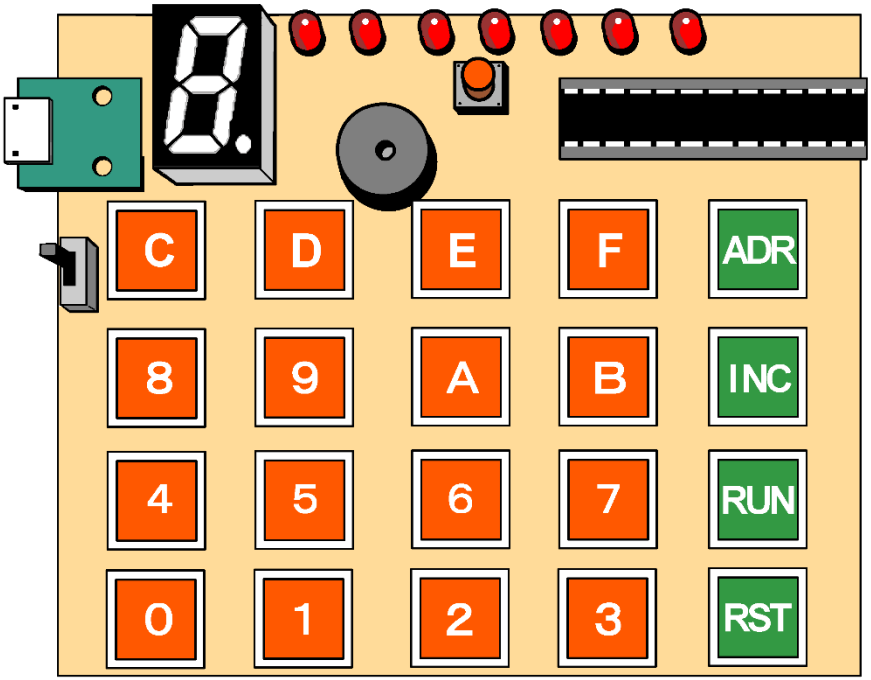
必ずわかる
ORANGE-4 機械語
プログラミング



はじめに

プログラミングの学習を始めるときに、最初はScratch (スクラッチ)がいいかもしれません。あるいはBASIC という言語を使うのがよいかもしれません。そして、コンピューターのことが何となくわかりかけてきたら、機械語を学びましょう。いいえ、最初に機械語を勉強しておいた方がいいかもしれません。
現在、機械語（アセンブリ言語）でプログラミングすることは、職業プログラマーでもほとんどありません。しかしながら、機械語（アセンブリ言語）を知っていると知らないとでは、プログラミングスキルに大きな差が出ます。

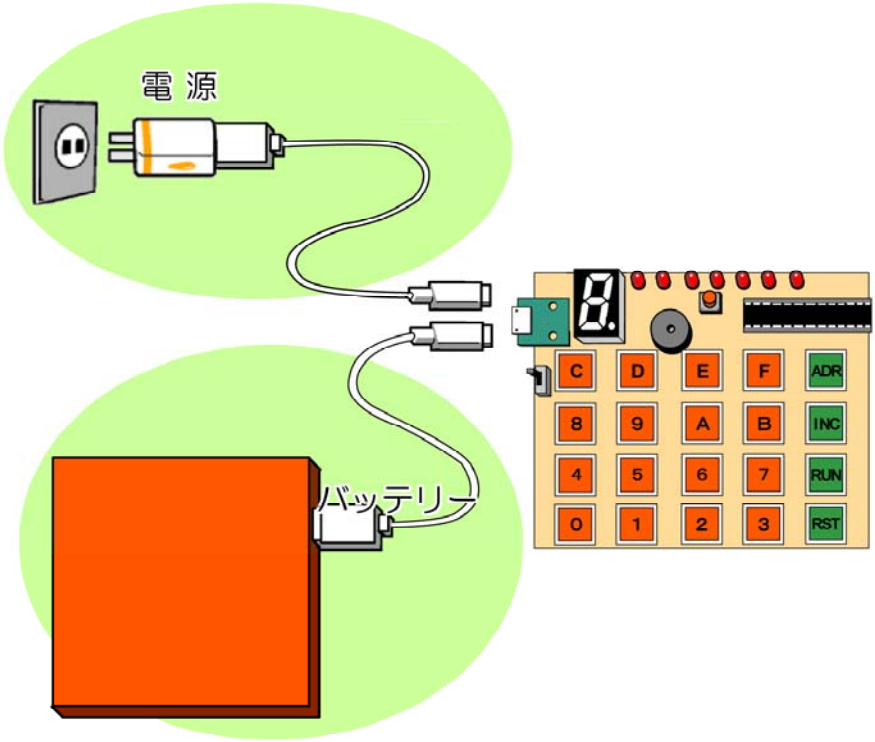
本書はORANGE-4 というトレーニングキットを利用して、2 進数の仕組みから説明し、コンピューターの構成、原理、機械語（アセンブリ言語）をやさしく解説します。
ORNAGE-4 は 4 ビットマイコンというシンプルな素材を使います。4 ビットマイコンの機械語（アセンブリ言語）を理解すれば、他のコンピューターの機械語（アセンブリ言語）を理解するときも手助けとなるでしょう。

