6. 03_{16} 番地には HALT 命令が格納されているので実行が停止する．

4.3　リセットスイッチ

基板の左下にある RESET と書かれた押しボタンスイッチのことです．TeC をリセットしたいときに使用します．リセットスイッチを押すと，フラグ，レジスタ，TeC に内蔵された入出力装置の状態が (‘0’ に) クリアされます．メモリの内容と，ロータリースイッチの状態はクリアされません．

4.4　操作音を小さくする

押しボタンスイッチを押すと，ボタンが押されたことが確認できるように,「ピッ」と短い電子音が鳴るようになっています．しかし，静かな部屋で使用する場合，この音が邪魔になります．

このようなときは，**STOP スイッチ**を押しながら **RESET スイッチ**を押して下さい．電子音が，かすかに聞こえる「プッ」という音に変わります．

演習

1. 付録Aの電子オルゴールプログラムを入力／実行しなさい．
2. 付録Aの電子オルゴールプログラムの曲データを変更して実行しなさい．

第5章　プログラミング

この章では，TeC のプログラミングを学習します．まず，TeC の内部構成を勉強し，ノイマン型コンピュータの例として，TeC が適切であることを確認します．次に，TeC の命令と，それを使用したプログラムの作成を勉強します．この章の内容をマスターすれば，ノイマン型コンピュータがどのようなもので，何ができて何ができないのか，はっきり分かってきます．

5.1　コンピュータの構成

プログラミングを始める前に，プログラミングの対象となるコンピュータの構成を確認します．ここでは，一般のコンピュータの構成と，TeC の構成の両方を説明します．TeC の構成は一般のコンピュータの構成を簡単化したもので，原理的には同じものだと分かります．

5.1.1　一般的なコンピュータの構成

パソコン等の一般的なコンピュータの構成は，図 5.1 のようになっています．各部分の役割は次の通りです．

1. CPU(Central Processing Unit:中央処理装置)
   命令を読み込んで，命令に従い計算をするコンピュータの心臓部です．計算機能，制御機能を持っています．CPU が他の部分に積極的に働きかけることにより，コンピュータが働きます．
2. 主記憶装置 (メモリ)
   プログラムやデータを記憶する記憶装置です．CPU が高速に読み書きをすることができます．
3. 入出力インタフェース
   入出力装置をコンピュータに接続するための回路です．入出力装置の種類や機種毎に，専用のインターフェース回路が必要になります．
4. 入出力装置
   コンピュータの外部と，データのやりとりをする装置のことです．キーボード，マウス，ディスプレイ，プリンタや通信装置等がこれにあたります．ハードディスクドライブや CD-ROM ドライブ等も入出力装置の仲間ですが，データを記憶する装置であるので少し性格が異なります．そこで，これらは補助記憶装置と呼ばれることがあります．
5. バス
   CPU と主記憶装置や入出力インタフェースを接続する配線のことです．アドレスやデータ，制御信号等がバスで伝達されます．CPU はバスを制御することにより，バスに接続された装置とデータのやりとりをします．

5.1.2　TeC の構成

TeC の構成は図 5.2 のようになっています．接続されている入出力装置が少ない，メモリの容量が小さい等の違いはありますが，基本的には一般的なコンピュータと同じになっています．これらの大部分は，図 4.1 の「マイコン本体」にある正方形の LSI の内部に集積されています．各部分の役割りは次の通りです．

1. CPU
   TeC の CPU も計算機能，制御機能を持っています．図 4.3 に示した**フラグ**と**レジスタ**は CPU の内部にあります．
2. 主記憶装置 (メモリ)
   TeC のメモリは，図 4.3 に示した 8 ビット構成，256 アドレスのものです．一つのアドレスに 8 ビットの情報を記憶することができます．0〜255(00_{16}〜FF_{16}) の範囲でアドレスを指定するためには，アドレス情報も 8 ビット必要ですので，8 ビットアドレスと呼ぶこともあります．
3. 入出力インタフェース
   標準ではシリアル入出力回路 (図 4.1 の通信機能部分) と，スピーカ，コンソールパネルのデータスイッチが入出力インターフェース回路を通してバスに接続されています．

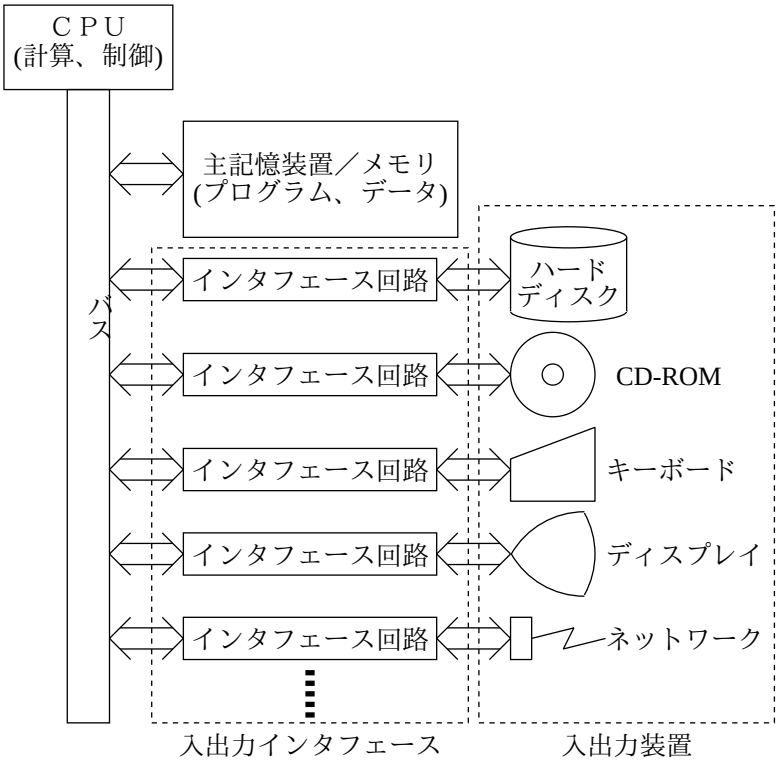


図 5.1: 一般的なコンピュータの構成

4. 入出力装置
   シリアル入出力回路，スピーカ，コンソールパネルのデータスイッチが搭載されています．

5. バス
   CPU と主記憶装置や入出力インタフェースの間で次の情報を伝達します．

   (a) 8 ビットのアドレス

   (b) 8 ビットのデータ

   (c) いくつかの制御情報

   また，拡張コネクタ (図 4.1 参照) にも接続されており，外部に入出力インタフェースを追加することが可能です．

5.2　機械語プログラミング

機械語命令の羅列がプログラムです．機械語命令の羅列を作る作業がプログラミングです．

5.2.1　機械語命令

コンピュータの CPU は，メモリから命令を取り出し，取り出した命令を解釈し，決められた計算等を行います．CPU が解釈できる状態の命令を，「CPU ＝機械」が解釈できる命令なので**「機械語命令」**と呼びます．機械語命令は 2 進数でメモリに書き込みます．

2 進数を用いた機械語の表現方法だけでは人間にとって分かり難いので，命令を意味する英語を簡略化した短い綴で表します．これを**「ニーモニック」**と呼びます．

機械語命令	ニーモニック	意味
$0000\ 0000_2$	NO	No Operation
$1111\ 1111_2$	HALT	Halt

5.2.2　ノイマン型コンピュータの特徴

TeC もノイマン型コンピュータの一種です．ノイマン型コンピュータの特徴として次の 3 点があげられますが，これは，TeC にも当てはまります．

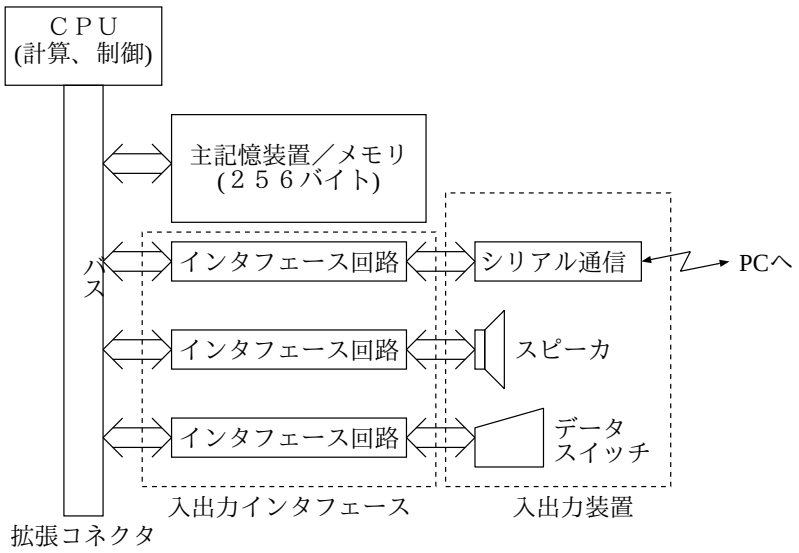


図 5.2: TeC の構成

1. プログラム内蔵 (ストアードプログラム) 方式
   データだけでなく，プログラムもメモリに記憶する方式です．TeC でもプログラムはメモリに書き込んで実行します．
2. 逐次実行
   プログラムの命令を，メモリのアドレス順に一つ一つ順番に実行することを言います．
3. 2 進法
   コンピュータの内部では，プログラムやデータの表現に 2 進数を使います．

5.2.3 機械語プログラミング

ノイマン型コンピュータ上での機械語プログラミングとは，逐次実行により実行される順に機械語命令の羅列を作る作業のことです．作ったプログラムは，2 進数にしてメモリに書き込んで実行します．

TeC にどのような機械語命令があるかを，次の節から種類別に説明します．

5.3 特殊な命令

まず，TeC の機械語命令の中で特別なものから説明します．

5.3.1 NO(No Operation) 命令

意味：何もしない．
ニーモニック：`NO`
命令フォーマット：NO 命令は，1 バイト長の命令です．

第 1 バイト	
OP	GR XR
0000_2	00 00_2

命令フォーマットは，その機械語命令が 2 進数でどのように表現されるかを表します．NO 命令の場合，機械語が 1 バイトであること，その 1 バイトの内容が，OP フィールド 4 ビットが 0000_2，GR フィールド 2 ビットが 00_2，XR フィールド 2 ビットが 00_2 であることが命令フォーマットの図から分かります．NO 命令以外の命令も，フィールドのビット数は同じです．

5.3.2 HALT(Halt) 命令

意味：プログラム実行の終了．
ニーモニック：`HALT`
命令フォーマット：HALT 命令は，1 バイト長の命令です．

第 1 バイト	
OP	GR XR
1111_2	11 11_2

5.4　データ転送命令

TeC の CPU とメモリの間でデータ転送をする機械語命令を説明します.

5.4.1　LD(Load) 命令

Load は,「荷物等を積み込む」と言う意味の英語です．LD は，Load の綴を縮めたものです.

意味： 主記憶 (メモリ) からレジスタへデータを転送します．(メモリからレジスタへ値をコピーします．メモリの値は変化しません．)

フラグ： 変化しません.

ニーモニック： `LD GR,A[,XR]`
`GR` は，転送先レジスタを表します．`A` はデータの置いてあるメモリのアドレス (番地) を表します．`[,XR]` は，省略可能です．将来，きちんと説明するまで，`[,XR]` は常に省略 (この部分を書かない) してください.

命令フォーマット： LD 命令は 2 バイトの長さを持ちます．命令の各ビットは次の通りです.

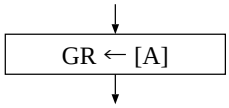
第 1 バイト		第 2 バイト
OP	GR XR	
0001_2	$GR\ XR$	aaaa aaaa

- OP フィールド (4 ビット) は LD 命令を示す 0001_2 にします.
- GR フィールド (2 ビット) はデータを格納するレジスタを表します．GR フィールドの値は次の表のような意味を持ちます.

GR	意味
00	G0
01	G1
10	G2
11	SP

- XR フィールド (2 ビット) は，将来，詳しく説明するまでは常に”00” にします.
- 第 2 バイト (*aaaa aaaa*) は，A(番地) を 2 進数で格納します.

フローチャート： フローチャートでは，LD 命令を次のように描きます．`[` と `]` を書き忘れないように，注意して下さい.

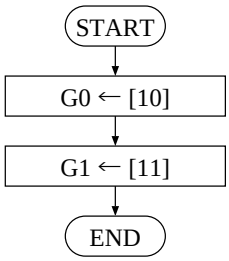


使用例： LD 命令を使用したプログラムの例を示します.

番地	機械語	ラベル	ニーモニック
00_{16}	$10_{16}\ 0A_{16}$		`LD GO,10`
02_{16}	$14_{16}\ 0B_{16}$		`LD G1,11`
04_{16}	FF_{16}		`HALT`

例の中で機械語は 16 進数で書いてありますが，2 進数に変換すると 0 番地の命令は $0001\ 0000_2$，$0000\ 1010_2$ です．上の命令フォーマットと対応を確認してください.

このプログラムをフローチャートで表現すると，次のようになります.



命令表
TeC でどんな機械語命令が使用できるかは，本体のケースに張り付けた命令表 (または，付録 C の命令表) により確認することができます．これまでに出てきた NO，HALT，LD 命令について，命令表を確認して下さい．ニーモニックと命令の名前，命令のフォーマットが分かりますね.
その他にも，まだ習っていない情報がたくさん掲載されています．一通り勉強が終わったら，この命令表だけで TeC を自由にプログラミングできるようになります.

RUN ランプが点滅したら
CPU が機械語命令として解釈できない命令を実行しようとしたことを表します．PC が解釈できなかった命令の次の番地を指した状態で CPU が停止します．PC の値からおかしな命令を見付けて訂正してください.

5.4.2 ST(Store) 命令

Store は,「倉庫に保管する」と言う意味の英語です．ST は，Store の綴を縮めたものです.

意味： レジスタからメモリへデータを転送します.(レジスタからメモリへ値をコピーします．レジスタの値は変化しません．)

フラグ： 変化しません.

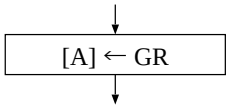
ニーモニック： `ST GR,A[,XR]`

GR は，転送元レジスタを表します．A はデータを保管するメモリのアドレス (番地) を表します．[,XR] は，省略可能です．将来，きちんと説明するまで，[,XR] は常に省略 (この部分を書かない) してください.

命令フォーマット： ST 命令は 2 バイトの長さを持ちます．命令のフォーマットは次の通りです．各フィールドの意味は，LD 命令と同様です.

第 1 バイト		第 2 バイト
OP	GR XR	
0010_2	$GR\ XR$	aaaa aaaa

フローチャート： フローチャートでは，ST 命令を次のように描きます．[と] を書き忘れないように，注意して下さい.



使用例： LD 命令と ST 命令を組み合わせて，7 番地のデータを 8 番地にコピーして停止するプログラムの例を示します．番地と機械語の欄はいつも 16 進数で書くので，この例から小さく 16 と書くのを止めます．なお，ニーモニック欄の数値は 10 進数ですので，注意してください.

番地	機械語	ラベル	ニーモニック
00	10 07		LD GO,7
02	20 08		ST GO,8
04	FF		HALT

問題

1. 11_{16} 番地のデータを 12_{16} 番地に，10_{16} 番地のデータを 11_{16} 番地にコピーして停止するプログラムを，上の使用例のように書きなさい.
2. 10_{16} 番地のデータと 11_{16} 番地のデータを交換して停止するプログラムを，上の使用例のように書きなさい.
3. これらのプログラムを，実際に TeC で実行して正しく動くことを確認しなさい.

プログラムの作成手順

上の使用例のように, 表を作成しながらプログラムを書きます.
手順は次の通りです.

1. 表の枠を書く.
2. ニーモニックでプログラムを書く．プログラムに間違えが無いか，この段階で良く考える.
3. 機械語命令の長さを考えながら番地欄を記入する．(これは，機械的な作業．ミスをしないよう慎重に．)
4. 機械語欄を記入する．(これも，機械的な作業．ミスをしないように．)
5. 全部記入したら，TeC に打ち込んで実行してみる.
6. うまく動かなかったら，全てのステップを再度確認し間違えを探す.
7. うまく動いたら完成 !!! (うまく動いたかどうかの確認も慎重に！)

表に記入したら終わりではありません．TeC に打ち込んで実際に動くことを確認してください．動くはずなのに動かないことが多い (まず，動かないと考えた方が良い) です.
実際に動かし正しく動作することを確認できるまでは，プログラムは完成していません.

RUN ランプに注意

TeC の RUN ランプはプログラム実行中に点灯しています．プログラムが暴走し終了しない時は，RUN ランプが点灯したままになります．プログラム実行中でも，コンソールパネルを操作してメモリやレジスタの値を表示できますので，プログラムが暴走していることに気づかないことがあります．RUN ランプを常に気にするようにして下さい.

5.5　算術演算命令

5.5.1　ADD(Add) 命令

ADD は名前の通り，加算をする命令です．

意味： レジスタの値とメモリデータの和を計算し，結果を元のレジスタに格納します．(メモリの値は変化しません．)

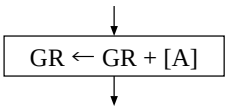
フラグ： 計算の結果により変化します．

ニーモニック： `ADD GR,A[,XR]`
`GR` は，足し算の対象になるレジスタを表します．このレジスタの値とメモリの値が足し合わされ，結果がこのレジスタに格納されます．`A` と `[,XR]` の意味は LD 命令と同様です．

命令フォーマット： ADD 命令は 2 バイトの長さを持ちます．命令の各フィールドの意味は，LD 命令と同様です．

第 1 バイト		第 2 バイト
OP	GR XR	
0011_2	$GR\ XR$	aaaa aaaa

フローチャート： フローチャートでは，ADD 命令を次のように描きます．[と] を書き忘れないように，注意して下さい．



使用例： 7 番地のデータと 8 番地のデータの和を計算し，9 番地に格納するプログラムの例を示します．

番地	機械語	ラベル	ニーモニック
00	10 07		LD GO,7
02	30 08		ADD GO,8
04	20 09		ST GO,9
06	FF		HALT

このプログラムを実行するときは，予め 7 番地，8 番地に足し合わせるデータを格納しておく必要があります．例えば，7 番地に 1，8 番地に 2 を格納してからこのプログラムを実行すると，計算結果としてプログラムにより 9 番地に 3 が格納されます．

5.5.2　SUB(Subtract) 命令

Subtract は,「引き算をする」と言う意味の英語です．SUB は Subtract の綴を縮めたものです．

意味： レジスタの値とメモリデータの差を計算し，結果を元のレジスタに格納します．(メモリの値は変化しません．)

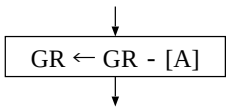
フラグ： 計算結果により変化します．

ニーモニック： `SUB GR,A[,XR]`
`GR` は，引き算の対象になるレジスタを表します．このレジスタの値からメモリの値が引かれ，結果がこのレジスタに格納されます．`A` と `[,XR]` の意味は LD 命令と同様です．

命令フォーマット： SUB 命令は 2 バイトの長さを持ちます．命令の各フィールドの意味は，LD 命令と同様です．

第 1 バイト		第 2 バイト
OP	GR XR	
0100_2	$GR\ XR$	aaaa aaaa

フローチャート： SUB 命令は次のように描きます．



使用例： 7 番地のデータから 8 番地のデータを引いた結果を 9 番地に格納するプログラムの例を示します．

番地	機械語	ラベル	ニーモニック
00	10 07		LD GO,7
02	40 08		SUB GO,8
04	20 09		ST GO,9
06	FF		HALT

このプログラムを実行するときは，予め 7 番地に引かれる数，8 番地に引く数を格納しておく必要があります．

問題

10_{16} 番地のデータと 11_{16} 番地のデータの和を 12_{16} 番地に，差を 13_{16} 番地に格納するプログラムを作成し，TeC で実行して正しく実行できたことを確かめなさい．

フラグ
いくつかのプログラムを作成して動かして見ました．そのとき，TeC の C，S，Z ランプが点灯したり消灯したりしたのが分かったでしょうか ？これらのランプは，対応したフラグ (Flag:旗) の値を表示しています．どんなときランプが点灯し (フラグが 1 になり)，どんなときランプが消灯した (フラグが 0 になった) のか説明します．

C(Caryy) フラグ
Carry は「桁を繰り上げる」と言う意味です．C フラグは，計算中に 8bit の最上位桁からの「桁上がり」や，8bit の最上位桁での「桁借り」が発生したことを表します．つまり，計算結果が 255 を超えてしまったことや，0 より小さくなったことを表します．C フラグは，計算値が符号無しと考えたときのオーバーフローを表しています．

S(Sign) フラグ
Sign は，**「符号」**を意味します．計算の結果を符号付き 2 進数として解釈した場合，負の値になることを表します．つまり，計算結果の 8bit の最上位ビットが 1 のとき 1 になります．

Z(Zero) フラグ
Zero は，名前の通り計算の結果がゼロになったことを表します．つまり，計算結果の 8bit の全てのビットが 0 のとき 1 になります．

これら三つのフラグは，命令表で「フラグ変化」の欄に「○」印が付いている演算命令を実行する度に，計算結果によって変化します．つまり，直前の演算の結果を反映しています．これまでに出てきた命令では，ADD，SUB がフラグを変化させる命令です．

うまく動かない場合
プログラムが正しく動かない人は，1 命令ずつ実行しながら，メモリやレジスタの値が予想通りに変化するか調べて下さい．1 命令ずつ実行する方法は，4.2.2 中の「プログラムのステップ実行」に説明してあります．
面倒くさいと思わないで，地道に問題点を捜しましょう．問題点を見つけるコツが分かってきたら，演習がすごく楽になります．慣れるまで少し辛抱して下さい．

5.6 ジャンプ命令

ノイマン型コンピュータの特徴は逐次実行です．命令を番地の順番に一つ一つ順に実行します．プログラムの実行が進んで行く流れを**「プログラムの流れ」**と言います．プログラムの流れは PC(Program Counter) によって管理され，通常は PC が順次増加します．

しかし，プログラムの同じ部分を繰り返したり，条件によりプログラムの動きを変更する目的で，流れを別の場所に飛ばす (Jump) ことができると便利です．

ジャンプ命令はこのような目的で PC の値を変更し，プログラムの流れを別の番地へ飛ばすものです．プログラムの流れは飛んで行った先にあるプログラムを順番に実行する流れになり，もとに戻って来ることはありません．

5.6.1 JMP(Jump) 命令

この命令を実行するとプログラムの流れが，必ず指定の番地にジャンプします．

意味： プログラムの流れをジャンプさせます．

フラグ： 変化しません．

ニーモニック： `JMP A[,XR]`
`A` はジャンプ先のアドレス (番地) を表します．`[,XR]` は，省略可能です．将来，きちんと説明するまで，`[,XR]` は常に省略 (この部分を書かない) してください．

命令フォーマット： JMP 命令は 2 バイトの長さを持ちます．GR フィールドは必ず 00 にします．第 2 バイトがジャンプ先のアドレスを示します．

第 1 バイト		第 2 バイト
OP	GR XR	
1010_2	00 XR	aaaa aaaa

フローチャート： JMP 命令はフローチャートの線に対応します．次の命令で詳しく説明します．

使用例： 00 番地の命令を，いつまでも繰り返す (止まらない) プログラムの例を示します．

番地	機械語	ラベル	ニーモニック
00	30 04		ADD G0,4
02	A0 00		JMP 0

ラベル

JMP 命令の使用例では，ジャンプする先のアドレスをニーモニックの中に直接書きました．もしも，プログラムに変更があり，ジャンプする先のアドレスが変化したらどうでしょうか．プログラムに直接アドレスが書いてあると，それを書き直す必要が生じます．

番地が変化してもニーモニックで書いたプログラムを変更しなくて良いように，アドレスの代わりにその場所に付けた記号を使うと便利です．

場所に付けた記号のことをラベルと言います．ラベルを用いてプログラムを書き換えると次のようになります．

番地	機械語	ラベル	ニーモニック
00	30 04	**LOOP**	ADD G0,4
02	A0 00		JMP **LOOP**

データの定義 (DC 命令)

ラベルを用いることにより，ニーモニックで書いたプログラムの取扱いが非常に便利になりました．しかし，データの部分はニーモニックに記述できていません．これでは，ニーモニックだけでプログラムの全体像が理解できません．

そこで，データを記述するための “DC(Define Constant)” 命令を追加します．DC 命令は，機械語の代わりにオペランドで指定した値のデータを生成します．また，DC 命令の行にラベルを付けることによりデータもラベルで参照できるようになります．

次は，JMP 命令の使用例を DC 命令を用いて書き直したものです．4 番地に値 “1” のデータがあることが，うまく記述できました．

番地	機械語	ラベル	ニーモニック
00	30 04	LOOP	ADD G0,**ONE**
02	A0 00		JMP LOOP
04	01	**ONE**	DC 1

領域の定義 (DS 命令)

DS 命令は DC 命令に良く似た命令です．DC 命令は機械語命令の代わりにデータを生成しました．DS 命令は結果を格納するための領域を生成します．指定された数値は，領域の大きさになります．

5.6.2 JZ(Jump on Zero) 命令

Z フラグが 1 のとき (計算結果が 0 だったとき) だけジャンプします．

意味： Z フラグが 1 ならジャンプします．

フラグ： 変化しません．

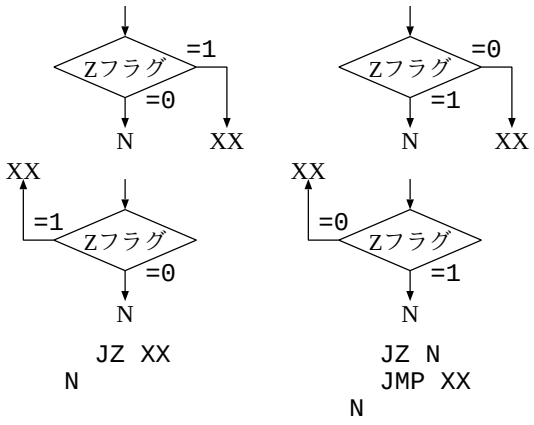
ニーモニック： `JZ A[,XR]`

JMP 命令と同様です．

命令フォーマット： JZ 命令は 2 バイトの長さを持ちます．GR フィールドは必ず 01 にします．第 2 バイトがジャンプ先のアドレスを示します．

第 1 バイト		第 2 バイト
OP	GR XR	
1010_2	01 XR	aaaa aaaa

フローチャート： JZ 命令のフローチャートは，次のように様々な描き方が考えられます．例には、JMP 命令と組合せて菱形一つに対応させたものもありますが，こだわる必要はありません．場合によって，柔軟にアレンジして下さい．



使用例： 結果がゼロだったら停止するプログラムの例を示します．定数の “1” を使用するために DC 命令でメモリの 07_{16} 番地にデータ 01_{16} を置きました．定数を使用するためにはメモリ上に定数データを置く必要があります．

番地	機械語	ラベル	ニーモニック
00	30 07	LOOP	ADD G0,ONE
02	A4 06		JZ STOP
04	A0 00		JMP LOOP
06	FF	STOP	HALT
07	01	ONE	DC 1

5.6.3 JC(Jump on Carry) 命令

C フラグが 1 のときだけジャンプします.

意味： C フラグが 1 ならジャンプします.

フラグ： 変化しません.

ニーモニック： `JC A[,XR]`
JMP 命令と同様です.

命令フォーマット： JC 命令は 2 バイトの長さを持ちます. GR フィールドは必ず 10 にします. 第 2 バイトがジャンプ先のアドレスを示します.

第 1 バイト		第 2 バイト
OP	GR XR	
1010_2	10 XR	aaaa aaaa

フローチャート： JC 命令のフローチャートは, JZ 命令と同様な考えで描きます. JZ 命令を参考にして下さい.

5.6.4 JM(Jump on Minus) 命令

S フラグが 1 のとき (計算結果が負だったとき) だけジャンプします. 名前が, JS 命令ではなく JM 命令になっているので注意して下さい.

意味： S フラグが 1 ならジャンプします.

フラグ： 変化しません.

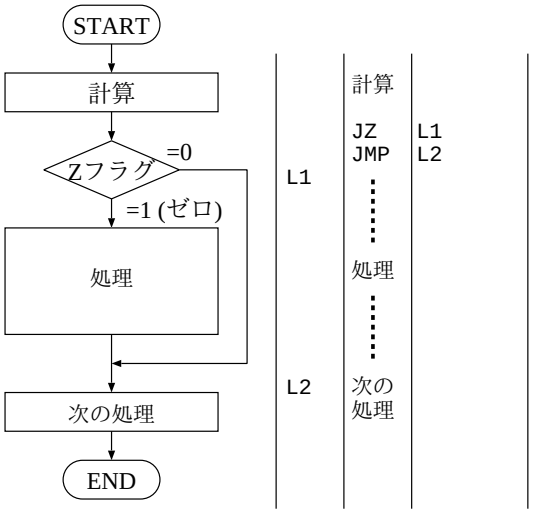
ニーモニック： `JM A[,XR]`
JMP 命令と同様です.

命令フォーマット： JM 命令は 2 バイトの長さを持ちます. GR フィールドは必ず 11 にします. 第 2 バイトがジャンプ先のアドレスを示します.

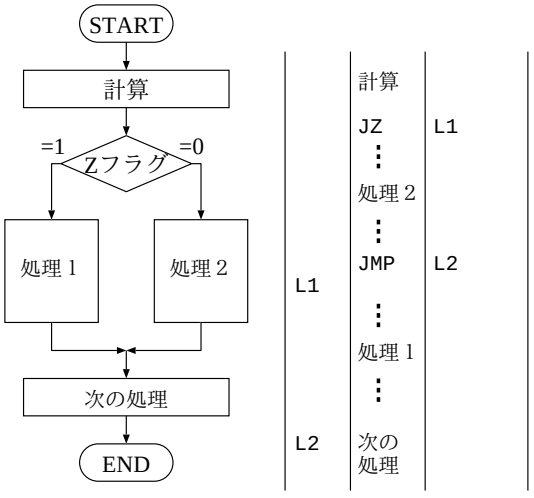
第 1 バイト		第 2 バイト
OP	GR XR	
1010_2	11 XR	aaaa aaaa

フローチャート： JM 命令のフローチャートも, JZ 命令と同様な考えで描きます. JZ 命令を参考にして下さい.

条件判断 1
ジャンプ命令を用いて, 条件判断があるプログラムを作ることができます. 次のフローチャートとプログラムは, 計算結果がゼロだった場合だけ処理をするものです. このように, ある条件の場合だけ処理をするプログラムを作ることができます.

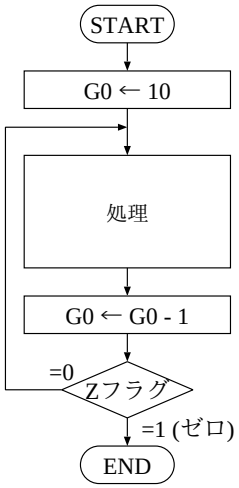


条件判断 2
条件によって, 二つの処理のどちらかを選んで実行するプログラムを作ることができます. 次のフローチャートとプログラムは, 計算結果がゼロだった場合は「処理 1」を, ゼロ以外だった場合は「処理 2」を実行します.



繰り返し処理 1

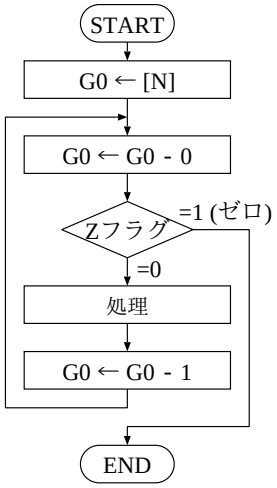
次のように，ジャンプ命令を用いて同じ操作を繰り返すプログラムを作ることができます．次のフローチャートとプログラムは，点線部分を 10 回繰り返すためのものです．




```
        LD    G0,N
LOOP    ⋮
        処理
        ⋮
        SUB   G0,ONE
        JZ    STOP
        JMP   LOOP
STOP    HALT
N       DC    10
ONE     DC    1
```

繰り返し処理 2

計算結果によって繰り返し回数が決まる場合等，繰り返し回数がゼロの場合も考慮しなければならないことがあります．そのような場合は，条件判断を前に移動すると，うまく処理できます．次のフローチャートとプログラムは，点線部分を N 回繰り返すためのものです．条件判断の前の引算は，G0 の値でフラグを強制的に変化させるためのものです．



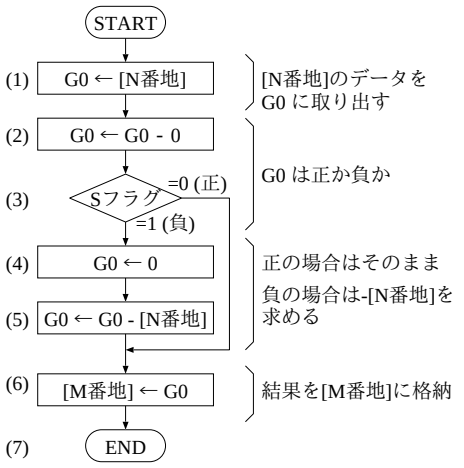

```
        LD    G0,N
LOOP    SUB   G0,ZERO
        JZ    STOP
        ⋮
        処理
        ⋮
        JMP   LOOP
STOP    HALT
N       DS    1
ZERO    DC    0
```

例題 5-1 絶対値を求める．

問題： N 番地のデータの絶対値を計算し，M 番地に格納するプログラムを作りなさい．

考え方： 値が正なのか負なのかは，値からゼロを (SUB 命令で) 引き，そのときのフラグの変化で調べます．
負の値の絶対値は，ゼロからその値を引くことで求めることができます．

解答： 次のフローチャートのようなプログラムを作ります．



注意：[N番地]は、N番地に格納されているデータのこと

番地	機械語	ラベル	ニーモニック	
00	10 10	START	LD	G0,N
02	40 0F		SUB	G0,ZERO
04	AC 08		JM	L1
06	A0 0C		JMP	L2
08	10 0F	L1	LD	G0,ZERO
0A	40 10		SUB	G0,N
0C	20 11	L2	ST	G0,M
0E	FF		HALT	
0F	00	ZERO	DC	0
10	FF	N	DC	-1
11	00	M	DS	1

解説： プログラムの内容を説明します．

(1) G0 レジスタに値をロードします．

(2) G0 レジスタから 0 を引きます．値は変化しませんが，計算結果によりフラグが変化します．

(3) JM 命令は S フラグの値によりジャンプします．

(4)，(5) この部分は，値が負だった場合のみ実行されます．$0 - [N$ 番地$]$ を計算し，値の絶対値を G0 に求めます．

(6) G0 の値を M 番地に格納します．

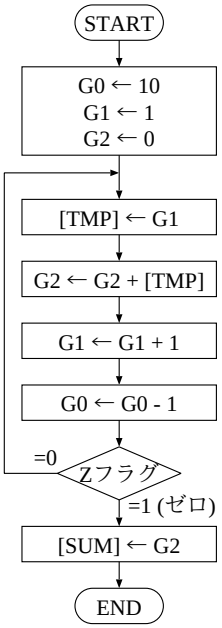
例題 5-2 $1+2+3+...+10$ を計算する.

問題： $1+2+3+...+10$ を計算し結果を SUM 番地に求めるプログラムを作りなさい.

考え方： G0 以外に，G1，G2，SP の三つのレジスタが自由に使用できるので，それぞれのレジスタに足す値を記憶する，合計を記憶する等の役割りを分担させ，繰り返し (10 回繰り返す) をうまく利用すると計算できます.

TeC にはレジスタの値同士を足し算する命令がありません．一方のレジスタ値をメモリに格納してから，足し算する必要があります.

解答： G0 を繰り返し回数のカウント，G1 をレジスタに足す数の記憶 (1,2,3,...,10)，G2 を合計の記憶に使用することにしました．フローチャートとプログラムは次のようになります.




```
        LD   G0,N
        LD   G1,ONE
        LD   G2,ZERO
LOOP    ST   G1,TMP
        ADD  G2,TMP
        ADD  G1,ONE
        SUB  G0,ONE
        JZ   STOP
        JMP  LOOP
STOP    ST   G2,SUM
        HALT
N       DC   10
ONE     DC   1
ZERO    DC   0
TMP     DS   1
SUM     DS   1
```

C フラグの詳しい説明

C フラグは，計算中に，8bit の最上位桁からの「桁上がり (Carry)」が発生したことや，8bit の最上位桁での「桁借り (Borrow)」が発生したことを表します．例えば，次の計算では最上位桁からの「桁上がり」は発生しませんので，C フラグは “0” になります.

	$0000\ 0001_2$	(1)
+	$0000\ 0011_2$	(3)
$\boxed{0}$	$0000\ 0100_2$	(4)
C		

次の計算では最上位桁からの「桁上がり」が発生し，C フラグが “1” になります．C フラグは，計算値が符号無しと考えたときのオーバーフローを表しています.

		符号無	符号付
	$1111\ 1111_2$	(255)	(-1)
+	$0000\ 0001_2$	(1)	(+1)
$\boxed{1}$	$0000\ 0000_2$	(オーバ	(0)
C		フロー)	

また，引き算でも C フラグが 1 になることがあります．例えば，次のように小さな数から大きな数を引いた場合です．機械的に引き算をすると，最上位桁で桁借りが発生します．この時も，C フラグが 1 になります.

		符号無	符号付
	$0110\ 0100_2$	(100)	(+100)
−	$0110\ 1110_2$	(110)	(+110)
$\boxed{1}$	$1111\ 0110_2$	(オーバ	(-10)
C		フロー)	

このように引き算の場合は，C フラグは，計算が符号無しと考えたときの，負へのオーバーフローを表しています.

問題

1. かけ算プログラム
   N 番地と M 番地の値のかけ算を計算し，L 番地に結果を格納するプログラムを作りなさい.
2. かけ算プログラムの改良
   前のプログラムを [N] または，[M] がゼロのときでも正しく動くように改良しなさい.

5.7　比較命令

5.7.1　CMP(Compare) 命令

Compare は,「比較をする」と言う意味の英語です．CMP は Compare の綴を縮めたものです．比較は引き算により行います．レジスタの値からメモリの値を引いて，その結果によりフラグを変化させます．引き算結果そのものは，どこにも格納しないで捨てます．

意味： レジスタの値とメモリデータを比較します．引き算の結果により，フラグだけが変化します．

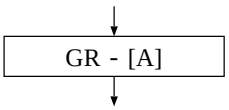
フラグ： 計算結果により変化します．

ニーモニック： `CMP GR,A[,XR]`
`GR` で指定されたレジスタの値と，メモリの値が比較されます．`A` と `[,XR]` の意味は LD 命令と同様です．

命令フォーマット： CMP 命令は 2 バイトの長さを持ちます．各フィールドの意味は，LD 命令と同様です．

第 1 バイト		第 2 バイト
OP	GR XR	
0101_2	$GR\ XR$	aaaa aaaa

フローチャート： CMP 命令は次のように描きます．



使用例： A，B 二つのデータで大きい方を選んで，C に格納するプログラムの例です．

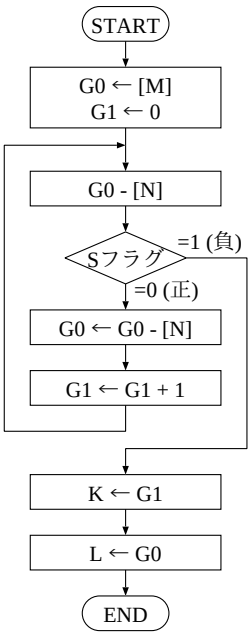
番地	機械語	ラベル	ニーモニック	
00	10 0D		LD	G0,A
02	50 0E		CMP	G0,B
04	AC 08		JM	LB
06	A0 0A		JMP	LC
08	10 0E	LB	LD	G0,B
0A	20 0F	LC	ST	G0,C
0C	FF		HALT	
0D	64	A	DC	100
0E	C8	B	DC	200
0F	00	C	DS	1

例題 5-3 割算を計算する．

問題： M 番地の値を N 番地の値で割り，商を K 番地，余りを L 番地に求めるプログラムを作りなさい．

考え方： TeC には割算命令がありません．割算は引き算の繰り返しで計算できます．割られる数から，割る数を引くことを繰り返します．引くことができなくなったら終了します．このとき，引いた回数が商，引いた結果が余りになります．

解答： フローチャートとプログラムは次のようになります．




```
      LD    G0,M
      LD    G1,ZERO
LOOP  CMP   G0,N
      JM    STOP
      SUB   G0,N
      ADD   G1,ONE
      JMP   LOOP
STOP  ST    G1,K
      ST    G0,L
      HALT
M     DC    100
N     DC    10
ONE   DC    1
ZERO  DC    0
K     DS    1
L     DS    1
```

解説： CMP 命令と JM 命令で G0 が N 番地のデータより大きいか確認してから，引き算を実行します．

問題

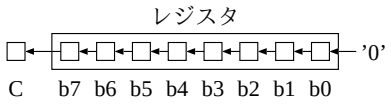
以前の，絶対値を求めるプログラムを CMP 命令を用いて書き直しなさい．

5.8 シフト (桁ずらし) 命令

データの 2 進数を左右に桁移動する命令をシフト命令と言います．TeC は以下に説明する 4 種類のシフト命令を持っています．

5.8.1 SHLA(Shift Left Arithmetic) 命令

左算術 (算術＝ Arithmetic) シフトと言います．レジスタのデータを左方向に 1 ビットずらします．右側から常に “0” が入力されます．左にはみ出したビットは，C フラグの値になります．



意味： レジスタの値を，左に 1 ビット，算術シフトします．

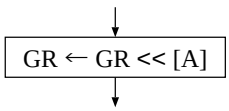
フラグ： C フラグは上の説明のように，S，Z フラグは計算結果により変化します．

ニーモニック： `SHLA GR`
GR が，シフトされるレジスタを表します．GR の値とレジスタの対応は，LD 命令等と同じです．

命令フォーマット： SHLA 命令は 1 バイトの命令です．XR フィールドは必ず 00 にします．

第 1 バイト	
OP	GR XR
1001_2	GR 00

フローチャート： SHLA 命令は次のように描くことにします．なお，この書き方は，Java 言語や C 言語のシフト演算子を真似たものです．



使用例： A 番地のデータを 2 ビット左にシフトし，B 番地に格納する例です．左に 2 ビットシフトすることは，× 4 を計算したのと同じ結果になります．

番地	機械語	ラベル	ニーモニック	
00	10 07		LD	G0,A
02	90		SHLA	G0
03	90		SHLA	G0
04	20 08		ST	G0,B
06	FF		HALT	
07	01	A	DC	1
08	00	B	DS	1

5.8.2 SHLL(Shift Left Logical) 命令

左論理 (論理=Logical) シフトと言います．この命令は SHLA と全く同じ動作をします．(左シフトでは，算術と論理の差はない)

意味： レジスタの値を，左に 1 ビット，論理シフトします．

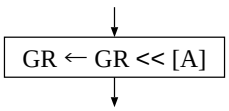
フラグ： SHLA 命令と同様です．

ニーモニック： `SHLL GR`
SHLA 命令と同様です．

命令フォーマット： SHLL 命令は 1 バイトの命令です．XR フィールドは必ず 01 にします．

第 1 バイト	
OP	GR XR
1001_2	GR 01

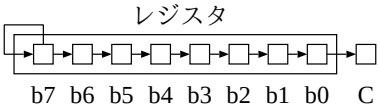
フローチャート： SHLL 命令は次のように描くことにします．SHLA 命令と全く同じ動作をするので，フローチャートの描き方も，SHLA 命令と全く同じです．



使用例： SHLA 命令と同じ動作をする命令なので，省略します．

5.8.3　SHRA(Shift Right Arithmetic) 命令

右算術 (算術＝ Arithmetic) シフトと言います．レジスタのデータを右方向に 1 ビットずらします．左側からシフト前のデータの最上位ビット (符号ビット) と同じ値が入力されます．右にはみ出したビットは，C フラグの値になります．



意味： レジスタの値を，右に 1 ビット，算術シフトします．

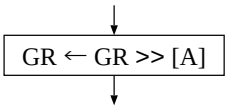
フラグ： SHLA 命令と同様です．

ニーモニック： `SHRA GR`
SHLA 命令と同様です．

命令フォーマット： SHRA 命令は 1 バイトの命令です．XR フィールドは必ず 10 にします．

第 1 バイト	
OP	GR XR
1001_2	GR 10

フローチャート： SHRA 命令は次のように描くことにします．

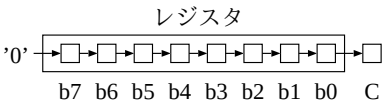


使用例： A，B 番地のデータを 1 ビット右にシフトし，C，D 番地に格納する例です．右に 1 ビットシフトすることは，÷ 2 を計算したのと同じ結果になります．算術シフトの場合は，負の数でも正しく 1/2 の値になります．

番地	機械語	ラベル	ニーモニック	
00	10 0B		LD	G0,A
02	92		SHRA	G0
03	20 0D		ST	G0,C
05	14 0C		LD	G1,B
07	96		SHRA	G1
08	24 0E		ST	G1,D
0A	FF		HALT	
0B	08	A	DC	8
0C	F8	B	DC	-8
0D	00	C	DS	1
0E	00	D	DS	1

5.8.4　SHRL(Shift Right Logical) 命令

右論理 (論理＝ Logical) シフトと言います．レジスタのデータを右方向に 1 ビットずらします．左側から “0” が入力されます．右にはみ出したビットは，C フラグの値になります．



意味： レジスタの値を，右に 1 ビット，論理シフトします．

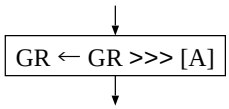
フラグ： SHLA 命令と同様です．

ニーモニック： `SHRL GR`
SHLA 命令と同様です．

命令フォーマット： SHRL 命令は 1 バイトの命令です．XR フィールドは必ず 11 にします．

第 1 バイト	
OP	GR XR
1001_2	GR 11

フローチャート： SHRL 命令は次のように描くことにします．なお，この書き方は，Java 言語のシフト演算子を真似たものです．



使用例： A，B 番地のデータを 1 ビット右にシフトし，C，D 番地に格納する例です．右に 1 ビットシフトすることは，÷ 2 を計算したのと同じ結果になります．論理シフトの場合は，128 以上の大きな数でも正しく 1/2 の値になります．

番地	機械語	ラベル	ニーモニック	
00	10 0B		LD	G0,A
02	93		SHRL	G0
03	20 0D		ST	G0,C
05	14 0C		LD	G1,B
07	97		SHRL	G1
08	24 0E		ST	G1,D
0A	FF		HALT	
0B	7F	A	DC	127
0C	80	B	DC	128
0D	00	C	DS	1
0E	00	D	DS	1

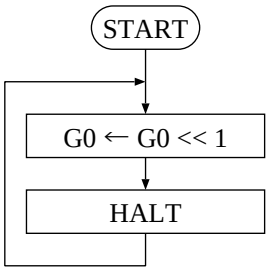
例題 5-4 シフトの動作確認

問題： 4 種類のシフト命令に付いて，G0 レジスタの値をシフトしながら動作を確認しなさい．

プログラム： 次のプログラムを作成します．

番地	機械語	ラベル	ニーモニック
00	90	L0	SHLA G0
01	FF		HALT
02	A0 00		JMP L0

フローチャート： フローチャートで描く場合は，次のように HALT を表現することにします．



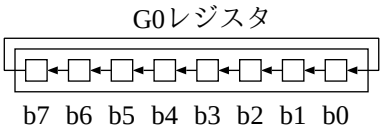
実行方法： 次の手順で実行します．

1. PC を 00_{16} にする．
2. G0 レジスタに適当なデータを書き込む．
3. ロータリースイッチを G0 に合わせたまま，RUN ボタンを押す．
4. G0 の値がシフトされたことが，データランプで確認できる．
5. 再度，RUN ボタンを押す．
6. G0 の値が更にシフトされる．

一度 RUN ボタンを押すと HALT 命令で停止します．そのとき，PC は JMP 命令を指していますので，再度 RUN ボタンを押すと，もう一度，最初からプログラムが実行されます．

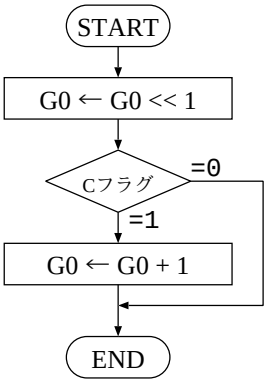
例題 5-5 ビットの回転

問題： 次の図のように，G0 レジスタの値を 1 ビット左回転するプログラムを作りなさい．



考え方： TeC には，ビットを回転する命令がありません．次の手順で目的を達成します．

1. シフト命令で G0 を 1 ビットシフトする．
2. はみ出したビットが C フラグに保存されているので，C フラグを調べる．
3. C フラグが 1 だった場合は，G0 最下位ビットを 1 にする．



プログラム： プログラムにすると次のようになります．例題 5-4 と同様に，最後に JMP 命令を追加しました．

番地	機械語	ラベル	ニーモニック
00	91	L0	SHLL G0
01	A8 05		JC L1
03	A0 07		JMP L2
05	30 0A	L1	ADD G0,ONE
07	FF	L2	HALT
08	A0 00		JMP L0
0A	01	ONE	DC 1

実行方法： 例題 5-4 と同様です．実行する度に，データランプの表示が回転します．

例題 5-6 シフトを用いた高速乗算

問題： シフトを用いて，A の 10 倍の値を B に求めるプログラムを作りなさい．

考え方： TeC には，かけ算命令がありません．そのため，繰り返しでかけ算をする方法を，以前，紹介しました．ここでは，繰り返しによる方法より高速なかけ算を紹介します．

× 10 は，次の式のように変形できます．

$$B = A \times 10 = A \times 8 + A \times 2$$

× 2 は，1 ビット左にシフトすることと同じです．× 8 は，3 ビット左にシフトすることと同じです．シフトした結果同士を足し合わせることにより，10 倍を計算することができます．

プログラム： プログラムにすると次のようになります．TMP は，一時的に A の 2 倍の値を記憶するために使用します．

番地	機械語	ラベル	ニーモニック	
00	10 0C		LD	G0,A
02	90		SHLA	G0
03	20 0E		ST	G0,TMP
05	90		SHLA	G0
06	90		SHLA	G0
07	30 0E		ADD	G0,TMP
09	20 0D		ST	G0,B
0B	FF		HALT	
0C	03	A	DC	3
0D	00	B	DS	1
0E	00	TMP	DS	1

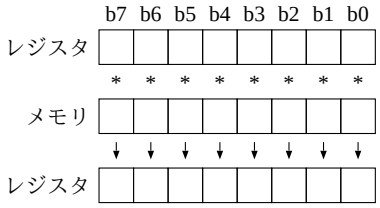
問題

1. A の値の 7 倍を B に求めるプログラムを作りなさい．
2. A のデータが符号無し 2 進数の場合，A の 1/4 を B に求めるプログラムを作りなさい．

5.9　論理演算命令

5.9.1　AND(Logical AND) 命令

AND 命令は，レジスタとメモリデータのビット毎の論理積 (AND) を計算します．ビット毎の論理積とは，次の図のように，レジスタとメモリの対応するビット同士の論理積の計算のことです．計算結果はレジスタに格納されます．



意味： ビット毎の論理積を計算します．結果は元のレジスタに格納します．(メモリの値は変化しません．)

フラグ： C フラグは常に 0 になります．S，Z フラグは計算の結果により変化します．

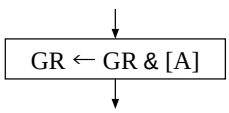
ニーモニック： `AND GR,A[,XR]`

`GR` は，計算の対象になるレジスタを表します．このレジスタの値とメモリの値の論理積が計算されます．結果はこのレジスタに格納されます．`A` と `[,XR]` の意味は LD 命令と同様です．

命令フォーマット： AND 命令は 2 バイトの長さを持ちます．各フィールドの意味は，LD 命令と同様です．

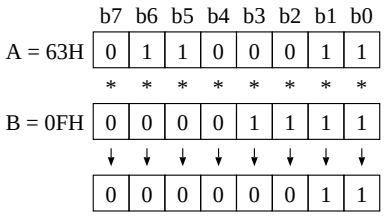
第 1 バイト		第 2 バイト
OP	GR XR	
0110_2	$GR\ XR$	aaaa aaaa

フローチャート： AND 命令は次のように描くことにします．これも，Java 言語や C 言語の演算子を真似したものです．



使用例： A 番地のデータと B 番地のデータの論理積を計算し，C 番地に格納するプログラムの例を示します.

番地	機械語	ラベル	ニーモニック
00	10 07		LD GO,A
02	60 08		AND GO,B
04	20 09		ST GO,C
06	FF		HALT
07	63	A	DC 63H
08	0F	B	DC 0FH
09	00	C	DS 1



AND 命令の応用： AND 命令は，データの特定のビットを 0 にクリアしたり，データの特定のビットの 1/0 を確かめたり，データの一部のビットだけを取り出したりするために使用できます.

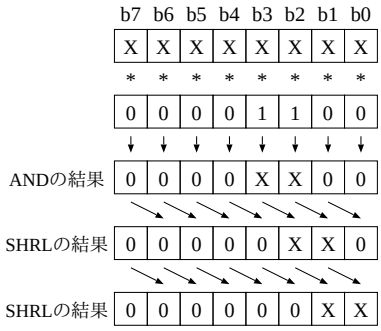
1. G0 の最下位ビット (LSB) がゼロなら L1 へジャンプする.

ラベル	ニーモニック
	...
	AND GO,ONE
	JZ L1
	...
L1	...
	...
ONE	DC 01H

   01_{16} との AND の結果が 00_{16} になることから，LSB が 0 だったことが分かります.

2. G0 の，b3,b2 の 2 ビットを右詰めにして取り出す.

ラベル	ニーモニック
	...
	AND GO,MSK
	SHRL GO
	SHRL GO
	...
MSK	DC 0CH



3. G0 の値を 8 の倍数になるように切り捨てる.

ラベル	ニーモニック
	...
	AND GO,MSK
	...
MSK	DC F8H

   下 3 ビットをゼロにすると，8 の倍数になります.

4. G0 の値を 8 で割った余りを求める.

ラベル	ニーモニック
	...
	AND GO,MSK
	...
MSK	DC 07H

   下 3 ビットだけを取り出すと，8 で割った余りになります.

   このように，2 の累乗 (2,4,8,16,...) で割った商や余りは，シフトや論理演算で簡単に計算できます．また，2 の累乗の倍数に切捨てたり，切り上げたりする計算も，論理演算を使うと簡単にできます。

16 進数の表記
ニーモニック中で数値を書くとき，16 進数で書き表したいことがあります．そのときは，前のプログラム中の DC 命令のように，数値の後ろに “H” を付けます.

5.9.2　OR(Logical OR) 命令

OR 命令は，レジスタとメモリデータの対応するビット同士の論理和 (OR) を計算します．AND 命令の論理和版です．計算結果はレジスタに格納されます．

意味： ビット毎の論理和を計算します．結果は元のレジスタに格納します．(メモリの値は変化しません．)

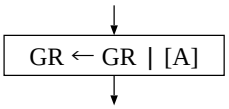
フラグ： C フラグは常に 0 になります．S，Z フラグは計算の結果により変化します．

ニーモニック： `OR GR,A[,XR]`
`GR` は，計算の対象になるレジスタを表します．このレジスタの値とメモリの値の論理和が計算されます．結果はこのレジスタに格納されます．`A` と `[,XR]` の意味は LD 命令と同様です．

命令フォーマット： OR 命令は 2 バイトの長さを持ちます．各フィールドの意味は，LD 命令と同様です．

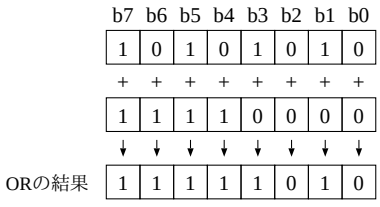
第 1 バイト		第 2 バイト
OP	GR XR	
0111_2	$GR\ XR$	aaaa aaaa

フローチャート： OR 命令は次のように描くことにします．これも，Java 言語や C 言語の演算子を真似したものです．



OR 命令の応用： OR 命令は，データの特定のビットを 1 にするために使用できます．次の例は，G0 の上 4 ビットを全て 1 にします．

ラベル	ニーモニック	
	...	
	LD	G0,DATA
	OR	G0,MSK
	...	
DATA	DC	0AAH
MSK	DC	0F0H



5.9.3　XOR(Logical XOR) 命令

XOR 命令は，レジスタとメモリデータの対応するビット同士の排他的論理和 (XOR) を計算します．計算結果はレジスタに格納されます．

意味： ビット毎の排他的論理和を計算します．結果は元のレジスタに格納します．(メモリの値は変化しません．)

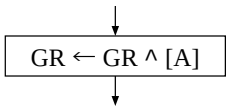
フラグ： C フラグは常に 0 になります．S，Z フラグは計算の結果により変化します．

ニーモニック： `XOR GR,A[,XR]`
`GR` は，計算の対象になるレジスタを表します．このレジスタの値とメモリの値の排他的論理和が計算されます．結果はこのレジスタに格納されます．`A` と `[,XR]` は LD 命令と同様です．

命令フォーマット： XOR 命令は 2 バイトの長さを持ちます．各フィールドの意味は，LD 命令と同様です．

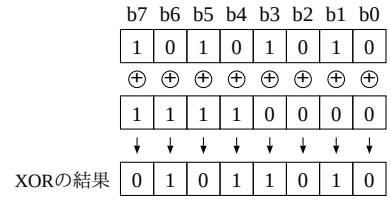
第 1 バイト		第 2 バイト
OP	GR XR	
1000_2	$GR\ XR$	aaaa aaaa

フローチャート： XOR 命令は次のように描くことにします．



XOR 命令の応用： XOR 命令は，データの特定のビットの 0/1 を反転するために使用できます．TeC には NOT 命令は用意されていませんが，XOR 命令を NOT 命令の代用にすることができます．次の例は，G0 の上 4 ビットの 0/1 を反転する例です．

ラベル	ニーモニック	
	...	
	LD	G0,DATA
	XOR	G0,MSK
	...	
DATA	DC	0AAH
MSK	DC	0F0H



5.10 アドレッシングモード

LD, ST, ADD, SUB, CMP, AND, OR, XOR, JMP, JZ, JC, JM 命令は，どれも次のような同じフォーマットでした．これまで，これらの命令の XR 部分は 00 にしてきました．

第 1 バイト		第 2 バイト
OP	GR XR	
OP	$GR\ XR$	aaaa aaaa

XR に 00 以外を指定することにより，メモリアドレス表現方法を変更することができます．このアドレス表現方法のことをアドレッシングモードと呼びます．

TeC で使用できるアドレッシングモードは，XR の値により次の 4 種類があります．

XR	意味	
00	ダイレクトモード	直接モード
01	G1 インデクスドモード	指標モード
10	G2 インデクスドモード	指標モード
11	イミディエイトモード	即値モード

(同じ表が命令表にも掲載されているので，そちらも確認してください．)

以下では LD 命令と ST 命令を例に，4 種類のアドレッシングモードを説明します．ここでは説明しませんが，ADD, SUB, CMP, AND, OR, XOR 命令でも，LD 命令と ST 命令と同様にアドレッシングモードが使用できます．

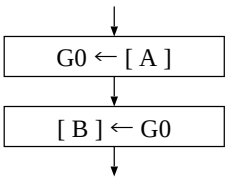
5.10.1 ダイレクト (直接) モード

これまで使用してきたのは，このモードです．命令の第 2 バイトがメモリアドレスを直接に表します．

ニーモニックでは次のように書いてきました．ここで，A，B は，データを置いたメモリのアドレスです．

```
LD   G0,A
ST   G0,B
```

フローチャートでは，次のように描きました．



5.10.2 インデクスド (指標) モード

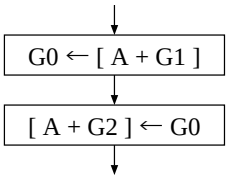
G1 または G2 レジスタの値と，命令の第 2 バイトの値の合計がメモリアドレスを表します．XR の値により，G1，G2 どちらのレジスタを使用するか決まります．G0 レジスタは使用できないので注意してください．

このモードは，配列データをアクセスするとき使用できます．また，ジャンプ命令でも使用できますが，ジャンプ命令での使用は高度なのでここでは説明しません．

ニーモニックでは次のように書きます．このプログラムは，$A + G1$ 番地のデータを G0 に読込み，$A + G2$ 番地に格納しています．

```
LD   G0,A,G1
ST   G0,A,G2
```

フローチャートでは，次のように描きます．



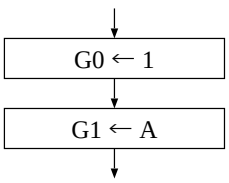
5.10.3 イミディエイト (即値) モード

第 2 バイトがデータそのものになるモードです．ST 命令やジャンプ命令では使用できません．(これらの命令で XR を 11 にすると，CPU が命令を実行しようとしたとき，命令コードのエラーになり RUN ランプが点滅します．)

ニーモニックでは次のように書きます．

```
LD   G0,#1
LD   G1,#A
```

最初の LD 命令は，G0 を 1 にします．次の LD 命令は，G1 を A の番地 (内容ではない) にします．フローチャートでは次のようになります．



5.10.4　アドレッシングモードの使用例

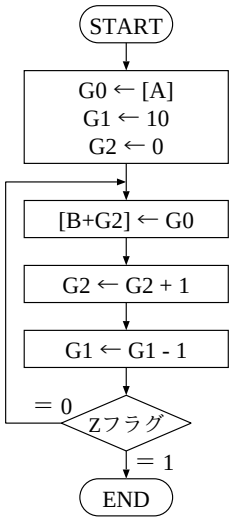
A 番地のデータで，B 番地からの 10 バイトの配列を初期化するプログラムの例を示します．

G1，G2 レジスタを初期化する部分，G1，G2 レジスタの値を±1 する部分にイミディエイトモードを使用しました．

また，10 回の繰り返しで 10 バイトの領域を初期化できるよう，ST 命令の格納アドレスが G2 の値で 1 番地ずつずれていきます．ずらすためにインデクスモードを使用しました．

番地	機械語	ラベル	ニーモニック	
00	10 11		LD	G0,A
02	17 0A		LD	G1,#10
04	1B 00		LD	G2,#0
06	22 12	LOOP	ST	G0,B,G2
08	3B 01		ADD	G2,#1
0A	47 01		SUB	G1,#1
0C	A4 10		JZ	STOP
0E	A0 06		JMP	LOOP
10	FF	STOP	HALT	
11	AA	A	DC	0AAH
12	00 00	B	DS	10
14	00 00			
16	00 00			
18	00 00			
1A	00 00			

次に，このプログラムのフローチャートを示します．イミディエイトモードとインデクスドモードを，フローチャート上でどのように表現しているか，よく確認してください．



問題

1. これまでに出てきたプログラムを，イミディエイトモードを使用して書き換えなさい．
2. A 番地からの 5 バイトのデータの合計を，B 番地に求めるプログラムを作りなさい．
3. A 番地からの 5 バイトのデータを，B 番地からの 5 バイトにコピーするプログラムを作りなさい．

5.11 入出力

図 5.2 に示した TeC の構成のうち主記憶に付いては，LD，ST 命令を使用することでアクセスできることが分かりました．

ここで説明する入出力 (Input Output=略して I/O) 命令は，入出力インターフェース回路をアクセスするための命令です．読み込みを行う IN 命令，書き込みを行う OUT 命令の 2 種類があります．

5.11.1 I/O マップ

いくつか存在するインターフェース回路の中の，どれをアクセスするかは I/O アドレス (メモリアドレスとは別のもの) により指定します．TeC の I/O アドレスは 0_{16}〜F_{16} までの 16 番地です．I/O アドレスの一覧を「I/O マップ」と言います．「I/O マップ」は次の表のようなものです．

I/O マップ		
番地	Read	Write
0	データスイッチ	ブザー
1	データスイッチ	スピーカ
2	SIO 受信データ	SIO 送信データ
3	SIO ステータス	SIO コントロール
4	空き	空き
...	...	...
F	空き	空き

この表から，例えば次のことが分かります．(詳しくは，後の方で説明します．)

1. 0 番地の Read(読み込み) により，データスイッチの値を読み込むことができる．
2. 0 番地の Write(書き込み) により，ブザーにアクセスできる．
3. 1 番地の Read(読み込み) でも，データスイッチの値を読み込むことができる．
4. 1 番地の Write(書き込み) により，スピーカーにアクセスできる．

I/O マップは命令表の中にも掲載されていますので，確認してください．

5.11.2 IN(Input) 命令

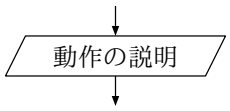
意味： 入出力インターフェース回路からデータをレジスタに入力します．I/O アドレスは 4 ビットで指定します．

ニーモニック： `IN GR,P`
`GR` は，値を入力するレジスタを表します．`P` は入出力インターフェース回路のアドレスです．

命令フォーマット： IN 命令は 2 バイトの長さを持ちます．XR フィールドは必ず 00 にします．第 2 バイトの上位 4 ビットも必ず 0000 にします．

第 1 バイト		第 2 バイト
OP	GR XR	
1100_2	GR 00	0000 pppp

フローチャート： IN 命令は次のように描くことにします．「動作の説明」は，各自が工夫して下さい．

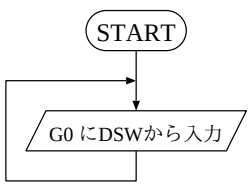


使用例： 次のプログラムはデータスイッチの状態を I/O アドレス 0 番地から読み取り，G0 レジスタに格納するものです．ロータリースイッチを G0 に合わせた状態で，プログラムを実行してください．データスイッチを変化させると即座にデータランプの表示が変化します．

プログラムは無限ループになっているので，実行を開始したら STOP ボタンを押すまで止まりません．

ラベル	ニーモニック
START	IN GO,0
	JMP START

フローチャートで描くと，次のようになります．



DSWは、データスイッチのこと

IN 命令の応用： 次のプログラムは，データスイッチから数値を入力し，合計を G0 に求めるものです．ロタリースイッチを G0 に合わせて実行すると，2 進数入力，2 進数表示の簡単な電卓として使用できます．

操作手順は次の通りです．

1. プログラムを入力する．
2. PC にプログラムの実行開始番地をセットする．
3. ロータリースイッチを G0 に合わせる．
4. データスイッチに，データをセットする．
5. RUN ボタンを押す．
6. データの件数分，4，5 を繰り返す．
7. データランプに合計が表示されている．

ラベル	ニーモニック	
START	LD	G0,♯0
LOOP	IN	G1,00H
	ST	G1,TMP
	ADD	G0,TMP
	HALT	
	JMP	LOOP

定数の定義 (EQU 命令)

IN 命令の使用例や応用例で示したプログラムでは，データスイッチの値を読み込むための I/O アドレス “0” が，プログラム中に直接書いてありました．データスイッチのことだと分かりやすい名前を用いるとプログラムが読みやすくなります．EQU 命令は，名前 (ラベル) を定義するための命令です．EQU 命令のオペランドの値を，ラベルに割り付けます．他の命令と異なり，機械語やデータを生成しません．

番地	機械語	ラベル	ニーモニック	
00		DSW	EQU	00H
00	C0 00	START	IN	G0,DSW
02	A0 00		JMP	START

5.11.3　OUT(Output) 命令

意味： レジスタのデータを，入出力インターフェース回路へ出力します．I/O アドレスは 4 ビットで指定します．

ニーモニック： `OUT GR,P`

`GR` は，値を出力するレジスタを表します．`P` は入出力インターフェース回路のアドレスです．

命令フォーマット： OUT 命令は 2 バイトの長さを持ちます．XR フィールドは必ず 11 にします．第 2 バイトの上位 4 ビットも必ず 0000 にします．

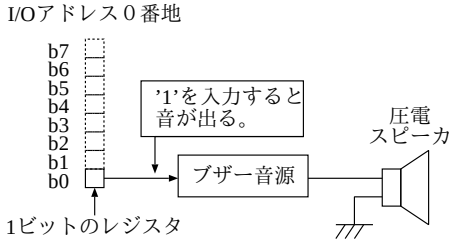
第 1 バイト		第 2 バイト
OP	GR XR	
1100_2	GR 11	0000 pppp

使用例： 次のプログラムはデータスイッチの値を読み取り，ブザー用のポートに出力するものです．

ブザーは次の図のような構造になっており，I/O アドレス 0 番地に書き込んだ値で鳴ったり止まったりします．プログラム実行中に，データスイッチを操作して最下位ビットを 1 にするとブザーが鳴ります．逆に最下位ビットを 0 にするとブザーの音が止まります．

プログラムは無限ループになっているので，実行を開始したら STOP ボタンを押すまで止まりません．ブザーが鳴っている状態でプログラムを停止しても，ブザーが鳴りっぱなしになります．その場合は，RESET スイッチを押して下さい．

ラベル	ニーモニック	
DSW	EQU	00H
BUZ	EQU	00H
START	IN	G0,DSW
	OUT	G0,BUZ
	JMP	START



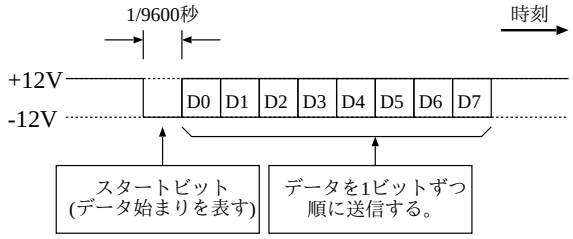
5.12 TeC シリアル入出力 (SIO)

TeC には，パソコンと接続できるシリアル入出力 (Serial Input Output : 略して **SIO**) インターフェースが備えてあります.

前出の IN，OUT 命令を使用して，SIO インターフェースにアクセスします.

5.12.1 シリアル入出力

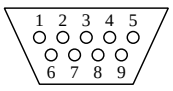
名前の通り (Serial ＝直列) データを 1 ビット毎，転送する方式です. 次の図のように，時間による電圧の変化としてデータを表現します. TeC は，$\pm 12V$ を使用して，1/0 を表現します. また，1 ビットの時間として 1/9600 秒を用います. 1/9600 秒で 1 ビットを送る通信速度を 9600bau(ボー) と言います.



5.12.2 入出力用のコネクタ

TeC は，図 4.1 の「RS-232C コネクタ」をシリアル入出力用に使用します. RS-232C コネクタのことを「シリアルポート」と呼ぶこともあります. TeC は，RS-232C コネクタの 3 本の信号線を次の図のように使用します.

RS-232Cコネクタ



2番:RxD受信データ
3番:TxD送信データ
5番:GND

5.12.3 PC との接続

PC と接続するときは，PC の裏にある RS-232C コネクタと**クロスケーブル**を用いて接続します. クロスケーブルで接続することにより，PC の送信データと TeC の受信データ，TeC の送信データと PC の受信データが接続され双方向の通信ができるようになります.

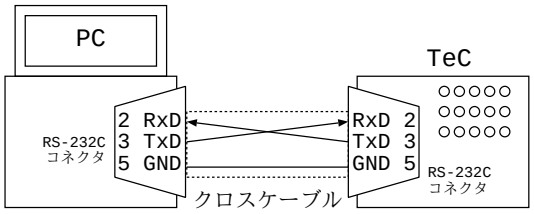
クロスケーブル

TeC と PC を接続するために使用するクロスケーブルについて説明します.

クロスケーブルは，写真のようなケーブルです. ケーブルの内部では，図のように配線が交差 (クロス) しています.

このような配線になっているので，TeC の送信データのピンと PC の受信データピン，PC の送信データのピンと TeC の受信データピンが接続され，TeC と PC の間で双方向のデータ通信が可能になります.

PCとTeCの接続



ニーモニック中の文字データ表記

文字を扱うプログラムを書くとき，文字コードが必要になります. そのときは，ニーモニック中に数値の代わりに,「'A'」のように書くことができます. 文字 A の文字コード (41H) と言う意味です. (文字コードには ASCII コードを用います.)

ニーモニック中の'A' は，41H または 65 と書いたのと全く同じ意味になります.

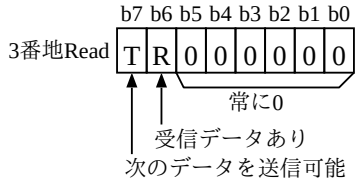
DC 命令では，連続した文字の定義に"文字列"を使用できます. 下のプログラム中にあるように，"TeC"の表記で 3 バイトのデータが生成されます.

番地	機械語	ラベル	ニーモニック
00	13 41	START	LD GO,#'A'
02	FF		HALT
03	42 43		DC 'B','C'
05	45 65		DC "TeC"
07	43		

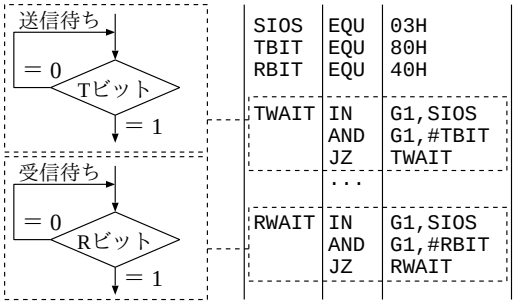
5.12.4　I/O ポート

SIO の I/O ポートは，2 番地，3 番地に配置されています (I/O マップ参照)．内容は，次の通りです．

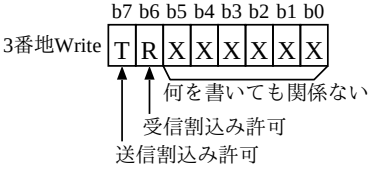
1. 受信データ
   2 番地を，IN 命令で読むと受信データを読み出すことができます．
2. 送信データ
   2 番地に，OUT 命令で送信データを書き出すことができます．
3. ステータス
   3 番地を，IN 命令で読むと次図のようなデータを読み込むことができます．各ビットの意味も図の通りです．



プログラムは，次のプログラムのような手順でステータスを調べ，データの入出力が可能か判断する必要があります．この判断は頻繁に使用するので，この本の中では，フローチャートを簡単化し次のように描くことにします．

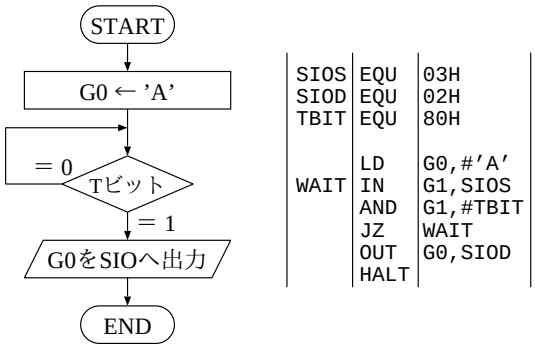


4. コントロール
   3 番地へ，OUT 命令で次図のようなデータを書き込みます．現在のところこの機能は使用しません．詳細は，6 章で説明します．



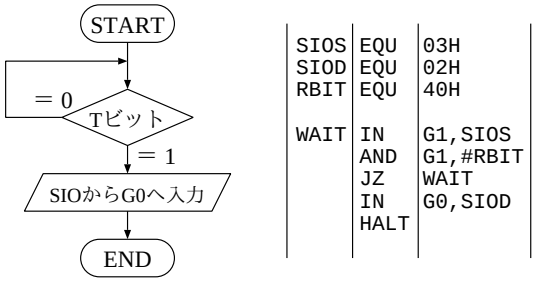
5.12.5　シリアル出力プログラム

シリアル出力を行うプログラムの，一般的なフローチャートを次の図に示します．I/O アドレス 3 番地の「T ビット」が “1” になり，送信可能になるのを待って出力データを 2 番地に書き込みます．T ビットが 1 か 0 か調べるために，AND 命令を使用しています．これで，(TeC が PC とクロスケーブルで接続してあれば，) 2 番地に書き込んだデータが PC に送信されます．



5.12.6　シリアル入力プログラム

シリアル入力を行うプログラムの，一般的なフローチャートを次の図に示します．I/O アドレス 3 番地の「R ビット」が “1” になり，受信データが届くのを待って 2 番地からデータを読み込みます．受信したデータは，G0 に格納されます．



ターミナルプログラム
UNIX 上の **tip プログラム**，Windows 上の**ハイパーターミナル**等がターミナルプログラムの一種です．ターミナルプログラムは，シリアルポートから受信したデータを ASCII 文字コードとみなし，対応する文字を画面に表示します．また，キーボードが押されると，押された文字の ASCII 文字コードをシリアルポートから送信します．
TeC の SIO を使用するときは，TeC と PC をクロスケーブルで接続するだけでなく，PC にターミナルプログラムを起動し，PC のシリアルポートが使用できるように準備する必要があります．

5.12.7 シリアル入出力データ

TeC が PC と通信するときは,「2.6 文字の表現」で勉強した ASCII コードを使用する約束とします. PC のキーボードを叩くと，対応する文字の文字コードが TeC に送信され，TeC が文字コードを送信すると，対応する文字が PC の画面に表示されます. (PC 側には，このように動作する**ターミナルプログラム**を動かしておく必要があります．)

ASCII 文字コード表

(下位4ビット) \ (上位4ビット)	0	1	2	3	4	5	6	7
0		DE(SP)	0	@	P	‘	p	
1	SH	D1	!	1	A	Q	a	q
2	SX	D2	"	2	B	R	b	r
3	EX	D3	#	3	C	S	c	s
4	ET	D4	$	4	D	T	d	t
5	EQ	NK	%	5	E	U	e	u
6	AK	SN	&	6	F	V	f	v
7	BL	EB	’	7	G	W	g	w
8	BS	CN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SB	*	:	J	Z	j	z
B	HM	EC	+	;	K	[	k	{
C	CL	→	,	<	L	\	l	|
D	CR	←	-	=	M	]	m	}
E	SO	↑	.	>	N	^	n	~
F	SI	↓	/	?	O	_	o	DEL

ターミナルプログラム画面の改行

ターミナルプログラムの表示を改行するには，復帰コード (CR) と改行コード (LF) の 2 文字を TeC から PC へ送るのが一般的です.
復帰コード (CR:文字コードは 0DH) は文字カーソルを左端に移動するための文字，改行コード (LF:文字コードは 0AH) は文字カーソルを一行下に移動するための文字です.
しかし，実際に UNIX の tip プログラムで試してみると，改行コードのみの送信で復帰と改行の両方ができてしまいました．(復帰コードを送っても害はありません．)
ターミナルプログラムにより，必要な復帰コードと改行コードの組合せが変化するようです．自分が実際に使用するシステムに合わせて，臨機応変に対応してください.

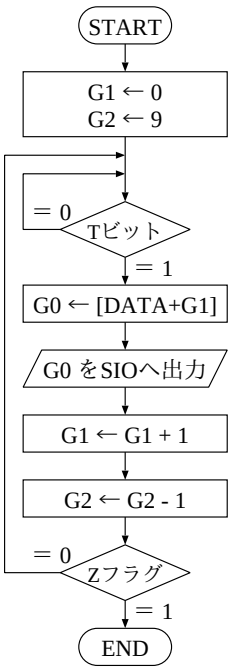
例題 5-7 文字列出力 1

問題： SIO へローマ字で自分の名前を出力するプログラムを作りなさい.

考え方： 筆者の場合，SHIGEMURA の 9 文字を出力するプログラムを作成します.

メモリ上にデータとして 9 文字を準備しておき，インデクスドモードのアドレッシングを使用し順に出力します.

フローチャート： 次のようになります.



プログラム：

番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	17 00	START	LD	G1,#0
02	1B 09		LD	G2,#9
04	C0 03	LOOP	IN	G0,SIOS
06	63 80		AND	G0,#80H
08	A4 04		JZ	LOOP
0A	11 17		LD	G0,DATA,G1
0C	C3 02		OUT	G0,SIOD
0E	37 01		ADD	G1,#1
10	4B 01		SUB	G2,#1
12	A4 16		JZ	END
14	A0 04		JMP	LOOP
16				
16	FF	END	HALT	
17				
17	53 48	DATA	DC	"SHIGEMURA"
19	49 47			
1B	45 4D			
1D	55 52			
1F	41			

例題 5-8 文字列出力 2

問題： SIO へローマ字で “Tokuyama Kousen”, “Shigemura” のような 2 行を出力するプログラムを作りなさい.

考え方： 繰り返しにより，1 文字ずつ SIO へ出力します．1 行目データと 2 行目データの間に，復帰コードと改行コードを置きます．これらのコードも通常の文字と同じように扱って構いませんので，プログラムは 2 行分を一気に出力するもので構いません.

プログラム： プログラムにすると次のようになります．データの最後の 00H は，データの終わりを示す目印です．文字コードが 00H の文字は通常使用しませんので，これを文字ではなく終わりの印に使用しました.

この方法だと，出力するデータの長さが変化してもプログラムを変更する必要がありません.

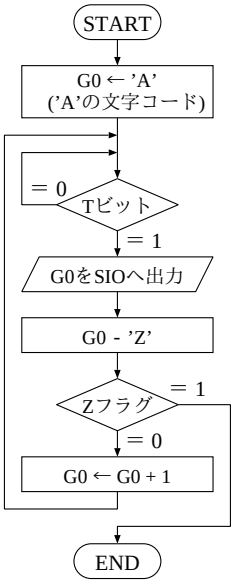
番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	17 00		LD	G1,♯00H
02	19 15	L0	LD	G2,DATA,G1
04	5B 00		CMP	G2,♯0
06	A4 14		JZ	END
08	C0 03	L1	IN	G0,SIOS
0A	63 80		AND	G0,♯80H
0C	A4 08		JZ	L1
0E	CB 02		OUT	G2,SIOD
10	37 01		ADD	G1,♯1
12	A0 02		JMP	L0
14				
14	FF	END	HALT	
15				
15	54 6F	DATA	DC	"Tokuyama
17	6B 75			Kousen"
19	79 61			
1B	6D 61			
1D	20 4B			
1F	6F 75			
21	73 65			
23	6E			
24	0D 0A		DC	0DH,0AH
26	53 68		DC	"Shigemura"
28	69 67			
2A	65 6D			
2C	75 72			
2E	61			
2F	0D 0A		DC	0DH,0AH
31	00		DC	00H

例題 5-9 ‘A’〜‘Z’ の文字を表示

問題： SIO へ ‘A’〜‘Z’ の文字を連続して出力するプログラムを作りなさい.

考え方： ‘A’〜‘Z’ の文字コードは連続しています (ASCII 文字コード表参照)．最初に ‘A’ の文字コードをレジスタにロードし SIO へ出力します．次に，レジスタに 1 加えて文字コードを増やします．増やした結果は ‘B’ の文字コードですので，それを出力します．後は同様に ‘Z’ の文字コードまで出力します.

フローチャート： 次の通りです.



プログラム： G0 レジスタを文字コードの格納に，G1 レジスタを「T ビット」のチェックに使います．出力可能になったら G0 の文字コードを SIO へ出力します．出力した後で出力した文字を調べ，‘Z’ だっら終了します.

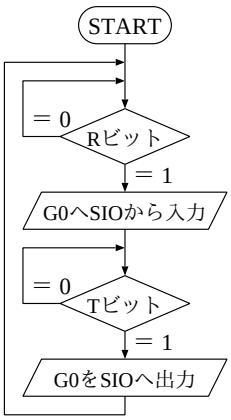
番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	13 41		LD	G0,♯’A’
02	C4 03	WAIT	IN	G1,SIOS
04	67 80		AND	G1,♯80H
06	A4 02		JZ	WAIT
08	C3 02		OUT	G0,SIOD
0A	53 5A		CMP	G0,♯’Z’
0C	A4 12		JZ	OWARI
0E	33 01		ADD	G0,♯1
10	A0 02		JMP	WAIT
12	FF	OWARI	HALT	

例題 5-10 echo プログラム

問題： SIO から入力した文字をそのまま SIO へ送り返す (こだま) プログラムを作りなさい.

考え方： STOP ボタンが押されるまで終了しないプログラムを作ります.

フローチャート： 次のようなアルゴリズムでできるはずです.



プログラム： G0 レジスタを文字コードの格納用に，G1 レジスタを「R ビット」,「T ビット」のチェック用に使います．IN 命令で G0 へ読み込んだ文字コードを，OUT 命令で書き戻すことにより文字を送り返します.

番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	C4 03	START	IN	G1,SIOS
02	67 40		AND	G1,♯40H
04	A4 00		JZ	START
06	C0 02		IN	G0,SIOD
08	C4 03	WAIT	IN	G1,SIOS
0A	67 80		AND	G1,♯80H
0C	A4 08		JZ	WAIT
0E	C3 02		OUT	G0,SIOD
10	A0 00		JMP	START

問題

SIO から 1 文字入力し，入力した文字がアルファベット小文字の場合大文字に変換し，そうでなければそのまま SIO へ出力するプログラムを作りなさい.

第6章　高度なプログラミング

6.1　クロス開発

第5章では，ハンドアセンブルにより機械語のプログラムを作成しました．プログラムをニーモニックから機械語に手作業で変換する際，ミスが多発しました．このようにミスが多発するので，実は，ハンドアセンブルによるプログラムの開発は，ほとんど行われません．

この授業の中では，原理をよく理解してもらうために，あえて面倒なハンドアセンブルをたくさんしてもらいました．そろそろ，機械語の原理はよく分かってきたので，楽にプログラミングする方法を教えます．その代わりに，これまでより，たくさんのプログラムを作成してもらいます．

プログラムを作成することを，プログラムを「開発する」と言います．楽に「開発する」ために使用するコンピュータやプログラムのことを,「開発環境」と言います.「開発環境」には,「セルフ開発環境」と「クロス開発環境」があります．

「セルフ開発環境」は，UNIX や Windows 上で Java 言語や C 言語でプログラムを作成し実行するような，プログラムを開発するコンピュータと実行するコンピュータが同じ場合に用いる「開発環境」です．

「クロス開発環境」は，TeC のような小さなコンピュータのプログラム開発によく用いられます．他のコンピュータ上でプログラムを作成し，出来上がったプログラムを目的のコンピュータに転送して実行するような「開発環境」です.「クロス開発」環境を用いたプログラムの開発は,「クロス開発」と呼ばれます．

ここでは,「TeC クロス開発環境」について簡単に説明します．詳細は,「付録 B TeC6 クロス開発環境」を参照してください．

6.1.1　アセンブラ

ニーモニックから機械語に変換する作業を，これまでは，手作業 (ハンドアセンブル) により行ってきました．このような単純で機械的な作業はコンピュータが得意なものです．この作業を行うプログラムを「アセンブラ」と言います．**アセンブラは，ニーモニックで記述したプログラムを，機械語に変換します．**

TeC 用アセンブラの使用方法

TeC 用のアセンブラは，UNIX または，WindowsXP 上で動作する「クロスアセンブラ」です．インストールの方法，使用方法の詳細は，TeC 付属の CD-ROM に格納された `ReadMe.txt` ファイルを参照してください．以下では，UNIX 上にインストールされていることを前提に説明します．

クロスアセンブラを使用したプログラムの作成手順は，次の通りです．

1. テキストエディタを用いて，プログラムのニーモニックを入力します．プログラムを格納するファイルの拡張子は，“`.t6`” でなければなりません．次に，プログラムの入力例を示します．

```
      $ emacs xxx.t6
1:  ; 入力例
2:      ld g0,a
3:      st g0,b
4:      halt
5:  a   dc 10
6:  b   ds 1
```

‘;’ は，注釈 (コメント) の始まりを表します．‘;’ から行末までが注釈になります．1 行目が，注釈の例です．ラベルのない行 (2，3，4 行) は，一つ以上の空白を書いた後，命令を書きます．ラベルのある行 (5，6) は，行の 1 文字目からラベルを書きます．なお，例では命令のインデントが揃っていますが，揃える必要はありません．

2. アセンブラを実行します．アセンブラのコマンド名は，`tasm6` です．実行例は，次の通りです．

```
$ tasm6 xxx.t6
アセンブル成功
結果は [xxx.lst] と [xxx.bin] に格納しました.

[xxx.lst]
ADR   CODE      Label   Instruction

00              ; 入力例
00    10   05           LD      G0,A
02    20   06           ST      G0,B
04    FF                HALT
05    0A        A       DC      10
06    00        B       DS      1
```

入力にミスが無ければ，アセンブルリストが画面に表示され，同時に，拡張子 “.lst” のファイルにアセンブルリストが，拡張子 “.bin” のファイルに機械語が格納されます．

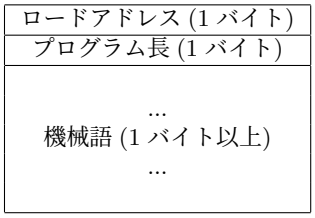
入力にミスがあった場合は，例えば次のようなエラーメッセージが表示され，エラーの場所と原因が分かるようになっています．

```
$ tasm6 yyy.t6
＊＊＊ エラー [未知のニーモニック] ＊＊＊
エラー発生場所 :  ファイル [yyy.t6] の 3 行で
エラー行の内容 :  [ SL G0,B]
エラートークン :  [SL]
```

TeC 用アセンブラの出力

アセンブルリスト： 画面と拡張子 “.lst” のファイルに，アセンブルリストが出力されます．アセンブルリストは，入力したニーモニックと，それから作られた機械語の対応を分かり易く表示したものです．これまでハンドアセンブルで行っていた機械語作成の結果と，同じ内容になっていることを，確認してください．

機械語プログラムファイル： 拡張子 “.bin” のファイルに機械語が格納されます．機械語プログラムファイルの内容は，次の図に示すようになっています．



ファイルの 1 バイト目は，機械語プログラムの TeC 主記憶上の配置アドレスです．機械語は，このバイトで表される番地を先頭にして主記憶に書き込まれます．ロードアドレスは，アセンブラの ORG 命令で指示することができます．ORG 命令を使用しない場合のロードアドレスは，0 番地になります．ファイルの 2 バイト目は，機械語プログラムの長さです．3 バイト目以降の機械語の長さをバイト単位で表します．

6.1.2　ダウンロード

アセンブラを実行することにより，拡張子 “.bin” のファイルに機械語が格納されました．次に，この機械語を TeC の主記憶にダウンロードします．

転送にはパソコンと TeC をシリアルケーブルで接続した上で，UNIX 上で tsend6 コマンド，TeC 上で IPL(Initial Program Loader:付録 B.8 参照) プログラムを実行します．tsend6 コマンドと，IPL プログラムが通信して UNIX から TeC にプログラムが転送されます．

シリアルケーブルの接続方法は,「5.12 TeC シリアル入出力」を参照してください．また，tsend6 コマンドのインストール方法や WindowsXP 上での使用方法は，TeC 付属の CD-ROM に格納された ReadMe.txt ファイルを参照してください．以下では，UNIX 上にインストールされていることを前提に説明します．

ダウンロード手順は，次の通りです．

1. UNIX 上で tsend6 プログラムを起動します．下の実行例のように，機械語が格納されたファイルを指定します．TeC を受信状態にするように指示が表示されます．

2. TeC 上で IPL プログラムを起動します．IPL は，E0H 番地から FFH 番地の ROM 領域 (「付録 C 命令表」のメモリマップを参照) に予め格納されています．PC に E0H をセットし，RUN ボタンを押すことで IPL を起動します．

3. tsend6 プログラムに Enter を入力します．

```
$ tsend6 xxx.bin
TeC6 を受信状態にして Enter キーを押して下さい.
[00][06][10][04][20][05][0a][00]
```

以上の操作で，コンソールパネルから機械語を打ち込むこと無く，機械語が TeC の主記憶の 00H 番地から始まる領域に書き込まれました．主記憶に書き込まれたプログラムを実行する方法は，コンソールパネルから機械語を打ち込んだ場合と同様です．

問題

1. 例題 5-5「ビット回転プログラム」を，クロス開発環境を用いて作成・実行しなさい.
2. 例題 5-3「割算を計算するプログラム」を，クロス開発環境を用いて作成・実行しなさい.
3. 例題 5-8「文字列出力プログラム」を，クロス開発環境を用いて作成・実行しなさい.
   注意：同じシリアルポートとケーブルを,「開発環境」と「文字列出力プログラム」の両方で使用します．プログラムのダウンロード時には，ターミナルプログラム (UNIX の tip や, Windows のハイパーターミナル) は終了しておく必要がありますので，注意してください.

ニーモニック記述上の注意

- TeC のアセンブラでは，アルファベット大文字と小文字は区別されません．小文字で記述しても，アセンブルリストには大文字に変換されて表示されます.
- ‘A’〜‘F’ の文字で始まる 16 進数は，前に ‘0’ を付加する必要があります．アルファベットで始まる 16 進数は，ラベルと区別ができないためです．(例 `F1H` → `0F1H`)
- 空白の表現には，スペースとタブのどちらも使用できます．インデントを揃えるにはタブを使用すると便利です．また，空白の文字数はいくつでも構いません．読みやすいように，上手に空白を使用してください.
- ‘;’ をニーモニックの中に書くと，次の文字から行末までが注釈 (コメント) になります．上手に注釈を入れて，読みやすいプログラムにしてください.
- 文字や文字列の表記も，これまでニーモニックの記述に使用してきた ‘文字’ や “文字列” の記述が使用できます．実際に使用する記号は,「’」(パソコンの日本語キーボードでは, シフトを押しながら「7」) と,「"」(シフトを押しながら「2」) です．本書の記述の中では，左と右のクォートに異なる記号を用いている場合もありますが，実際には同じ記号です.

6.2 スタック

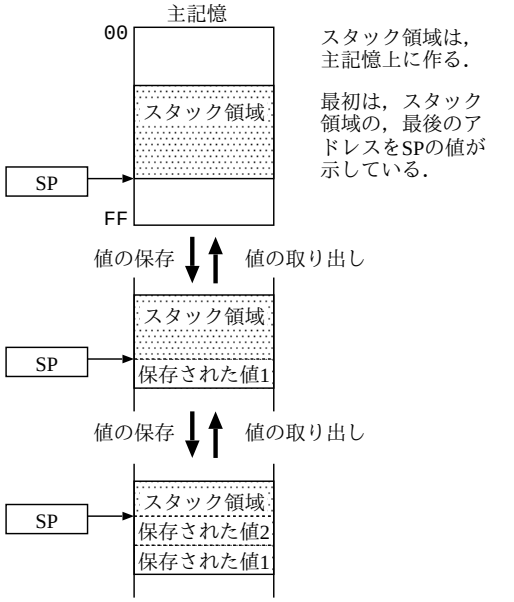
スタックは，一時的なデータをメモリ上に保管するのに都合の良い仕組みです．例えば，レジスタの個数が足らなくなったとき，レジスタを別の目的に使用する前に値をスタックに保存します．レジスタを使用し終わったらスタックから値を復元します.

同じことを，ST 命令と LD 命令と「決まった領域」を用いて行うことも可能ですが，スタックを利用すると「決まった領域」を使用する必要がありません．スタックもメモリ上の領域ですが，名前を付けたりすることなしに，プログラムのあちこちから同じ領域を繰り返し使用することができます.

6.2.1 仕組み

スタックは，CPU のスタックポインタ (SP レジスタ) により管理されます．最初，SP は，スタック領域の最後の番地を記憶しています．スタックに値を保存するときは，SP の値から 1 を引き新しいアドレスを決め，そのアドレスに値を保存します．値を保存する度に SP の値は減少していきます.

スタックから値を取り出すときは，SP の示すアドレスの値を取り出し，次に SP の値に 1 を加えます．値を取り出す度に SP の値は増加していきます.



このように，スタックを用いると，保存したのとは逆の順番で値を取りだすようになります．一番最後に入れたデータを，一番最初に取り出すので，スタックは LIFO(Last In First Out) 方式のデータ構造です.

6.2.2　PUSH 命令

Push は,「押す」と言う意味の英語です．データをスタックに「押し込む」のに使用します．

意味： レジスタの値をスタックに押し込む．

ニーモニック： `PUSH GR`

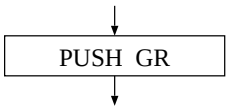
`GR` で指定されたレジスタがスタックに押し込まれます．`GR` が SP を表す場合は，SP の値がスタックに押し込まれます．

命令フォーマット： PUSH 命令は，1 バイト長の命令です．XR フィールドは必ず 00 にします．

第 1 バイト	
OP	GR XR
1101_2	*GR* 00

動作の詳細： PUSH 命令は，まず，スタックポインタ (SP) の値を 1 減らします．次に，SP の値をアドレスとみなし，主記憶の SP 番地に指定されたレジスタの値を書き込みます．

フローチャート： 次のように描くことにします．



6.2.3　POP 命令

Pop は,「飛び出る」と言う意味の英語です．データをスタックから取り出すのに使用します．

意味： スタックのデータをレジスタに取り出す．

ニーモニック： `POP GR`

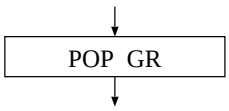
`GR` で指定されたレジスタにスタックからデータが取り出されます．`GR` が SP を表す場合は，スタックからデータが取り出され，SP に格納されます．

命令フォーマット： POP 命令は，1 バイト長の命令です．XR フィールドは必ず 10 にします．

第 1 バイト	
OP	GR XR
1101_2	*GR* 10

動作の詳細： POP 命令は，まず，主記憶の SP 番地からデータを取り出し，指定されたレジスタに格納します．次に，SP の値を 1 増やします．

フローチャート： 次のように描くことにします．



PUSH/POP 命令の応用例 (1)

PUSH 命令と POP 命令を組み合わせて，レジスタの値を一時的に保管することができます．まず，SP の値を初期化します．DCH 番地に初期化する理由は,「付録 C 命令表」のメモリマップを参照して考えてください．スタックは LIFO なので，**PUSH(保存) した順序と逆の順序で POP(復元) する必要があります．**

ラベル	ニーモニック
	`LD SP,#0DCH`
	`...`
	GO，G1 を使用する処理 1
	`...`
	`PUSH GO ; GO を保存`
	`PUSH G1 ; G1 を保存`
	`...`
	GO，G1 を使用する処理 2
	`...`
	`POP G1 ; G1 を復元`
	`POP GO ; GO を復元`
	`...`
	処理 1 の続き
	`...`

PUSH/POP 命令の応用例 (2)

PUSH 命令と POP 命令を組み合わせて，レジスタの値を別のレジスタにコピーすることができます．TeC には，レジスタ間でデータを転送する命令がないので，このテクニックは，大変，役に立ちます．次の例では，G0 の値を G1 にコピーします．

ラベル	ニーモニック
	`LD SP,#0DCH`
	`...`
	`PUSH GO ; GO を保存`
	`POP G1 ; G1 に復元`
	`...`

問題

G0 と G1 の値を交換する方法を考えなさい．

6.3 サブルーチン

プログラムの中に，同じ処理が何度も出てくることがあります．また，一つのまとまった機能を実現しているプログラムの部分があり，他の部分と独立させた方がプログラムの見通しが良くなる場合があります．例えば,「5.12 TeC シリアル入出力 (SIO)」の例題プログラムの多くで，1 文字入力や 1 文字出力を行うプログラムの部分がありました．このような，一つのまとまった機能を実現している部分がそうです．

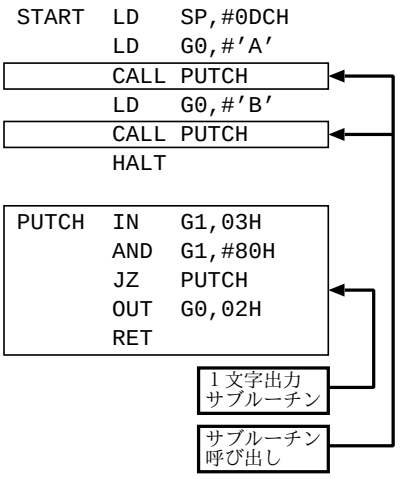
このような部分は，サブルーチン (副プログラム) としてまとめることにより，繰り返し呼び出して使用したり，他の部分から独立させて見やすいプログラムにすることができます．ここでは，このようなサブルーチンを作る方法を学びます．

6.3.1 仕組み

サブルーチンは，他のプログラムから呼び出されて実行されます．サブルーチンの実行が終わったら，呼び出したプログラムに戻り続きを実行します．

サブルーチンを呼び出すための CALL 機械語命令と，サブルーチンの最後から呼び出し元に戻るための RET 機械語命令を，新たに導入します．

次のプログラム例は，1 文字出力機能をサブルーチンとして独立させ，メインルーチン (主プログラム) から 2 回呼び出して使用する例です．プログラムは先頭から実行が開始され，CALL 命令で PUTCH サブルーチンへジャンプします．PUTCH サブルーチンを最後まで実行すると，RET 命令でメインルーチンに戻り，CALL 命令の次の命令から実行を再開します．



6.3.2 CALL 命令

Call は,「呼び出す」と言う意味の英語です．サブルーチンを呼び出すために使用します．CALL 命令は JMP 命令に似ていますが，サブルーチンから戻る準備をしてからジャンプしなければなりません．CALL 命令は，自身の次の命令のアドレスをスタックに保存 (PUSH) してから、サブルーチンにジャンプします．

意味： サブルーチンを呼び出す．

フラグ： 変化しません．

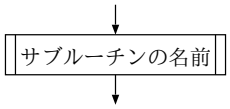
ニーモニック： `CALL A[,XR]`
A 番地のサブルーチンを呼び出します．`[,XR]` は，インデックスドモードでジャンプアドレスを計算する場合に使用します．

命令フォーマット： CALL 命令は 2 バイトの長さを持ちます．各フィールドの意味は， JMP 命令と同様です．

第 1 バイト		第 2 バイト
OP	GR XR	
1011_2	11 XR	aaaa aaaa

動作の詳細： CALL 命令は，まず，PC の値 (CALL 命令実行時には既に次の命令のアドレスになっている) を，スタックに保存 (PUSH) します．次に，サブルーチンにジャンプします．

フローチャート： 次のように描きます．箱の中には，呼び出すサブルーチンの名前を書きます．



使用上の注意： CALL 命令はスタックを使用するので，SP を初期化してから使用しなければなりません．

6.3.3　RET(Return) 命令

Return は,「戻る」と言う意味の英語です．RET は Return の綴を縮めたものです．サブルーチンから戻るために使用します.

意味： サブルーチンから戻る.

フラグ： 変化しません.

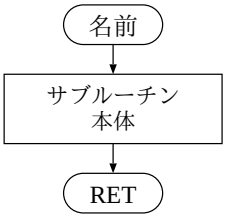
ニーモニック： RET

命令フォーマット： RET 命令は，1 バイト長の命令です.

第 1 バイト	
OP	GR XR
1110_2	11 00

動作の詳細： スタックから値を一つ取り出し PC にセットします．スタックには CALL 命令が保存した PC の値が保存されているので，プログラムの実行は，サブルーチンを呼び出した CALL 命令の次の命令に移ります.

フローチャート： 次のフローチャートの，最後の角を取った四角が RET 命令に対応します．フローチャートは，サブルーチン全体を示しています.



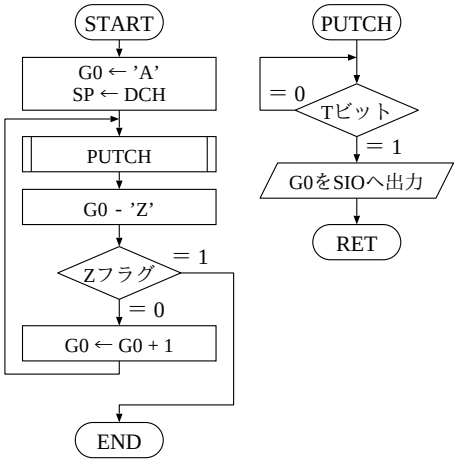
使用上の注意： RET 命令は POP 命令と同様な方法でスタックから値を取り出します．サブルーチンの中で PUSH 命令や POP 命令を使用した場合は，スタックの状態がサブルーチンの実行開始時と同じになるようにしてから，RET 命令を実行しなければなりません．サブルーチンの中で，PUSH 命令の実行回数と POP 命令の実行回数が同じになるようにして下さい.

例題 6-1 ‘A’〜‘Z’ の文字を表示 (改良版 1)

問題： SIO へ ‘A’〜‘Z’ の文字を連続して出力するプログラムを，前ページの 1 文字出力サブルーチン (PUTCH) を使用して作りなさい.

考え方： 基本的な考え方は，例題 5-9 と同じです．1 文字出力サブルーチンは，G0 レジスタにセットされた値を SIO へ出力します．メインルーチンは，G0 レジスタに ‘A’〜‘Z’ の文字の文字コードをセットして，サブルーチンを呼び出します.

フローチャート： フローチャートで，サブルーチンの呼び出しと，サブルーチンは次のように描きます.



プログラム： CALL，RET 命令がスタックを使用するので，SP の初期化を忘れないようにします．G0 レジスタを文字コードの格納に，G1 レジスタを「T ビット」のチェックに用います.

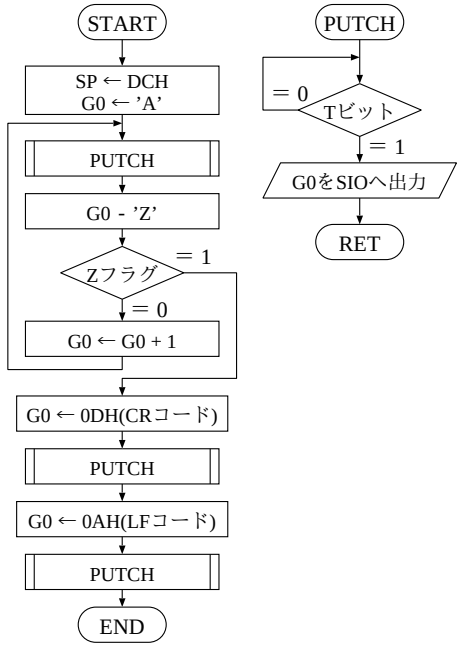
番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	1F DC		LD	SP,#0DCH
02	13 41		LD	G0,#'A'
04	BC 0F	LOOP	CALL	PUTCH
06	53 5A		CMP	G0,#'Z'
08	A4 0E		JZ	OWARI
0A	33 01		ADD	G0,#1
0C	A0 04		JMP	LOOP
0E	FF	OWARI	HALT	
0F				
0F	C4 03	PUTCH	IN	G1,SIOS
11	67 80		AND	G1,#80H
13	A4 0F		JZ	PUTCH
15	C3 02		OUT	G0,SIOD
17	EC		RET	

例題 6-2 ‘A’〜‘Z’ の文字を表示 (改良版 2)

問題： SIO へ ‘A’〜‘Z’ の文字を連続して出力した後，SIO へ改行コードを出力するプログラムを作りなさい．

考え方： 例題 6-1 では，サブルーチンを使用したメリットが明確になりませんでした．同じ，1 文字出力サブルーチン (PUTCH) を，メインルーチンの複数の箇所から呼び出すプログラムを作成します．

フローチャート： 改行するために，CR，LF を出力します．



プログラム： PUTCH ルーチンは，前の例題と同様なので省略しました．

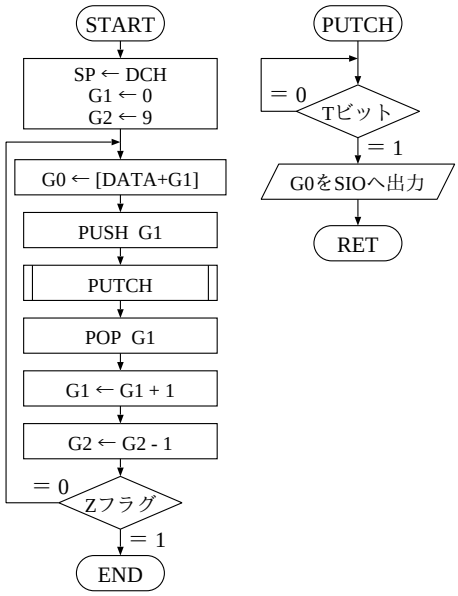
番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	1F DC		LD	SP,♯0DCH
02	13 41		LD	G0,♯'A'
04	BC 17	LOOP	CALL	PUTCH
06	53 5A		CMP	G0,♯'Z'
08	A4 0E		JZ	OWARI
0A	33 01		ADD	G0,♯1
0C	A0 04		JMP	LOOP
0E	13 0D	OWARI	LD	G0,♯13
10	BC 17		CALL	PUTCH
12	13 0A		LD	G0,♯10
14	BC 17		CALL	PUTCH
16	FF		HALT	
17				
17	C4 03	PUTCH	...	

例題 6-3 文字列出力 (改良版 1)

問題： SIO へローマ字で自分の名前を出力するプログラムを作りなさい．

考え方： 例題 5-7 を，サブルーチンを使用して作りなおします．

フローチャート： G1 レジスタが PUTCH サブルーチンで破壊される (T ビットのチェックで使用します．) ので，メインルーチン側で G1 レジスタをスタックに保存しています．



プログラム： PUTCH ルーチンとデータは省略しました．

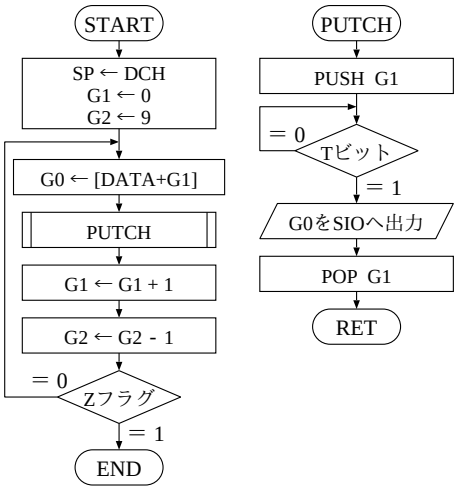
番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	1F DC		LD	SP,♯0DCH
02	17 00		LD	G1,♯0
04	1B 09		LD	G2,♯9
06	11 1E	LOOP	LD	G0,DATA,G1
08	D4		PUSH	G1
09	BC 15		CALL	PUTCH
0B	D6		POP	G1
0C	37 01		ADD	G1,♯1
0E	4B 01		SUB	G2,♯1
10	A4 14		JZ	END
12	A0 06		JMP	LOOP
14				
14	FF	END	HALT	
15				
15	C4 03	PUTCH	...	
1E				
1E	53 48	DATA	DC	"SHIGEMURA"
20	...			

例題 6-4 文字列出力 (改良版 2)

問題： SIO へローマ字で自分の名前を出力するプログラムを作りなさい.

考え方： 例題 5-7 をサブルーチンを使用して作りなおします.

フローチャート： PUTCH サブルーチンが G1 レジスタを破壊しないよう，サブルーチン側で G1 レジスタをスタックに保存しています．例題 6-3 の PUTCH サブルーチンより完成度の高い PUTCH サブルーチンです.



プログラム：

番地	機械語	ラベル	ニーモニック	
02		SIOD	EQU	02H
03		SIOS	EQU	03H
00				
00	1F DC		LD	SP,♯0DCH
02	17 00		LD	G1,♯0
04	1B 09		LD	G2,♯9
06	11 1E	LOOP	LD	G0,DATA,G1
08	BC 13		CALL	PUTCH
0A	37 01		ADD	G1,♯1
0C	4B 01		SUB	G2,♯1
0E	A4 12		JZ	END
10	A0 06		JMP	LOOP
12				
12	FF	END	HALT	
13				
13	D4	PUTCH	PUSH	G1
14	C4 03	WAIT	IN	G1,SIOS
16	67 80		AND	G1,♯80H
18	A4 14		JZ	WAIT
1A	C3 02		OUT	G0,SIOD
1C	D6		POP	G1
1D	EC		RET	
1E				
1E	53 48	DATA	DC	"SHIGEMURA"
20	49 47			
22	45 4D			
24	55 52			
26	41			

問題

1. SIO から 1 文字を入力するサブルーチン (GETCH) を作成しなさい．GETCH は，入力した文字の文字コードを G0 に格納して戻るものとします.
2. 前出の 1 文字出力サブルーチン (PUTCH) と，GETCH を利用して「例題 5-10 echo プログラム」を作りなおしなさい.
3. 「例題 5-3 割算を計算する」を参考に，割算サブルーチン DIV を作成しなさい．DIV は，G0 に割られる数，G2 に割る数をセットして呼び出され，G1 に商，G0 に余りを格納して戻るものとします.
4. 「例題 5-6 シフトを用いた高速乗算」を参考に，G0 の数値の 10 倍を計算し，G0 に求めるサブルーチン MUL10 を作成しなさい.

6.4 時間の計測

コンピュータで，時間を計る必要がある場合があります．例えば，TeC を使って 3 分間のタイマーを作ることを想像してください．

PC 等の本格的なコンピュータは，時間を計測するための専用の回路を持っています．しかし，TeC のような小規模マイコンは，専用の回路を持っていないことがあります (TeC は持っていません)．

そこで，プログラムだけで時間を計測する工夫が必要になります．

6.4.1 マシンステート

TeC で一定の時間が経過するのを待つには，各命令の実行に必要な時間を調べて，ちょうど目的の時間が必要な命令の組合せを実行させます．

命令表にステート数と書かれた欄があります．ここの数値が，命令の実行に要する時間を表します．例えば，NO 命令はステート数 3 の命令ですので，**3 マシンステート**の時間が必要になります．

TeC の場合，**1 マシンステート**は，回路の動く基準になるクロック信号の 1 周期分になります．TeC のクロック信号は，2.4576MHz ですので，**1 マシンステート**は，$1/(2.4576 \times 10^6) = 0.4069 \times 10^{-6} = 0.4069\mu s$ です．

例えば，NO 命令はステート数が 3 なので，実行に $0.4069\mu s \times 3 = 1.2207\mu s$ の時間がかかります．他の命令も，同様に実行時間が決まります．

6.4.2 1ms タイマー (マシンステートの応用)

マシンステートの合計を調節して，実行にちょうど 1ms の時間がかかるサブルーチンを作ります．

クロックの周波数から，1 秒が 2.4576×10^6 マシンステートになりますので，$1ms$ は，$(2.4576 \times 10^6) \times (1 \times 10^{-3}) = 2,457.6$ マシンステートになります．

実行に，2,458 マシンステートを要するサブルーチンを作ります．ステート数が多いので繰り返しを使用しないと，うまくいきそうにありません．繰り返しを作るには，次のようなプログラムが必要です．

ラベル	ニーモニック		ステート
MS1	PUSH	G0	6
	LD	G0,♯n	5
MATU	SUB	G0,♯1	5
	JZ	KAERU	4/5
	JMP	MATU	5
KAERU	POP	G0	6
	RET		6

このプログラムの実行に要するステート数は，繰り返し回数を n とすると，$6 + 5 + (5 + 4 + 5) \times (n - 1) + 5 + 5 + 6 + 6$ となります．

ステート数が 2,458 になるように n を計算すると，n は 174 になります (3 ステート不足)．上のプログラムで n を 174 にしたものが，実行に約 $1ms$ かかるサブルーチンになります．

6.4.3 0.2 秒タイマー (入れ子サブルーチンの利用)

もう少し長いタイマーを作って，実行に時間がかかっていることを実感しましょう．

$1ms$ タイマーサブルーチンを利用すると，簡単にもっと長い時間のタイマーサブルーチンを作ることができます．$200ms$ タイマールーチンは，$1ms$ タイマーサブルーチンを 200 回呼び出すように作ります．(**「サブルーチンの中から別のサブルーチンを呼び出すこと」 =「サブルーチンの入れ子」を利用します．**)

次のプログラムは，実行に約 1 秒かかるようにしたものです．

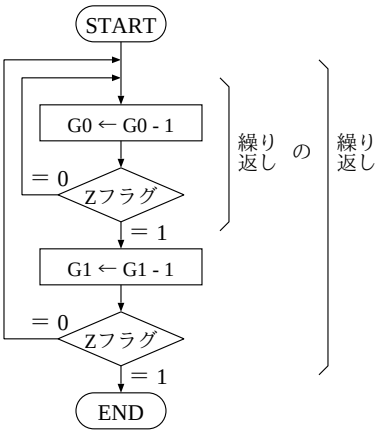
ラベル	ニーモニック	
	; メインルーチン	
START	LD	SP,♯0DCH
	CALL	MS200
	CALL	MS200
	CALL	MS200
	CALL	MS200
	CALL	MS200
	HALT	
	; 200ms サブルーチン	
MS200	PUSH	G0
	LD	G0,♯200
L1	CALL	MS1
	SUB	G0,♯1
	JZ	L2
	JMP	L1
L2	POP	G0
	RET	
	; 1ms サブルーチン	
MS1	PUSH	G0
	LD	G0,♯174
MATU	SUB	G0,♯1
	JZ	KAERU
	JMP	MATU
KAERU	POP	G0
	RET	

6.4.4　1 秒タイマー (多重ループの利用)

約 $1ms$ のタイマーを 200 回呼び出すサブルーチンを 5 回呼び出す方法では，誤差がたまって，正確な 1 秒タイマーではなくなっている可能性があります．正確な 1 秒タイマーを，1 から作りなおします．

1 秒タイマーは，2,457,600 マシンステートのプログラムになります．$1ms$ タイマーでは，繰り返し 1 回が 14 ステートでした．同じステート数だとすると，$2,447,600/14 = 174,828$ 回 ループを繰り返す必要があります．

カウンタに使用したレジスタは 8 ビットですから，256 回のカウントが限界です．そこで，下のフローチャートのような繰り返しを更に繰り返す (2 重ループ) プログラムを作成する必要があります．



ところが，256 回を 256 回繰り返しても，まだ，繰り返し回数の 174,828 回 に足りません．そこで，実際には「「繰り返し」の繰り返し」の繰り返し」(3 重ループ) を作ります．

プログラムは以下のようになります．繰り返し回数は，プログラムを書いてプログラム各部のステート数が定まった後で，試行錯誤で決めました．

番地	機械語	ラベル	ニーモニック		
00	13 6C	START	LD	G0,♯108	; 5
02	17 CC	L1	LD	G1,♯204	; 5
04	1B 07	L2	LD	G2,♯7	; 5
06	4B 01	L3	SUB	G2,♯1	; 5
08	A4 0C		JZ	L4	; 4/5
0A	A0 06		JMP	L3	; 5
0C	47 01	L4	SUB	G1,♯1	; 5
0E	A4 12		JZ	L5	; 4/5
10	A0 04		JMP	L2	; 5
12	43 01	L5	SUB	G0,♯1	; 5
14	A4 18		JZ	L6	; 4/5
16	A0 02		JMP	L1	; 5
18					
18	00	L6	NO		; 3
19	00		NO		; 3
1A	FF		HALT		; 3

問題

1. 0.2 秒タイマーサブルーチンを使用した，1 秒タイマープログラムを入力して実行しなさい．
2. 0.2 秒タイマーサブルーチンを利用して，ブザーを 1 秒間隔で 0.2 秒間鳴らすプログラムを作りなさい．このプログラムは STOP を押すまで繰り返し実行するものとします．
3. 多重ループを利用した 1 秒タイマープログラムを，入力して実行しなさい．
4. 2 重ループの練習として，SIO に ‘A’ を 10 文字，5 行出力 (下の出力例参照) するプログラムを作りなさい．
5. 入れ子サブルーチンの練習として，SIO に ‘A’ を 10 文字，5 行出力するプログラムを作りなさい．(1 行出力サブルーチンを 5 回呼び出します．1 行出力サブルーチンは，1 文字出力サブルーチンを 10 回呼び出します．)
6. 最初の 3 分間は 1 秒毎にブザーを鳴らし，3 分経過したらブザーを鳴らしっぱなしするプログラム (3 分タイマー) を作りなさい．

‘A’ を 10 文字，5 行出力した例

```
A A A A A A A A A A
A A A A A A A A A A
A A A A A A A A A A
A A A A A A A A A A
A A A A A A A A A A
```

6.5 数値の入出力

これまでに，SIO を使用した文字の入出力を勉強しました．ここでは，数値の入出力を扱います．

6.5.1 2 進数の出力

データを 2 進数として表現したものを，人間に読める形式で SIO に出力します．ビットの 0 は，文字の ‘0’ に，ビットの 1 は，文字の ‘1’ に変換します．ここでは，1 バイトのデータを表示するプログラムの例を示します．

プログラムは，DATA 番地の内容を上位ビットから順に ‘0’, ‘1’ の文字に変換し，SIO へ出力します．8 ビット分の出力が終わると，CR(0DH)，LF(0AH) を出力して画面を改行します．文字の出力には，前出の PUTCH サブルーチンを使用します．

このプログラムを実行すると，PC の画面に，DATA 番地のデータ CAH を 2 進数で表した，“11001010” が表示されます．実際に実行してみてください．

番地	機械語	ラベル	ニーモニック
02		SIOD	EQU 2
03		SIOS	EQU 3
00			
00	1F DC		LD SP,♯0DCH
02	14 29		LD G1,DATA
04	1B 08		LD G2,♯8
06	13 31	L1	LD G0,♯’1’
08	95		SHLL G1
09	A8 0D		JC L2
0B	13 30		LD G0,♯’0’
0D	BC 1E	L2	CALL PUTCH
0F	4B 01		SUB G2,♯1
11	A4 15		JZ L3
13	A0 06		JMP L1
15			
15	13 0D	L3	LD G0,♯0DH
17	BC 1E		CALL PUTCH
19	13 0A		LD G0,♯0AH
1B	BC 1E		CALL PUTCH
1D	FF		HALT
1E			
1E	D4	PUTCH	PUSH G1
1F	C4 03	PL1	IN G1,SIOS
21	67 80		AND G1,♯80H
23	A4 1F		JZ PL1
25	C3 02		OUT G0,SIOD
27	D6		POP G1
28	EC		RET
29			
29	CA	DATA	DC 0CAH

6.5.2 16 進数の出力

DATA 番地のデータを 16 進数で SIO に出力するプログラムの例を，以下に示します．

まず，データの上位 4 ビットを取り出し，その値を用いて ‘0’〜‘F’ の 1 文字を選択し SIO へ出力します．次に，下位 4 ビットを用いて同様の出力をします．

最後に，CR(0DH)，LF(0AH) を出力して画面を改行します．このプログラムを実行すると，PC の画面に，DATA 番地のデータ “CA” が表示されます．

番地	機械語	ラベル	ニーモニック
02		SIOD	EQU 2
03		SIOS	EQU 3
00			
00	1F DC		LD SP,♯0DCH
02			; 上位桁の出力
02	14 38		LD G1,DATA
04	97		SHRL G1
05	97		SHRL G1
06	97		SHRL G1
07	97		SHRL G1
08	11 28		LD G0,HSTR,G1
0A	BC 1D		CALL PUTCH
0C			; 下位桁の出力
0C	14 38		LD G1,DATA
0E	67 0F		AND G1,♯0FH
10	11 28		LD G0,HSTR,G1
12	BC 1D		CALL PUTCH
14			; 改行の出力
14	13 0D		LD G0,♯0DH
16	BC 1D		CALL PUTCH
18	13 0A		LD G0,♯0AH
1A	BC 1D		CALL PUTCH
1C	FF		HALT
1D			; 1 文字出力
1D	D4	PUTCH	PUSH G1
1E	C4 03	WAIT	IN G1,SIOS
20	67 80		AND G1,♯80H
22	A4 1E		JZ WAIT
24	C3 02		OUT G0,SIOD
26	D6		POP G1
27	EC		RET
28			; 16 進数字
28	30 31	HSTR	DC ’0’,’1’,’2’,’3’
2A	32 33		
2C	34 35		DC ’4’,’5’,’6’,’7’
2E	36 37		
30	38 39		DC ’8’,’9’,’A’,’B’
32	41 42		
34	43 44		DC ’C’,’D’,’E’,’F’
36	45 46		
38			; 出力するデータ
38	CA	DATA	DC 0CAH

6.5.3　10 進数の出力

DATA 番地のデータを 10 進数で SIO に出力するプログラムを考えましょう．扱うデータは 8 ビットの符号無し 2 進数ですので，値の範囲は 0 ～ 255(最大 3 桁) です．10 進数で表示するには，10 進数の一桁一桁に分解した後，一桁毎に対応する数字に変換して出力します．

各桁への分解

次のような手順で，数値を一桁一桁に分解します．下の，計算例を参照しながら確認して下さい．

1. データを 10 で割って商と余りを求めます．この時，余りが最下位桁 (1 位桁) の値です．
2. 商を更に 10 で割って，新しい商と余りを求めます．新しい余りが，下から 2 桁目 (10 位桁) の値です．
3. 新しい商は下から 3 桁目 (100 位桁) の値になります．

計算例

10) 123
10) 12 ･･･3　(余り 3 が 1 位桁の値)
1 ･･･2　(余り 2 が 10 位桁の値)
(商 1 が 100 位桁の値)

以上の手順をプログラムにするためには，割算機能が必要です．しかし，TeC は割算命令を持っていません．ここでは，**例題 5-3** を参考に割算サブルーチンを準備し，DIV と言う名前で作成済みと仮定します．

次は，割算サブルーチン DIV を使用した桁への分解プログラム (部分) です．各桁の値は，D1，D2，D3 の 3 バイトに格納されます．後で，これらの値を上の桁から順に出力します．

各桁に分解する部分

```
PUTDEC  LD    SP,#0DCH   ; SP 初期化
        LD    G0,DATA    ; 割られる数
        LD    G2,#10     ; 割る数
        CALL  DIV        ; 割算
        ST    G0,D1      ; 余り (1 位桁)
        PUSH  G1         ; 商を
        POP   G0         ; 割られる数に
        CALL  DIV        ; 割算
        ST    G0,D2      ; 余り (10 位桁)
        ST    G1,D3      ; 商 (100 位桁)
```

各桁の出力

桁への分解ができたら，分解した各桁を上の桁から順に数字に変換して SIO へ出力します．数字の文字コードが ‘0’ ～ ‘9’ まで連続していることを利用すると，数値から数字への変換が簡単にできます．

数字 ＝ 数値 ＋ ‘0’ の文字コード

変換したら，前出の PUTCH サブルーチンを利用して SIO へ出力します．次は，100 位桁の値を数字に変換し出力するプログラム (部分) です．

一桁目を数字に変換し出力する部分

```
        LD    G0,D3      ; 100 位桁の数値
        ADD   G0,#'0'    ; 文字コードへ変換
        CALL  PUTCH      ; SIO へ出力
```

6.5.4　10 進数の入力

10 進数の出力ができるようになりましたので，今度は入力を考えましょう．

10 進数の出力と，ちょうど逆の手順で行います．まず，一桁毎に入力し，数字を数値に変換します．次に，各桁の値を合成して一つの数値にします．

各桁の入力

入力は符号無し 8bit 2 進数で表現できる 0 ～ 255 の数値です．今回はプログラムの作りやすさを優先して，3 桁の数字しか入力されない前提で考えます．100 未満の数値を入力するときは，“012” のように上の 1 桁に ‘0’ を入力します．10 未満の数値を入力するときは，“001” のように上の 2 桁に ‘0’ を入力します．

1 文字入力し，入力した文字の文字コードから ‘0’ の文字コードを引きます．これで，数字に対応した数値が求まります．入力された各桁の値は，メモリ上の領域に保存しておきます．

数値 ＝ 数字 − ‘0’ の文字コード

ここまでの手順をプログラムにすると，次のリストのようになります．プログラム中で 1 文字入力 (GETCH) 毎に 1 文字出力 (PUTCH) しているのは，入力した文字が端末 (PC) の画面に表示されるようにするためです．

3 桁入力し数値に変換してメモリに保存する部分

ラベル	ニーモニック		コメント
GETDEC	LD	SP,#0DCH	; SP 初期化
	CALL	GETCH	; 1 文字入力
	CALL	PUTCH	; エコーバック
	SUB	G0,#'0'	; 文字コードから値に
	ST	G0,D3	; 100 位桁
	CALL	GETCH	; 1 文字入力
	CALL	PUTCH	; エコーバック
	SUB	G0,#'0'	; 文字コードから値に
	ST	G0,D2	; 10 位桁
	CALL	GETCH	; 1 文字入力
	CALL	PUTCH	; エコーバック
	SUB	G0,#'0'	; 文字コードから値に
	ST	G0,D1	; 1 位桁

各桁の値の合成

下の式のような計算で，目的の値を求めることができます．2 行目の式のように計算すれば，10 倍を計算するサブルーチンを 2 回呼び出せば良いことになります．

$$\begin{aligned} \text{値} &= 100\text{ 位桁} \times 100 + 10\text{ 位桁} \times 10 + 1\text{ 位桁} \\ &= ((100\text{ 位桁} \times 10) + 10\text{ 位桁}) \times 10 + 1\text{ 位桁} \end{aligned}$$

この計算をプログラムにすると，下のリストのようになります．ここで MUL10 は，G0 値の 10 倍を計算するサブルーチンです．**例題 5-6** を参考に，予め MUL10 を作成しておく必要があります．

各桁の値を合成する部分

ラベル	ニーモニック		コメント
	LD	G0,D3	; 100 位桁
	CALL	MUL10	; × 10
	ADD	G0,D2	; 10 位桁
	CALL	MUL10	; × 10
	ADD	G0,D1	; 1 位桁
	ST	G0,DATA	; 結果を保存

掛算の回数

一般に，掛算や割算は時間のかかる計算です．TeC のような掛算命令や割算命令を持たない小さなコンピュータにとっては，特に時間がかかります．そこで，掛算や割算の回数が少なくなるように，プログラムするよう心掛ける必要があります．上で掛算の回数が少なくるなるように式を変形したのは，そのような意味があるからです．

問題

1. SIO から入力した文字の文字コードを 16 進数で出力するプログラムを作成しなさい．(1 文字入力サブルーチン (GETCH)，16 進数出力サブルーチン (PUTHEX) を作成し，これらを用いてプログラムを完成しなさい．)
2. 本文中のプログラムを参考に，DATA 番地のデータを 10 進数で出力するプログラムを作成しなさい．
3. 10 進数の出力プログラムを，出力する数値の桁数に合わせて先行する余分な '0' が出力されないよう改良しなさい．
4. 10 進数の出力プログラムを，D1，D2，D3 領域の代わりにスタックを使用するように改造しなさい．
5. 本文中のプログラムを参考に，10 進数を入力し DATA 番地に格納するプログラムを作成しなさい．
6. 10 進数を入力するプログラムを，数字以外が入力されるまで入力を受け付けるように改造しなさい．(3 桁固定でなく，10[エンター] や 1[エンター] で入力できるプログラムにする．)
7. 10 進数を入力し，10 加えた値を 10 進数で出力するプログラムを作成しなさい．(10 進数入力サブルーチン (GETDEC)，10 進数出力サブルーチン (PUTDEC) を作成し，これらを用いてプログラムを完成しなさい．)
8. 二つの 10 進数を入力し，和を 10 進数で出力するプログラムを作成しなさい．
9. ゼロが入力されるまで 10 進数を入力し，合計を 10 進数で出力するプログラムを作成しなさい．

6.6　符号付数の入出力

「2.3 負数の表現」で学んだように，負の数を 2 の補数表現を用いて 2 進数で表現することができます．TeC も，負の数を 2 の補数表現を用いて表現します．TeC は 8bit 符号付 2 進数を扱いますので，−128 〜 +127 の数値を扱うことができます．

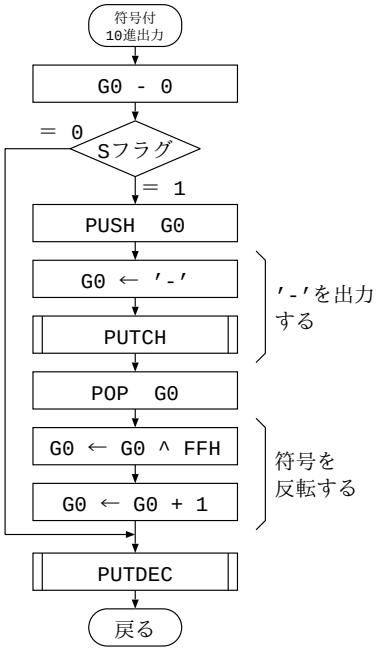
データが符号付であるか符号無しであるかは，そのデータをプログラムがどのように扱うかで決まります．特に，データの外部表現を左右する入出力ルーチンは，符号付の場合とそうでない場合で，処理内容がはっきり区別されます．

6.6.1　符号付 10 進数の出力

以下に，G0 に格納された符号付 2 進数の値を，SIO へ出力するサブルーチンのフローチャートを示します．なお，G0 に格納した 1 文字を出力する PUTCH サブルーチン，G0 に格納した値を 10 進数に変換して出力する PUTDEC サブルーチンが，予め準備されているものとします．

まず，G0 の値と 0 を比較し，G0 の値が正か負かを判断します．負の場合は，‘−’ を出力した後，G0 の値の符号を反転します．符号の反転は，「2.3.6 2 の補数から元の数を求める手順」で学習したように，**「ビット反転した後，1 を加える」**ことで実行できます．ビットの反転には XOR 命令を使用します．

最後に，PUTDEC サブルーチンを用いて G0 の値を 10 進数で出力します．



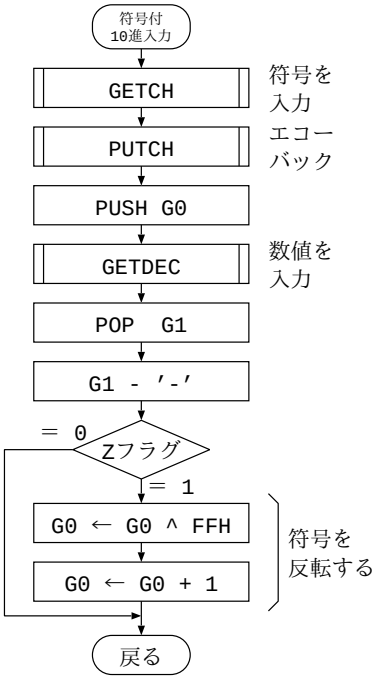
6.6.2　符号付 10 進数の入力

入力の形式に自由度があるとプログラムが難しくなるので，必ず次のような形式で入力されるものとします．

正の数：空白 数字 数字 数字
負の数：‘−’ 数字 数字 数字

このような形式で数値が入力される場合の，符号付数値の入力サブルーチンのフローチャートを示します．なお，SIO から 1 文字を入力し G0 に格納するサブルーチン GETCH，SIO から 10 進数を入力し G0 に格納するサブルーチン GETDEC が，予め準備してあるものとします．

まず，符号を入力しスタックに退避します．続いて 10 進数を G0 に入力します．退避しておいた符号を G1 に取り出し，文字 ‘−’ と比較します．同じなら，G0 の符号を反転します．



問題

1. 符号付の 10 進数出力サブルーチン (PUTSDEC) を作成しなさい．
2. 符号付の 10 進数入力サブルーチン (GETSDEC) を作成しなさい．
3. ゼロが入力されるまで符号付の 10 進数を入力し，合計を符号付 10 進数で出力するプログラムを作成しなさい．

6.7 アドレスデータ

番地をデータとして扱えると便利なことがあります．番地 (＝アドレス) のデータですので，ここではアドレスデータと呼びます．C 言語を勉強したことのある人は，ポインタのことだ言えばピンとくると思います．アドレスデータを使用すると便利な例としては，サブルーチンにデータの置き場所を教えたいような場合が考えられます．

6.7.1 TeC のアドレスデータ

TeC では，インデクスドモードのアドレッシングを用いて，レジスタで指定された，特定アドレスのデータをアクセスすることが可能です．例えば，12H 番地にあるデータを G0 レジスタにロードするプログラムは，次のようになります．

```
LD   G1,♯12H
LD   G0,0,G1
```

G1 インデクスドモードでは，データのアドレスを 「第 2 バイト+G1」 で求めます．このプログラムの場合，第 2 バイトが 0 ですから，G1 の値がそのままデータのアドレスになります．つまり，G1 に格納されたアドレスデータを用いて，そのアドレスをアクセスします．

6.7.2 文字列出力サブルーチン (アドレスデータの使用例)

文字列出力サブルーチンを作ってみましょう．サブルーチンには，出力すべき文字列を渡す必要があります．ここでは文字列を，例題 5-8 で用いたのと同じ，文字コード 0 で終わるバイト列として表現することにします．(この文字列表現方式は，C 言語と同じです．)

サブルーチンにデータを渡す方法として，これまではレジスタを使用してきました．例えば，1 文字出力サブルーチン (PUTCH) は，出力したい文字の文字コードを G0 に格納して呼び出されるようになっていました．文字列の場合はデータの量が多いので，レジスタにデータ全体を格納してサブルーチンに渡すことができません．そこで，文字列データの置いてあるアドレスを渡すことにします．

次のプログラムは，文字列出力サブルーチン (PUTSTR) が，出力する文字列のアドレスを受け取るようにした例です．G1 に文字列のアドレスをセットして，PUTSTR を呼び出します．なお，プログラム中「♯STR1」のような記述は，STR1 ラベルの値を意味します．どのような機械語へ変換されているか，確認してください．

番地	機械語	ラベル	ニーモニック
02		SIOD	EQU 02H
03		SIOS	EQU 03H
00			
00	1F DC	START	LD SP,♯0DCH
02	17 2F		LD G1,♯STR1
04	BC 13		CALL PUTSTR
06	17 41		LD G1,♯CRLF
08	BC 13		CALL PUTSTR
0A	17 39		LD G1,♯STR2
0C	BC 13		CALL PUTSTR
0E	17 41		LD G1,♯CRLF
10	BC 13		CALL PUTSTR
12	FF		HALT
13			
13			; 文字列出力ルーチン
13	D0	PUTSTR	PUSH G0
14	D4		PUSH G1
15	11 00	LOOP	LD G0,0,G1
17	53 00		CMP G0,♯0
19	A4 21		JZ RETURN
1B	BC 24		CALL PUTCH
1D	37 01		ADD G1,♯1
1F	A0 15		JMP LOOP
21	D6	RETURN	POP G1
22	D2		POP G0
23	EC		RET
24			
24			; 1 文字出力ルーチン
24	D4	PUTCH	PUSH G1
25	C4 03	WAIT	IN G1,SIOS
27	67 80		AND G1,♯80H
29	A4 25		JZ WAIT
2B	C3 02		OUT G0,SIOD
2D	D6		POP G1
2E	EC		RET
2F			
2F	53 48	STR1	DC "SHIGEMURA",0
31	49 47		
33	45 4D		
35	55 52		
37	41 00		
39			
39	54 45	STR2	DC "TETSUJI",0
3B	54 53		
3D	55 4A		
3F	49 00		
41			
41	0D 0A	CRLF	DC 0DH,0AH,0
43	00		

6.7.3　ジャンプテーブル

これまで，データのアドレスばかり考えてきましたが，プログラムのアドレスも同様に扱うことができます．インデクスドモードのアドレッシングは，ジャンプ命令や CALL 命令でも使用できます．

このことを利用すると，表に登録してあるいくつかのサブルーチンから一つを選んで実行するような，プログラムをつくることができます．サブルーチンを登録してある表のことを，ここでは，ジャンプテーブルと呼びます．

ジャンプ命令を表にする場合

ジャンプ命令や CALL 命令のインデクスドアドレッシングは，ジャンプ先アドレスをインデクスレジスタを用いて計算します．例えば次の JMP 命令は，A+G2 番地にジャンプします．

```
        JMP   A,G2
```

つぎのプログラムは，G2 の値を使用して TBL 以下のどれか一つの JMP 命令を選択し，PRG0, PRG1, ... のいずれかにジャンプする例です．つまり，Java 言語や C 言語の switch 文に似た動きをします．JMP 命令は 2 バイトの命令なので，SHLL を用いて G2 の値を 2 倍にしています．

```
        LD    G2,N
        SHLL  G2
        JMP   TBL,G2
TBL     JMP   PRG0
        JMP   PRG1
          ...
```

CALL 命令を用いてサブルーチンを呼び出すこともできます．CALL 命令の次の命令にサブルーチンから戻ってき来るので，注意して下さい．

```
        LD    G2,N
        SHLL  G2
        CALL  TBL,G2
サブルーチンから戻る場所
            ...
TBL     JMP   PRG0
        JMP   PRG1
            ...
```

アドレスを表にする場合

ジャンプ命令を表にすると表の要素が 2 バイトになります．アドレスだけを表にすると表の要素が 1 バイトで済み，メモリの節約になります．

次のプログラムは，アドレスを表にした例です．G1 の値を使用し，表から PRG0, PRG1, ... のどれかのアドレスを選び G2 にロードします．次に，JMP 命令で G2 の内容が表す番地にジャンプします．

```
        LD    G1,N
        LD    G2,TBL,G1
        JMP   0,G2
TBL     DC    PRG0
        DC    PRG1
          ...
```

次のプログラムは，CALL 命令を用いてサブルーチンを呼び出す例です．

```
        LD    G1,N
        LD    G2,TBL,G1
        CALL  0,G2
サブルーチンから戻る場所
          ...
TBL     DC    PRG0
        DC    PRG1
          ...
```

問題

1. G1 レジスタで指定された番地から始まる G0 レジスタで指定されたバイト数の領域を，00H で埋めつくすサブルーチンを作りなさい．
2. C 言語の switch 文が，TeC の機械語でどのように表現できるか考察しなさい．

実効アドレス

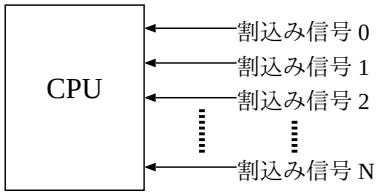
本文では，演算，転送やジャンプの対象となるメモリのアドレスのことを,「データのアドレス」,「ジャンプ先アドレス」と書いてありますが，正しい用語では「実効アドレス (Effective Address)」と呼びます．また，Effective Address を EA と略して書くこともあります．命令表で命令の動作説明の欄にさかんに `EA` とか `[EA]` の記述が出てきますが，これは実効アドレスそのものや，実効アドレスに置いてあるデータのことを指しています．

6.8 割込み (Interrupt)

ノイマン型のコンピュータはプログラムを逐次実行しますので，仕事を順番に処理するのが基本です．しかし緊急の仕事が発生したとき等は，処理中の仕事を後回しにして，緊急の仕事を先に処理しなければならない場合があります．割込みは，そのような緊急の事象をコンピュータに伝えて，優先される処理を割り込ませる仕組です．英語では割り込みのことを，Interrupt と言います．

割込みは，CPU の外部で発生した事象をハードウェアにより CPU に伝えることにより発生します．割込みにはいくつかの種類があり，CPU には割込みの種類毎に入力が用意されています．

割込みが発生すると，CPU は現在実行中のプログラムを中断し，予め登録された割込み処理プログラムの実行を開始します．割込み処理プログラムは，割込みの種類毎に登録することができます．



6.8.1 TeC の割込み種類

TeC の場合，割込みの種類は 4 種類です．四つの割込みには，“INT0”，“INT1”，“INT2”，“INT3” という名前が付いています．各割込みの意味は下の表のようになっています．

なお，“INT0” は，ハンダ付け練習基板を拡張ポートに接続した場合に，約 1 秒毎に割込みが発生するタイマー割込みとして機能します．“INT3” は，拡張ポートに接続した回路からの割込み用に予約されています．

ベクタには，割込み毎に割込み処理プログラムを登録します．例えば “INT0” の割込み処理プログラムが 12H 番地から始まる場合は，メモリの DCH 番地に，12H を書き込みます．

割込みの種類

名前	意味	ベクタ
INT0	外部割込み (タイマー)	DCH
INT1	SIO 受信割込み	DDH
INT2	SIO 送信割込み	DEH
INT3	外部割込み	DFH

6.8.2 TeC の割込み動作

TeC の CPU は，新しい命令を実行する直前に，割込みが発生していないかチェックします．割込みが発生していた場合は，次のような動作をします．

1. PC をスタックに保存 (PUSH) する．
2. 割込み許可フラグをリセットする．(割込み禁止にする)
3. 割込みの種類 (INT0〜INT3) に対応するベクタから，割込み処理プログラムのアドレスを読み込む．
4. 割込み処理プログラムのアドレスを PC にセットする．

6.8.3 EI(Enable Interrupt) 命令

Enable は,「可能にする」という意味の英語です．EI 命令は，割り込みを「可能にする」命令です．

EI 命令は，CPU が割込みを受け付ける準備ができたとき，割込みを許可するために使用します．CPU は，リセットされると割込み禁止状態になります．プログラムは，SP の初期化，割込みベクタの設定等をしたあと，EI 命令を用いて割込みを許可します．

割込みが許可状態かそうでないかは,「割込み許可フラグ」の状態によって決まります．EI 命令は「割込み許可フラグ」をセットします．

意味： 割込みを許可する．(割込み許可フラグをセットする．)

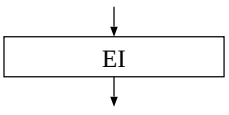
フラグ： C，S，Z フラグは変化しません．割込み許可フラグがセットされます．

ニーモニック： EI

命令フォーマット： EI 命令は，1 バイト長の命令です．

第 1 バイト	
OP	GR XR
1110_2	00 00

フローチャート： 次のように描くことにします．



6.8.4　DI(Disable Interrupt) 命令

Disable は,「できなくする」という意味の英語です．DI 命令は，割り込みを「できなくする」命令です.

CPU が割込みを受け付けると都合が悪いとき，DI 命令を使用して割込みを禁止します．EI 命令を用いて割込みを許可したあと，割込みが発生すると都合が悪くなったときに使用します.

割り込みが発生し，割り込み処理ルーチンの実行が始まった時は，自動的に割り込み禁止状態になります．それ以外で，割り込み禁止にしたいとき使用する命令です.

意味： 割込みを禁止する．(割込み許可フラグをリセットする．)

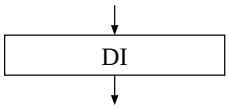
フラグ： C，S，Z フラグは変化しません．割込み許可フラグがリセットされます.

ニーモニック： `DI`

命令フォーマット： DI 命令は，1 バイト長の命令です.

第 1 バイト	
OP	GR XR
1110_2	00 11

フローチャート： 次のように描くことにします.



6.8.5　RETI(Return from Interrupt) 命令

割込み処理プログラムから通常のプログラムへ戻るための，特別な RET 命令です．RETI 命令を実行して割込み処理プログラムが終了すると，CPU は割込み許可状態になり，割込みが発生したとき実行中だったプログラムに戻ります.

意味： 割込み処理プログラムから戻る.

フラグ： C，S，Z フラグは変化しません．割込み許可フラグがセットされます.

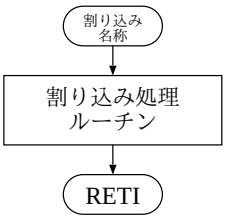
ニーモニック： `RETI`

命令フォーマット： RETI 命令は，1 バイト長の命令です.

第 1 バイト	
OP	GR XR
1110_2	11 11

動作の詳細： スタックから値を一つ取り出し PC にセットし，割込み許可フラグをセットします.

フローチャート： 次のフローチャートの，最後の角を取った四角が RETI 命令に対応します．フローチャートは，割り込み処理ルーチン全体を示しています.



6.8.6 PUSHF(Push Flag) 命令

フラグをスタックに保存する命令です．主に割込み処理プログラムで使用します．割込みは，いつ発生するか分かりません．割込み処理プログラムが終了したあと，以前のプログラムに戻って処理を続行するには，CPU の状態を完全に元通りに復元する必要があります．CPU の状態とはレジスタ (PC, SP を含む) とフラグの値のことです．

レジスタの値は，PUSH 命令でスタックに退避することができます．フラグの値は，PUSHF 命令でスタックに退避します．

意味： フラグをスタックに退避する．

フラグ： フラグは変化しません．

ニーモニック： `PUSHF`

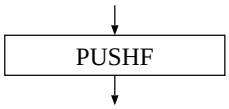
命令フォーマット： PUSHF 命令は，1 バイト長の命令です．

第 1 バイト	
OP	GR XR
1101_2	11 01

動作の詳細： スタックに退避されるデータは，次のような形式の 1 バイトになります．ここで，E が割込み許可フラグです．

E	0	0	0	0	C	S	Z

フローチャート： 次のように描くことにします．



6.8.7 POPF(Pop Flag) 命令

フラグをスタックから復元する命令です．主に割込み処理プログラムで使用します．割込み処理プログラムは，POPF 命令を使用してフラグの値を復元してから，RETI 命令で以前のプログラムに戻ります．

意味： フラグをスタックから復元します．

フラグ： フラグは，スタックから取り出した値により変化します．

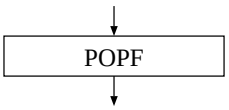
ニーモニック： `POPF`

命令フォーマット： POPF 命令は，1 バイト長の命令です．

第 1 バイト	
OP	GR XR
1101_2	11 11

動作の詳細： スタックから取り出すデータは，PUSHF 命令と同じ形式です．

フローチャート： 次のように描くことにします．



タイマー割込みに不具合がある場合

タイマー割込みを発生するハンダ付け練習基板は，コストの関係であまりしっかりした作りになっていません．1 回の割込みがかかるはずが，2 回の割込みがかかる現象が稀に発生します．その場合はハンダ付け練習基板の抵抗器 (R3) を，半分程度の抵抗値のものに変更してみてください．(オシロスコープで信号の変化時間を観測してみると，数 ms がかかっていることが分かります．数 ms はコンピュータにとって長すぎて，複数回の割込み発生の原因になっています．)

6.9　タイマー割込み

一般に，コンピュータには，割込みを発生するタイマーが内蔵されています．時間を計ったり，複数の仕事を切替えるきっかけとして，タイマーから発生する割込みを用いることが良くあるからです．

UNIX や Windows 等では，複数のプログラムが同時に実行されているように見えますが，これは 1/10 〜 1/100 秒程度の間隔で実行するプログラムを切替えることにより，複数のプログラムを同時に実行しているように見せているのです．このとき，プログラムを切替えるきっかけになるのは，タイマーからの割込みです．

6.9.1　TeC のタイマー割込み

付属のハンダ付け練習基板 (図 3.1) を組み立てて，TeC の拡張コネクタに接続すると，1 秒間隔で割込みを発生するタイマーとして動作します．複数のプログラムが同時に動いているように見せるには，間隔が長すぎて適切ではありませんが，タイマー割込みの原理を理解するには使えます．

基板を接続すると，タイマー割込みが INT0 として TeC に入力されるようになります．割込み許可命令 (EI) を実行するだけで，1 秒間隔で INT0 割込みが発生するようになります．

6.9.2　タイマー割込みの使用例

TeC のタイマー割込みを用いて，ブザーを 1 秒間隔で ON/OFF するプログラムを作成しましょう．今回は簡単のため，何もしない (無限ループ) で割込みが発生するのを待ち，割込み発生時に，割込み処理プログラムの中で，タイマーを ON/OFF するプログラムを作成します．フローチャートを下に示します．

メインプログラム

メインプログラムは，まず，スタックポインタの値を初期化します．割込みが発生すると CPU が自動的に PC をスタックに保存するので，割込みが発生する前に，**必ず，スタックポインタの値を初期化する必要があります．**

次に，INT0 の割込みベクタに割込み処理プログラムの番地を書き込みます．DCH 番地が INT0 の割込みベクタですので，この番地に割込み処理プログラムの先頭番地を書き込みます．

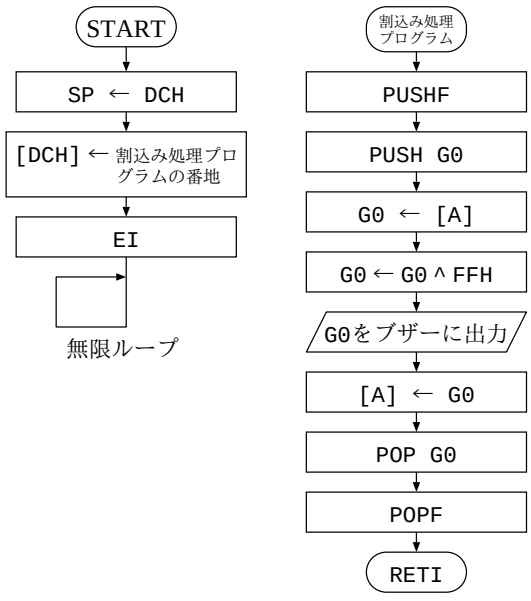
最後に，EI 命令を実行して割込み許可状態にします．これで，割込みを受け付ける準備ができましたので，無限ループを実行して割込みの発生を待ちます．TeC は，CPU が停止しているときは割込みを受け付けませんので，プログラムを終了しないで割込みの発生を待つ必要があります．

割込み処理プログラム

割込みが発生すると，CPU は PC の値をスタックに保存した後，割込みベクタに書いてある番地にジャンプします．メインプログラムが INT0 の割込みベクタに割込み処理プログラムの先頭番地を書き込んでいるので，タイマー割込み (INT0) が発生すると，割込み処理プログラムへジャンプします．

割込み処理プログラムは，メインプログラムを実行中の CPU の状態 (コンテキスト) を保存した後，ブザーを操作します．ブザーの操作が終わったら，CPU の状態を元に戻してメインプログラムに RETI 命令で戻ります．

この例では，メインプログラムが意味のある処理をしていないので，コンテキストの保存をする必要はありません．しかし，一般にはメインプログラムが意味のある処理を続けているので，割込み処理プログラムがコンテキストを破壊しないような構造でなければなりません．



コーディング例

前のフローチャートをプログラムにすると，下のようになります．プログラム中で，分かり難い部分を説明します．

“LD G0,♯TINT” は，TINT ラベルの値 (割込み処理プログラムの先頭アドレス) を，G0 レジスタにロードします．“ST G0,INTV0” は，INT0 の割込みベクタに，割込み処理プログラムの先頭アドレスを書き込みます．

割込み許可をしたら，無限ループで割込みの発生を待ちます．“L1 JMP L1” は，自分自身にジャンプする命令なので，同じ命令を繰り返し続けます．

タイマー割込みプログラムの例

```
ラベル    ニーモニック      コメント
INTOV     EQU     ODCH
          ; メインプログラム
START     LD      SP,♯ODCH
          LD      G0,♯TINT
          ST      G0,INTOV  ; ベクタ設定
          EI                ; 割込み許可
L1        JMP     L1        ; 無限ループ
          ; 割込み処理プログラム
TINT      PUSHF             ; コンテキス
          PUSH    G0        ; スト保存
          LD      G0,A      ; ブザーの
          XOR     G0,♯0FFH  ; 状態を反転
          OUT     G0,00H
          ST      G0,A
          POP     G0        ; コンテキス
          POPF              ; ト復元
          RETI              ; 戻る
          ; ブザーの状態記録領域
A         DS      1
```

問題

1. 「タイマー割込みの使用例」の動作を確認しなさい．
2. “LD G0,♯TINT” は，どんな機械語に変換されたか確認しなさい．

例題 6-5 3 分間タイマー

問題： タイマー割込みを使用した，3 分間タイマーを作りなさい．タイマーは，3 分経過したらブザーを鳴らしっぱなしにするものとします．

考え方： タイマー割込みが，約 1 秒間隔で発生することを利用して時間を計ります．割込みルーチンでカウンタの値をインクリメントし，180 回目の割込みが発生したら，ブザーを鳴らしたままプログラムを終了します．ブザーは，リセットボタンを押して止めます．

プログラム： 割込みルーチンは，180 をカウントしたらフラグ (FLAG) セットします．メインルーチンは，フラグがセットされたらブザーを ON にして，プログラムを終了します．

```
ラベル    ニーモニック      コメント
INTOV     EQU     ODCH
          ; メインプログラム
START     LD      SP,♯ODCH  ; SP 初期化
          LD      G0,♯0     ; フラグと
          ST      G0,FLAG   ; カウンタ
          ST      G0,CNT    ; リセット
          LD      G0,♯TINT
          ST      G0,INTOV  ; ベクタ設定
          EI                ; 割込み許可
L1        LD      G0,FLAG   ; フラグが 0
          CMP     G0,♯0     ; の間
          JZ      L1        ; 繰り返す
          LD      G0,♯01H
          OUT     G0,00H    ; ブザー ON
          HALT              ; 終了
          ; 割込み処理プログラム
TINT      PUSHF             ; コンテキス
          PUSH    G0        ; スト保存
          LD      G0,CNT    ; 回数のカウ
          ADD     G0,♯1     ; ント
          ST      G0,CNT
          CMP     G0,♯180   ; 3 分経過
          JZ      TINT1
          JMP     TINT2
TINT1     ST      G0,FLAG   ; フラグ ON
TINT2     POP     G0        ; コンテキ
          POPF              ; スト復元
          RETI
          ; 作業領域
FLAG      DS      1         ; フラグ
CNT       DS      1         ; カウンタ
```

6.10　入出力割込み

一般に，CPU の処理速度と入出力装置の動作速度には大きな差があるので，入出力装置を動かすとき CPU は長い待ち状態になります (囲み記事「入出力装置と CPU の速度差」参照)．入出力装置の**動作が完了**するのを待って，CPU がループを回っているだけでは時間が勿体ないので，その間に，他の仕事ができるような仕組みが望まれます．

そこで，入出力装置は**動作が完了**したことを，CPU に割込みで知らせることができるように設計してあります．キーボード，マウス，ハードディスク，ネットワーク等のほとんどの入出力装置が，動作完了の割込みを発生する機構を持っています．このような割込みを，**「入出力割込み」**と呼びます．

6.10.1　TeC の入出力割込み

TeC は標準で，SIO から入出力割込みを発生することができます．その他には，拡張ポートに入出力装置を拡張した場合，INT3 を入出力割込みとして使用できます．

SIO からの割込みは，文字を受信したとき発生する「受信割込み」と，文字を送信可能になったとき発生する「送信割込み」の 2 種類があります．これらの割込みを使用するかしないかは，割込み許可ビットにより，別々に制御できるようになっています．

SIO 受信割込み

TeC の SIO は文字を受信したとき，INT1 割込みを発生することが可能です．割込みの発生を許可する手順は次の通りです．

1. スタックポインタを初期化する．
2. INT1 割込みベクタに，SIO 受信割込み処理プログラムの番地を書き込む．
3. SIO の受信割込み許可ビットを ON にする．割込み許可ビットは，I/O 3 番地の b6 です．(「5.12.4 I/O ポート」参照)
4. EI 命令で CPU の割込みを許可する．

文字を受信すると INT1 割込みが発生します．受信した文字を，I/O マップ 3 番地の SIO 受信データレジスタ (「5.11.1 I/O マップ」参照) から読み出して下さい．データを読み出さないで割込み処理プログラムから戻ると，再度，割込み処理プログラムが呼び出されるので注意してください．

SIO 送信割込み

TeC の SIO は文字を送信することが可能なとき，INT2 割込みを発生することができます．割込みの発生を許可する手順は次の通りです．

1. スタックポインタを初期化する．
2. INT2 割込みベクタに，SIO 送信割込み処理プログラムの番地を書き込む．
3. SIO の送信割込み許可ビットをに ON する．割込み許可ビットは，I/O 3 番地の b7 です．(「5.12.4 I/O ポート」参照)
4. EI 命令で CPU の割込みを許可する．

次の文字を送信することが可能になると，INT2 割込みが発生します．送信する文字を，I/O マップ 3 番地の SIO 送信データレジスタ (「5.11.1 I/O マップ」参照) に書き込んでください．データを書き込まないで割込み処理プログラムから戻ると，再度，割込み処理プログラムが呼び出されるので注意してください．これ以上，送信する文字が無い場合は，割込み許可ビットを OFF にしてください．

6.10.2　入出力割込みの使用例

SIO を割込み駆動で使用する例を示します．

SIO 送信割込みの使用

ローマ字で自分の名前を出力するプログラムを，割込みを使用して作成します．(例題 5-7，例題 6-3，例題 6-4 と同様なプログラムです．) プログラムは下の通りです．プログラムの動きは次の通りです．

1. メインルーチンで割込みを許可すると，すぐに最初の割込みが発生する．
2. 割込みプログラムの中で，最初の文字を SIO データレジスタに書き込む．

3. その後は，1 文字の送信が終了する毎に割込みが発生するので，割込みプログラムの中で SIO データレジスタに次の文字を書き込む.

4. 全ての文字を送信したら，SIO の送信割込み許可ビットを OFF にし，それ以上の割込みが発生しないようにする.

```
ラベル      ニーモニック          コメント
            ; 注意！このプログラムは，タイマー割込み
            ; が発生することを考慮していない．実行時
            ; はハンダ付け練習基板を取り外すこと.
INT2V       EQU     0DEH
SIOD        EQU     02H
SIOC        EQU     03H
            ; メインプログラム
START       LD      SP,#0DCH        ; SP 初期化
            LD      G0,#SIOXINT     ; 割込みルー
            ST      G0,INT2V        ; チンを登録
            LD      G0,#80H         ; SIO 送信割
            OUT     G0,SIOC         ; 込み許可 ON
            LD      G0,#DATA        ; データの位
            ST      G0,POINTER      ; 置をセット
            LD      G0,#1           ; 出力中 Flag
            ST      G0,FLAG         ; セット
            EI                      ; 割込み許可
LOOP        LD      G0,FLAG         ; 出力中フラ
            CMP     G0,#0           ; グがクリア
            JZ      END             ; で終了
            JMP     LOOP            ; 割込み待ち
END         HALT
DATA        DC      "SHIGEMURA"
            DC      0
            ; SIO 送信割込み処理プログラム
SIOXINT     PUSHF
            PUSH    G0
            PUSH    G1
            LD      G1,POINTER      ; 一文字
            LD      G0,0,G1         ; 取り出す
            CMP     G0,#0           ; 文字列の
            JZ      OWARI           ; 終わりか？
            OUT     G0,SIOD         ; 送信する
            ADD     G1,#1           ; 次の文字
            ST      G1,POINTER      ; に進める
            JMP     KAERI
OWARI       LD      G0,#00H         ; SIO 送信割
            OUT     G0,SIOC         ; 込み許可 OFF
            ST      G0,FLAG         ; Flag クリア
KAERI       POP     G1
            POP     G0
            POPF
            RETI
            ; 割込み処理プログラムのデータ
POINTER     DS      1               ; 文字列位置
FLAG        DS      1               ; 動作中 FLAG
```

SIO 受信割込みの使用

次は，繰り返しブザーを鳴らすプログラムです. ブザーを鳴らす間隔を，SIO から受信した文字 (アルファベット小文字) により切替えることができます．リストでは省略してありますが,「6.4.3 0.2 秒タイマー」で作成した，MS200 サブルーチンを利用しています．実際に動かす場合は，MS200 サブルーチンも打ち込んで下さい.

```
ラベル      ニーモニック          コメント
INT1V       EQU     0DDH
BUZZ        EQU     00H
SIOD        EQU     02H
SIOC        EQU     03H
START       LD      SP,#0DCH        ; SP 初期化
            LD      G0,#SIORINT     ; 割込みルーチン
            ST      G0,INT1V        ; を登録
            LD      G0,#40H         ; SIO 受信割込み
            OUT     G0,SIOC         ; 許可ビット ON
            LD      G0,#'a'         ; 最後に入力した
            ST      G0,IKEY         ; 文字='a'
            EI                      ; 割込み許可
LOOP        LD      G0,IKEY         ;
            ST      G0,KEY
            CMP     G0,#'z'         ; 'z' が入力され
            JZ      END             ; たら終了
            LD      G0,#'a'
L1          CALL    MS200           ; 200MS 待ち
            CMP     G0,KEY
            JZ      L2
            ADD     G0,#1
            JMP     L1              ; 割込み待ち
L2          LD      G0,#1
            OUT     G0,BUZZ         ; ブザー ON
            LD      G0,#'a'
L3          CALL    MS200
            CMP     G0,KEY
            JZ      L4
            ADD     G0,#1
            JMP     L3
L4          LD      G0,#0
            OUT     G0,BUZZ         ; ブザー OFF
            JMP     LOOP
END         HALT
KEY         DS      1
IKEY        DS      1
SIORINT     PUSHF                   ; SIO 受信割込み処理
            PUSH    G0
            IN      G0,SIOD         ; 入力した文字を
            ST      G0,IKEY         ; IKEY に格納する
            POP     G0
            POPF
            RETI
```

入出力装置と CPU の速度差

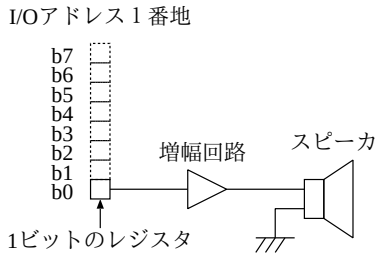
一般に，コンピュータの CPU に比較して，入出力装置はデータの処理に長い時間がかかります. 例えば，TeC の SIO (「5.12 TeC のシリアル入出力」参照) は，$9,600bau$ で動作するので，1 文字を入出力するのに約 1ms の時間が必要です．1ms は TeC の 2,458 ステートに相当します (「6.4.2 1ms タイマー」参照)．1 機械語命令が平均 5 ステートだと仮定すると，約 491 命令を実行できる時間になります．491 命令分の時間を，SIO の**動作が完了**したか調べるだけのループで費すのは勿体ないことです．その間に別の仕事をできるような工夫が望まれます.

6.11　マシンステートとスピーカ

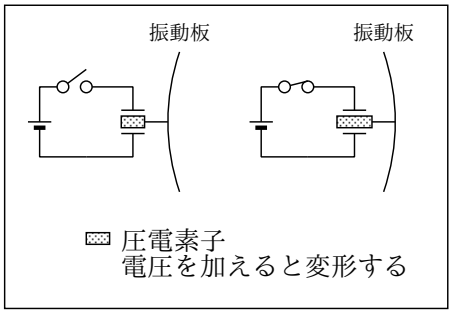
TeC には，音をだすためのスピーカが準備されています．スピーカには，OUT 命令でアクセスします．ここでは，マシンステートを意識したプログラミングを行い，スピーカから音楽を出力します．

6.11.1　スピーカーの仕組み

次の図に示すように，I/O アドレス 1 番地 (I/O マップ参照) の LSB(b0) には 1 ビットのレジスタが存在します．このレジスタの 1/0 によりスピーカに加わる電圧が変化するようになっています．なお，図では省略しましたが,「5.11.3 OUT(Output) 命令」で紹介した「ブザー音源」も，同じスピーカに接続されています．



スピーカの構造は次のようになっていて，加えた電圧により振動板が前後に移動します．



レジスタの値を，$1,0,1,0,1,0,1,0$ と変化させることにより，振動板が前後に振動して音がでます．

1kHz の音を出したい時は，プログラムで $1,0$ 書き込みを 1 秒間に 1000 回行う必要があります．

6.11.2　スピーカ出力プログラム

プログラムで 1 秒間に 1000 回のペースで $1,0$ の書き換えをするためには，時間を計って 1/1000 秒毎に 1 度，I/O ポートの書き換えが行われるようにする必要があります．

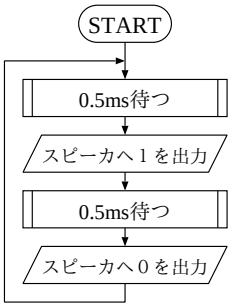
マシンステートを使用して時間を計り，一定の時間毎にスピーカポートの 0/1 を書き換えるプログラムを作ります．書き換えの間隔により，音の高さが決まります．

例題 6-6 1kHz ブザー

問題： スピーカーから 1kHz の音を出すプログラムを作りなさい．

考え方：「6.4.2 1ms タイマー」では，$1ms$ のタイマーを作りました．今回の問題では，半分の $0.5ms$ 毎にスピーカへの出力の 0/1 を書き換えるようにします．

フローチャート： 次のようになります．



プログラム： 若干のステート数の増減は誤差と考えて無視します．プログラムは，次のようになります．

番地	機械語	ラベル	ニーモニック
00			; 1kHz ブザー
01		SPK	EQU 01H
00			
00	1F DC	START	LD SP,#0DCH
02	BC 10	LOOP	CALL MS5
04	17 01		LD G1,#01H
06	C7 01		OUT G1,SPK
08	BC 10		CALL MS5
0A	17 00		LD G1,#00H
0C	C7 01		OUT G1,SPK
0E	A0 02		JMP LOOP
10			
10			; 0.5ms サブルーチン
10	D0	MS5	PUSH G0
11	13 57		LD G0,#87
13	43 01	MATU	SUB G0,#1
15	A4 19		JZ KAERU
17	A0 13		JMP MATU
19	D2	KAERU	POP G0
1A	EC		RET

問題

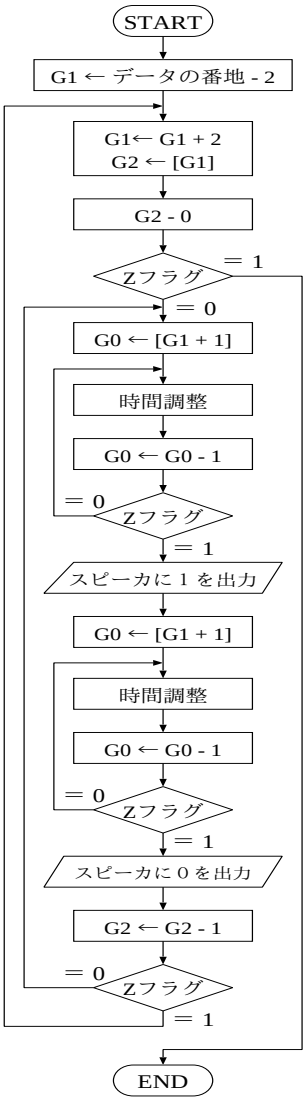
$200Hz$ ブザーを作りなさい．

6.11.3 電子オルゴールプログラム

スピーカ出力プログラムの応用として，電子オルゴールプログラムを紹介します．電子オルゴールプログラムでは，曲の楽譜をデータとして準備します．データは，音の高さを決めるためのループの繰り返し回数と，音の長さを決めるためのループの繰り返し回数を，組み合わせたものです．

フローチャート

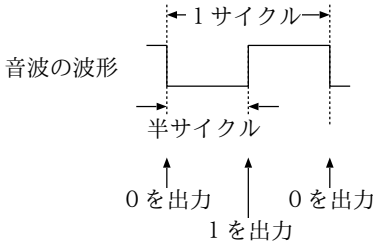
プログラムは3重のループになっています．最も内側の二つのループが，スピーカに1を出力している時間と，0を出力している時間を計っています．2番目のループの繰り返しにより，音の継続時間を制御します．最も外側のループは曲のデータを順番に取り出しています．



音符データ

最も内側のループは20ステートあります．このループの繰り返し回数で時間を計って，スピーカへの出力を半サイクル毎に書き換えます．

例えば「ラ」の音は$440Hz$ですので，半サイクルの時間は$2,457,600/440/2 = 2,792$サイクル になります．半サイクルの時間をループで計っているわけですから，ループの繰り返し回数は，$2,792/20 = 139 = 8B_{16}$回 となります．この値が，音の高さを決めます．



音の長さは，音の出力を何サイクル行うかにより決めます．例えば「ラ」の音は$440Hz$なので，$220 = DC_{16}$サイクル で0.5秒になります．このプログラムでは，0.5秒を4分音符一つの時間としました．「ラ」以外の音も，同様に対応するデータ値を次の表のように決めました．

音	周波数 (Hz)	長さ	高さ
ド	262	83	E8
ド♯	277	8B	DD
レ	294	93	D1
レ♯	311	9C	C5
ミ	330	A5	BA
ファ	349	AF	AF
ファ♯	370	B9	A6
ソ	392	C4	9C
ソ♯	415	D0	93
ラ	440	DC	8B
ラ♯	466	E9	83
シ	494	F7	7C
ド	523	FF	75

プログラム

完成したプログラムとデータを次ページに示します．プログラム中のORG命令は，プログラムのアドレスを指定する命令です．この場合，データを30H番地に生成するために使用しました．

このプログラムは，休符が表現できない，同じ高さの音が連続すると一つの音になってしまう等の問題があります．改良を考えて下さい．

電子オルゴールプログラム

番地	機械語	ラベル	ニーモニック	
01		SPK	EQU　01H	
00				
00			; 電子オルゴール	
00	17 2E		LD　G1,♯TABLE-2	
02	37 02	L9	ADD　G1,♯2	
04	19 00		LD　G2,0,G1	
06	5B 00		CMP　G2,♯0	
08	A4 2C		JZ　L16	
0A	11 01	L10	LD　G0,1,G1	; 7
0C	00	L11	NO	; 3
0D	00		NO	; 3
0E	43 01		SUB　G0,♯1	; 5
10	A4 14		JZ　L12	; 4/5
12	A0 0C		JMP　L11	; 5
14	13 01	L12	LD　G0,♯01H	; 5
16	C3 01		OUT　G0,SPK	; 8
18	11 01	L13	LD　G0,1,G1	; 7
1A	00	L14	NO	; 3
1B	00		NO	; 3
1C	43 01		SUB　G0,♯1	; 5
1E	A4 22		JZ　L15	; 4/5
20	A0 1A		JMP　L14	; 5
22	13 00	L15	LD　G0,♯00H	; 5
24	C3 01		OUT　G0,SPK	; 8
26	4B 01		SUB　G2,♯1	; 5
28	A4 02		JZ　L9	; 4/5
2A	A0 0A		JMP　L10	; 5
2C	FF	L16	HALT	
2D				
2D			; ドレミの歌	
30		TABLE	ORG　30H	
30	C5 E8		DC　0C5H,0E8H	; ド
32	49 D1		DC　049H,0D1H	; レ
34	F7 BA		DC　0F7H,0BAH	; ミ
36	41 E8		DC　041H,0E8H	; ド
38	A5 BA		DC　0A5H,0BAH	; ミ
3A	83 E8		DC　083H,0E8H	; ド
3C	A5 BA		DC　0A5H,0BAH	; ミ
3E	A5 BA		DC　0A5H,0BAH	; ミ
40	DC D1		DC　0DCH,0D1H	; レ
42	52 BA		DC　052H,0BAH	; ミ
44	57 AF		DC　057H,0AFH	; ファ
46	57 AF		DC　057H,0AFH	; ファ
48	52 BA		DC　052H,0BAH	; ミ
4A	49 D1		DC　049H,0D1H	; レ
4C	AF AF		DC　0AFH,0AFH	; ファ
4E	AF AF		DC　0AFH,0AFH	; ファ
50	AF AF		DC　0AFH,0AFH	; ファ
52	AF AF		DC　0AFH,0AFH	; ファ
54	F7 BA		DC　0F7H,0BAH	; ミ
56	57 AF		DC　057H,0AFH	; ファ
58	C4 9C		DC　0C4H,09CH	; ソ
5A	62 9C		DC　062H,09CH	; ソ
5C	52 BA		DC　052H,0BAH	; ミ
5E	C4 9C		DC　0C4H,09CH	; ソ
60	A5 BA		DC　0A5H,0BAH	; ミ
62	C4 9C		DC　0C4H,09CH	; ソ
64	C4 9C		DC　0C4H,09CH	; ソ
66	00		DC　000H	; 終了
67				

付 録A　電子オルゴールプログラムの実行例

以下は，図 A.1 の電子オルゴールプログラムを入力して実行する手順です．少し，音痴なオルゴールですが TeC から音楽が流れます．

A.1　プログラムの打ち込み

電子オルゴールプログラムは予め作ってあります．(図 A.1 の 16 進数のリストです．) これを打ち込みます．リストは，00_{16} 番地に 17_{16}，01_{16} 番地に $2E_{16}$，...，$0F_{16}$ 番地に 01_{16}，10_{16} 番地に $A4_{16}$，... を打ち込むことを表わしています．

打ち込み手順は次の通りです．操作方法は，4.2.2 を参照してください．

1. ロータリースイッチを MM の位置に合わせます．
2. アドレスランプに 00_{16}(2 進数で 0000 0000_2) をセットします．
3. 16 進数を順に打ち込みます．
   最初は，17_{16} ですから，データスイッチに 0001 0111_2 をセットして WRITE スイッチを押します．次は，$2E_{16}$ ですから，今度はデータスイッチに 0010 1110_2 をセットして WRITE スイッチを押します．以降，同様にリストの最後まで打ち込みます．
4. 最後まで打ち込んだら，最初からもう一度データを確認します．
   アドレスランプに 00_{16} をセットしデータを確認します．合っていたら，INCA スイッチを押して次の番地に進みデータを確認します．間違っていた場合は，正しい値をデータスイッチにセットして WRITE スイッチを押します．

A.2　プログラムの実行

このプログラムの実行開始番地は 00_{16} です．PC の値を 00_{16} にセットし，プログラムの実行を開始します．手順は次の通りです．

1. ロータリースイッチを PC の位置に合わせます．
2. データスイッチに 0000　0000_2 をセットし WRITE スイッチを押します．
3. BREAK スイッチ，STEP スイッチが下に倒れていることを確認します．
   (下に倒れていなかったら，下に倒します．)
4. RUN ボタンを押し，プログラムの実行を開始します．

A.3　残念ですが...

苦労して打ち込んだプログラムは保存しておきたいのですが，TeC にはその機能がありません．電源を切るとプログラムが消えてしまいます．残念ですが，毎回，打ち込む必要があります．

A.4　曲データの変更

打ち込んだ 16 進数のリストの構成は，00_{16} 番地から $2F_{16}$ 番地までにプログラム，30_{16} 番地から 65_{16} 番地までに「ドレミの歌」(途中まで) の音楽データ，66_{16} 番地に音楽データの終了を表す 00_{16} となっています (図 A.1 参照)．音楽データの部分を他の曲のデータに置き換えると別の曲を鳴らすことができます．

音楽データは 2 バイトで一つの音を表すようになっています．四分音符の各高さの音のデータは下表の通りです．1 バイト目が音の長さを調整するパラメータ，2 バイト目が音の高さを制御するパラメータです．1 バイト目の値は音の高さ毎に意味が変化します．

例えば，30_{16} 番地と 31_{16} 番地の $C5_{16}$ と $E8_{16}$ は，付点四分音符のドの音を表わしています．四分音符のドのデータは 83_{16} と $E8_{16}$ ですので，付点四分音符になるように 1 バイト目の値を 1.5 倍にして，このデータができました．

FF_{16} より長い値は指定できませんので，指定できない長い音は同じデータの繰り返しで表現して下

図 A.1: 電子オルゴールプログラムの 16 進数リスト

さい．(休符は表現できません．^^;;;) 少々(かなり?)，制限が多いですがプログラムの打ち込みの練習になりますので，是非，別の曲のデータを作って打ち込んでみて下さい．

電子オルゴールデータの意味

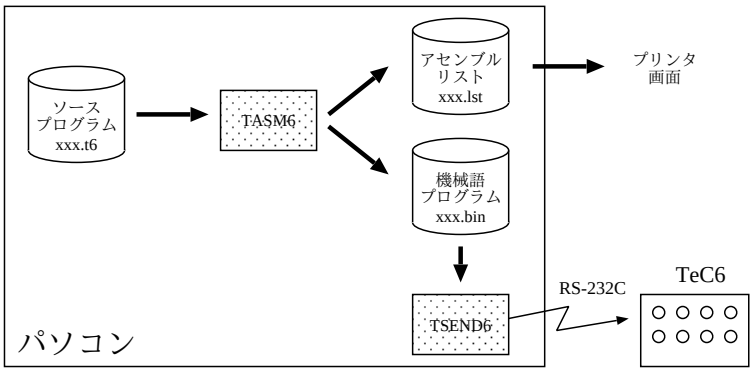
音	長さ	高さ
ド	83	E8
ド♯	8B	DD
レ	93	D1
レ♯	9C	C5
ミ	A5	BA
ファ	AF	AF
ファ♯	B9	A6
ソ	C4	9C
ソ♯	D0	93
ラ	DC	8B
ラ♯	E9	83
シ	F7	7C
ド	FF	75

付 録B　TeC6クロス開発環境

B.1　始めに

TeC6 クロス開発環境は，教育用コンピュータ TeC6(Tokuyama kousen Educational Computer Ver.6) の機械語プログラムを開発するためのパソコン上で動作するプログラム群である．クロス開発環境は，クロスアセンブラ (TASM6)，ダウンロードプログラム (TSEND6) よりなる．

TASM6 は，ニーモニックで記述されたソースプログラムを解釈し，アセンブルリストと機械語をファイルに出力する. 出力された機械語は TSEND6 プログラムにより，RS-232C 経由で TeC6 にダウンロードする．TASM6 と TSEND6 を使用することにより，TeC6 のプログラムをパソコン上でクロス開発することができる．その様子を図に示す.



処理の流れ

B.2　アセンブルコマンド

TASM6 は次の形式のコマンド行から起動され，アセンブラソースプログラム (拡張子 .t6 のファイルに格納) をアセンブルリスト (拡張子 .lst のファイルに出力) と機械語 (拡張子 .bin のファイルに出力) に変換 (アセンブル) する．

`tasm6 [-l nn] [-w nn]` *source_file*

source_file は B.4 章で説明する文法で記述されたアセンブラソースファイルの名前である．ソースファイル名の拡張子は “.t6” である必要がある．オプションの意味は次の通りである．

オプション	意味
`-l nn`	アセンブルリストの 1 ページを nn 行にする．
`-w nn`	アセンブルリストの 1 ページを nn 桁にする．

B.3 転送コマンド

TSEND6 は TASM6 が出力した機械語プログラムを RS-232C 経由で TeC6 に転送するコマンドである．次の形式のコマンド行から起動する．

`tsend6` *bin_file* *[com_port]*

bin_file は TASM6 の出力した機械語プログラムファイルの名前である．*[com_port]* は，TeC6 と接続されたシリアルポートを表すデバイスファイルの名前である．TSEND6 を起動すると画面に「TeC6 を受信状態にする」ように指示が表示されるので，TeC6 の E0H 番地に格納されている IPL プログラム (B.8 IPL プログラム 参照) を起動する．

B.4 アセンブラの文法

以下では TASM6 の入力となるソースプログラムの文法について説明する．

B.4.1 行フォーマット

ソースファイルは，行の連なりからなるテキストファイルである．なお，TASM6 のソースファイルでは文字定数・文字列中を除き英字大文字と小文字は区別されない．空白はトークンの前後にいくつでも挿入することができる．空白は，スペース (文字コード 20H) とタブ (文字コード 09H) の 2 種類である．但し，行頭に空白を置いた場合は「ラベルが存在しない」ことを表す．

行＝ [ラベル][命令とオペランド][; コメント]

B.4.2 ラベル

行の最初の文字が英字の場合はその行にラベルがあるものとみなされる．ラベルは英字で始まりその後に任意の長さの英数字が続く文字列である．ラベルが無い行の場合は行を空白文字で始める．ラベルの長さに制限はないが，アセンブルリストの見栄えから 7 文字以内を推奨する．EQU 命令のラベルを除き，その行のアドレスがラベルの値になる．

B.4.3 疑似命令

以下では，TASM6 が処理できる 4 種類の疑似命令について説明する．

EQU 命令

EQU 命令はラベルを定義するために使用する疑似命令である．命令行のラベルが式値で定義される．式では，ラベルの前方参照はできない．EQU 命令の書式は次の通りである．

EQU 命令 = *EQU*　式

EQU 命令の使用例

ラベル	命令	オペランド
START	EQU	03H
SIZE	EQU	100
END	EQU	START+SIZE

ORG 命令

ORG 命令は機械語を生成するアドレスを変更する疑似命令である．機械語 (データも含む) が生成される全ての行より前で使用された場合は，単に次の命令のアドレスを変更する．機械語が生成される行の途中で使用された場合は，指定アドレスに達するまで，ゼロで初期化された領域を生成する．ORG 命令の式では，ラベルの前方参照はできない．ORG 命令の書式は次の通りである．

$$ORG\text{ 命令} = ORG \quad \text{式}$$

ORG 命令の使用例

ラベル	命令	オペランド
START	ORG	03H

DS 命令

DS 命令は領域を定義するために使用する疑似命令である．領域の値はゼロで初期化される．DS 命令の式では，ラベルの前方参照はできない．DS 命令の書式は次の通りである．

$$DS\text{ 命令} = DS \quad \text{式}$$

DS 命令の使用例

ラベル	命令	オペランド
WORK	DS	10

DC 命令

DC 命令はメモリ上に任意の定数を出力するために使用する疑似命令である．DC 命令の書式は次の通りである．

$$DC\text{ 命令} = DC \quad \left\{ \begin{array}{c} \text{式} \\ \text{文字列} \end{array} \right\} [, \left\{ \begin{array}{c} \text{式} \\ \text{文字列} \end{array} \right\}, \cdots, \left\{ \begin{array}{c} \text{式} \\ \text{文字列} \end{array} \right\}]$$

DC 命令の使用例

ラベル	命令	オペランド
DATA	DC	1,2,3,4
	DC	1+1
CHA	DC	'a'
STR	DC	"abc"

B.4.4　機械語命令

以下では，TASM6 が処理できる 6 グループの機械語命令について説明する．

機械語命令 1

オペランドの無い 1 バイトの機械語命令である．下表の命令がこのグループに属する．

機械語命令 1 = 命令

機械語命令 *1* に属す命令	
命令	意味
NO	何もしない
PUSHF	フラグをスタックに退避
POPF	フラグをスタックから復旧
EI	割り込み許可
DI	割り込み禁止
RET	サブルーチンから復帰
RETI	割り込みから復帰
HALT	プログラム終了

機械語命令 *1* の使用例		
ラベル	命令	オペランド
END	HALT	

機械語命令 2

レジスタ指定付きの 1 バイトの機械語命令である．レジスタとして．G0，G1，G2，SP が使用可能である．下表の命令がこのグループに属する．

機械語命令 2 = 命令　レジスタ

機械語命令 *2* に属す命令	
命令	意味
SHLA	左算術シフト
SHLL	左論理シフト
SHRA	右算術シフト
SHRL	右論理シフト
PUSH	スタックにレジスタを退避
POP	スタックからレジスタを復旧

機械語命令 *2* の使用例		
ラベル	命令	オペランド
L1	SHRA	SP
	PUSH	G0
	POP	G1

機械語命令 3

オペランドにレジスタと式を書く 2 バイトの機械語命令である．レジスタとして．G0，G1，G2，SP が使用可能である．下表の命令がこのグループに属する．

機械語命令 3 = 命令　レジスタ，式

機械語命令 *3* に属す命令	
命令	意味
IN	入出力ポートから読み込む
OUT	入出力ポートに書き込む

機械語命令 *3* の使用例		
ラベル	命令	オペランド
SIO	EQU	03H
	IN	G0,SIO

機械語命令 4

レジスタとメモリをオペランドに指定できる命令で，メモリ指定にイミディエイトモードが使用できるグループである．レジスタとして．G0，G1，G2，SP が使用可能である．インデクスレジスタとして．G1，G2 が使用可能である．下表の命令がこのグループに属する．

$$\text{機械語命令 } 4 = \text{命令} \quad \left\{ \begin{array}{c} \text{レジスタ，式　[，インデクスレジスタ]} \\ \text{レジスタ，\#式} \end{array} \right\}$$

機械語命令 *4* に属す命令	
命令	意味
LD	レジスタにメモリオペランド値をロード
ADD	レジスタとメモリオペランド値の和
SUB	レジスタとメモリオペランド値の差
CMP	レジスタとメモリオペランド値を比較
AND	レジスタとメモリオペランド値の論理積
OR	レジスタとメモリオペランド値の論理和
XOR	レジスタとメモリオペランド値の排他的論理和

機械語命令 *4* の使用例		
ラベル	命令	オペランド
L1	LD	G0,DATA
	LD	G1,#0
	SUB	G0,DATA,G1
	ADD	G1,#1

機械語命令 5

機械語命令 4 と，ほぼ，同様であるが，イミディエイトモードが使用できない機械語命令のグループである．レジスタとして．G0，G1，G2，SP が使用可能である．インデクスレジスタとして．G1，G2 が使用可能である．下表のように ST 命令だけがこのグループに属する．

機械語命令 5 = 命令　レジスタ，式　[，インデクスレジスタ]

機械語命令 *5* に属す命令	
命令	意味
ST	レジスタの値をメモリオペランドに格納

機械語命令 *5* の使用例		
ラベル	命令	オペランド
	ST	G0,WORK
	ST	G0,WORK,G1

機械語命令 6

メモリオペランドだけの機械語命令グループである．下表のようにジャンプ・コール命令がこのグループに属する．インデクスレジスタとして．G1，G2 が使用可能である．

機械語命令 6 = 命令　式　[，インデクスレジスタ]

機械語命令 *6* に属す命令	
命令	意味
JMP	無条件ジャンプ
JZ	Z フラグによりジャンプ
JC	C フラグによりジャンプ
JM	S フラグによりジャンプ
CALL	サブルーチン呼び出し

機械語命令 *6* の使用例		
ラベル	命令	オペランド
	JMP	LOOP
	CALL	SUB1

B.5 アセンブラの文法まとめ

以下に文法を BNF 風にまとめる.

```
<プログラム>   ::=  { <行> } EOF
<行>           ::=  <命令行> | <空行> | <コメント行>
<命令行>       ::=  <ラベル欄> [<命令欄>] [<コメント>] <改行>
<空行>         ::=  <改行>
<コメント行>   ::=  <コメント> <改行>
<ラベル欄>     ::=  <ラベル> | <空白>
<命令欄>       ::=  <命令記述１> | <命令記述２> | <命令記述３>
                  | <命令記述４> | <命令記述５> | <命令記述６>
                  | <命令記述７> | <命令記述８>
<命令記述１>   ::=  <命令１>
<命令記述２>   ::=  <命令２> <レジスタ>
<命令記述３>   ::=  <命令３> <レジスタ> , <式>
<命令記述４>   ::=  <命令４> <レジスタ> , <アドレス１>
<命令記述５>   ::=  <命令５> <レジスタ> , <アドレス２>
<命令記述６>   ::=  <命令６> <アドレス２>
<命令記述７>   ::=  <命令７> <式>
<命令記述８>   ::=  <命令８> <値列>
<アドレス１>   ::=  # <式> | <式> [, <インデクス>]
<アドレス２>   ::=  <式> [, <インデクス>]
<値列>         ::=  <拡張式> { , <拡張式> }
<拡張式>       ::=  <式> | <文字列>
<式>           ::=  <項> ＋ <式> | <項> － <式>
<項>           ::=  <因子１> ＊ <項> | <因子１> ／ <項>
<因子１>       ::=  <因子> | ＋<因子> | －<因子>
<因子>         ::=  <数値> | <ラベル> | ( <式> )
<コメント>     ::=  ; { <文字１> }
<数値>         ::=  <１０進数> | <１６進数> | <文字定数>
<１０進数>     ::=  <数字> { <数字> }
<１６進数>     ::=  <数字> { <１６進数字> } H
<文字定数>     ::=  '<文字１>'
<文字列>       ::=  "{ <文字２> }"
<ラベル>       ::=  <英字> { <英数字> }
<英数字>       ::=  <英字> | <数字>
<文字１>       ::=  <改行>以外の文字
<文字２>       ::=  <改行>, '"' 以外の文字
<１６進数字>   ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
                  | A | B | C | D | E | F
<英字>         ::=  A | B | C | D | E | F | G | H | I | J
                  | K | L | M | N | O | P | Q | R | S | T
                  | U | V | W | X | Y | Z | _
<数字>         ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<命令１>       ::=  NO | PUSHF | POPF | EI | DI | RET |RETI | HALT
<命令２>       ::=  SHLA | SHLL | SHRA | SHRL | PUSH | POP
<命令３>       ::=  IN | OUT
<命令４>       ::=  LD | ADD | SUB | CMP | AND | OR |XOR
<命令５>       ::=  ST
<命令６>       ::=  JMP | JZ | JC | JM | CALL
<命令７>       ::=  EQU | DS | ORG
<命令８>       ::=  DC
<インデクス>   ::=  G1 | G2
<レジスタ>     ::=  G0 | G1 | G2 | SP
<改行>         ::=  '￥ｎ'
<空白>         ::=  ' ' |'￥ｔ'
```

B.6 ローマ字で名前を SIO に出力するプログラムの例

```
ADR  CODE          Label   Instruction             Comment              Page(1)

02                 SIOD    EQU     02H
03                 SIOS    EQU     03H
00
00  17 00          START   LD      G1,#0
02  19 15          L0      LD      G2,DATA,G1
04  5B 00                  CMP     G2,#0
06  A4 14                  JZ      END
08  C0 03          L1      IN      G0,SIOS
0A  63 80                  AND     G0,#80H
0C  A4 08                  JZ      L1
0E  CB 02                  OUT     G2,SIOD
10  37 01                  ADD     G1,#1
12  A0 02                  JMP     L0
14
14  FF             END     HALT
15
15  73 69 67 65    DATA    DC      "sigemura"
19  6D 75 72 61
1D  0D 0A                  DC      0DH,0AH
1F  00                     DC      0               ; 終わりの印
```

B.7 アセンブル実行例

ローマ字で名前を SIO に出力するプログラムをアセンブル・実行した例を示す.

```
$ tasm6 sigemura.t6                              <--- アセンブル
アセンブル成功
結果は [sigemura.lst] と [sigemura.bin] に格納しました。

... 省略...

$ tsend6 sigemura.bin                            <--- ダウンロード
TeC6 を受信状態にして Enter キーを押して下さい。
[00][20][17][00][19][15][5b][00][a4][14][c0][03][63][80][a4][08][cb][02][37]
[01][a0][02][ff][73][69][67][65][6d][75][72][61][0d][0a][00]
$ tip com1                                       <--- 通信ソフト起動
can't open log file /var/log/aculog.
connected
sigemura                                         <--- 実行結果
~                                                <--- 通信ソフトの終了
[EOT]
```

B.8 IPL プログラム

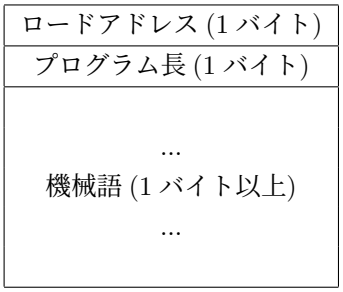
TeC6 の ROM に格納されている IPL プログラムのアセンブルリストを，参考のため以下に示す．TeC6 のメモリ空間は E0H〜FFH 番地が ROM になっており，以下の IPL プログラムが予め格納されている．この IPL プログラムは，パソコン上で TASM6 を使用してクロス開発された機械語プログラムを RS-232C から受信してメモリに格納する．

```
ADR  CODE          Label   Instruction             Comment                Page(1)

02                 SIOD    EQU     02H
03                 SIOS    EQU     03H
E0                         ORG     0E0H
E0   1F DC         IPL     LD      SP,#0DCH        ; 割込みベクタの直前がスタック
E2   BC F6                 CALL    READ            ; ロードアドレスを入力
E4   D0                    PUSH    G0
E5   D6                    POP     G1
E6   BC F6                 CALL    READ            ; 長さを入力
E8   D0                    PUSH    G0
E9   DA                    POP     G2
EA   A4 FF         LOOP    JZ      STOP
EC   BC F6                 CALL    READ
EE   21 00                 ST      G0,0,G1
F0   37 01                 ADD     G1,#1
F2   4B 01                 SUB     G2,#1
F4   A0 EA                 JMP     LOOP
F6                 ;
F6   C0 03         READ    IN      G0,SIOS
F8   63 40                 AND     G0,#40H
FA   A4 F6                 JZ      READ
FC   C0 02                 IN      G0,SIOD
FE   EC                    RET
FF                 ;
FF   FF            STOP    HALT                    ; PC がゼロになって止まる
```

B.9 機械語プログラムファイル形式

TASM6 が出力する機械語プログラムの形式は次の通りである．



付 録C　命令表

以下は，TeC の機械語命令表です．ケース蓋の裏に張り付けてあるものと同じ内容です．

TEC6命令表　Ver. 2.3 (2006.3.25)

※ステート数：イミディエイト／ダイレクト／インデクスド
(ジャンプ命令では、条件不成立／ダイレクト／インデクスド)

ニーモニック	命令	第１バイト OP	第１バイト GRXR	第２バイト	フラグ変化	ステート数※	動作
NO	No Opration	0000	00 00	————	×	3	何もしない
LD	Load	0001	GR XR	aaaa aaaa	×	5/7/7	GR <- [EA]
ST	Store	0010	GR XR	aaaa aaaa	×	-/7/7	[EA] <- GR
ADD	Add	0011	GR XR	aaaa aaaa	○	5/7/7	GR <- GR + [EA]
SUB	Subtract	0100	GR XR	aaaa aaaa	○	5/7/7	GR <- GR - [EA]
CMP	Compare	0101	GR XR	aaaa aaaa	○	5/7/7	GR - [EA]
AND	Logical And	0110	GR XR	aaaa aaaa	○	5/7/7	GR <- GR & [EA]
OR	Logical Or	0111	GR XR	aaaa aaaa	○	5/7/7	GR <- GR | [EA]
XOR	Logical Xor	1000	GR XR	aaaa aaaa	○	5/7/7	GR <- GR ^ [EA]
SHLA	Shift Left Arithmetic	1001	GR 00	————	○	4	GR <- GR << 1
SHLL	Shift Left Logical	1001	GR 01	————	○	4	GR <- GR << 1
SHRA	Shift Right Arithmetic	1001	GR 10	————	○	4	GR <- GR >> 1
SHRL	Shift Right Logical	1001	GR 11	————	○	4	GR <- GR >>> 1
JMP	Jump	1010	00 XR	aaaa aaaa	×	-/5/6	PC <- EA
JZ	Jump on Zero	1010	01 XR	aaaa aaaa	×	4/5/6	if Zero PC <- EA
JC	Jump on Carry	1010	10 XR	aaaa aaaa	×	4/5/6	if Carry PC <- EA
JM	Jump on Minus	1010	11 XR	aaaa aaaa	×	4/5/6	if Sign PC <- EA
CALL	Call subroutine	1011	11 XR	aaaa aaaa	×	-/6/7	[--SP]<-PC, PC<-EA
IN	Input	1100	GR 00	0000 pppp	×	8	GR <- IO[P]
OUT	Output	1100	GR 11	0000 pppp	×	8	IO[P] <- GR
PUSH	Push Register	1101	GR 00	————	×	6	[--SP] <- GR
PUSHF	Push Flag	1101	11 01	————	×	6	[--SP] <- FLAG
POP	Pop Register	1101	GR 10	————	×	6	GR <- [SP++]
POPF	Pop Flag	1101	11 11	————	○	6	FLAG <- [SP++]
EI	Enable Interrupt	1110	00 00	————	×	4	割り込み許可
DI	Disable Interrupt	1110	00 11	————	×	4	割り込み禁止
RET	Return from subroutine	1110	11 00	————	×	6	PC <- [SP++]
RETI	Return from Interrupt	1110	11 11	————	×	6	PC <- [SP++], EI
HALT	Halt	1111	11 11	————	×	4	停止

GR	意味
00	G0
01	G1
10	G2
11	SP

XR	意味
00	ダイレクトモード
01	G1インデクスドモード
10	G2インデクスドモード
11	イミディエイトモード

メモリマップ

Addr	内容
00 | DB	RAM
DC	INT0 割り込みベクタ
DD	INT1 割り込みベクタ
DE	INT2 割り込みベクタ
DF	INT3 割り込みベクタ
E0 | FF	ROM(IPL)

INT1 : SIO 受信割り込み
INT2 : SIO 送信割り込み

I/Oマップ

Addr	Read/Write
0	Data-Sw/b0:Beep
1	Data-Sw/b0:Speaker
2	SIO-Data/SIO-Data
3	b7:Tx Ready /b7:Tx STI b6:Rx Ready/ b6:Rx STI
4	空き/空き
....	....
F	空き/空き

図 C.1: 命令表

TeC 教科書

発行年月　２００９年６月　Ver.2.1.1
発　　行　独立行政法人国立高等専門学校機構
徳山工業高等専門学校
情報電子工学科　重村哲至
〒 745-8585 山口県周南市学園台
sigemura@tokuyama.ac.jp