目次

1. ORANGE-4の動作確認

●電源を入れる●

最初にORANGE-4の動作確認をしておきましょう。micro USBケーブルでモバイルバッテリーまたはUSBアダプターと接続します。必要な電源は5Vです。スマートフォンで使用しているものが使用できます。
スライドスイッチのツマミを下へスライドすると、数字LEDが点灯します。(Fが表示されます。)



●サンプルプログラムを動かす●

ORANGE-4 には、あらかじめサンプルプログラムが書き込んであります。電源を入れたら、

と順に押してみてください。

「証城寺の狸囃子」の自動演奏が始まります。繰り返し演奏されますので、演奏を中止するときは、

を押してください。

次に、

と押してみてください。

2 進 LED（7 個の LED）が順に一つずつ点灯、消灯を繰り返します。こちらも繰り返し実行しますので、中止するときは、

を押してください。

1．数の表現

●10進数●

機械語を学ぶ前に、2進数、10進数、16進数ついて知っておく必要があります。10進数の復習をしましょう。10進数は0から9までの10種類の数字を使って数を表現します。
小さい方から順に
0
1
2
3
4
5
6
7
8
9
となります。

これより大きい数を表現しようとするときは、桁を増やして表現します。
10
11
12
～
99

さらに大きい数を表現しようとするときは、また桁を増やします。
100
101
102
～
999
同様にいくらでも大きな数を表現できます。

●2 進数●

2 進数は 0 と 1 の 2 種類の数字だけを使って数を表現します。
小さい方から順に
0
1
となりますが、これより大きい数を表現しようとすると、もう桁を増やさなければいけません。
10
11
そして、またここで桁を増やします。
100
101
110
111
同様に桁数を増やせば、2 進数でも大きな数を表現できます。
2 進数 1 桁の情報を 1 ビット(1bit)と言います。
3 ビットの 2 進数で表現できる数は以下のようになります。

10 進数	2 進数
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

●ORANGE-4 での 2 進数●

ORANGE-4 の電源を入れましょう。
電源投入時は 7 つの LED がすべて消灯しています。(ORANAGE-4 は 7 ビットの 2 進 LED を装備しています。)
ここで、

を押すと、1 番右の LED が点灯します。
○○○○○○●
さらに

を押し続けると、LED は以下のように変化して行きます。
○○○○○●○
○○○○○●●
○○○○●○○
○○○○●○●
○○○○●●○
○○○○●●●

この LED の状態は、以下のように 2 進数を表現しています。

10 進数	2 進数	LED
0	0	○○○○○○○
1	1	○○○○○○●
2	10	○○○○○●○
3	11	○○○○○●●
4	100	○○○○●○○
5	101	○○○○●○●
6	110	○○○○●●○
7	111	○○○○●●●

さらに

を押し続けると、LED は

●●●●●●●

となります。

1111111 という 2 進数を表現しています。これは 10 進数ではいくつになるでしょうか？

先ほどの表のように順に書き出して行けば答えはわかります。

●16 進数●

16 進数は 0 から 9 までの 10 種類の数字の他に A、B、C、D、E、F の 6 種類の文字を使い、合計 16 種類の文字を使って数を表現します。

0
1
2
3
4
5
6
7
8
9

(使える文字がまだありますので、桁上がりの必要はありません。)

A
B
C
D
E
F

(ここで、使える文字がなくなったので桁を増やします。)

10
11
12
13
14
15
16
17

18
19
1A
1B
1C
1D
1E
1F
20
21
22
～

以上、10 進数、16 進数、2 進数、LED の対応は以下のようになります。

10 進数	16 進数	2 進数	LED
0	00	0000000	○○○○○○○
1	01	0000001	○○○○○○●
2	02	0000010	○○○○○●○
3	03	0000011	○○○○○●●
4	04	0000100	○○○○●○○
5	05	0000101	○○○○●○●
6	06	0000110	○○○○●●○
7	07	0000111	○○○○●●●
8	08	0001000	○○○●○○○
9	09	0001001	○○○●○○●
10	0A	0001010	○○○●○●○
11	0B	0001011	○○○●○●●
12	0C	0001100	○○○●●○○
13	0D	0001101	○○○●●○●
14	0E	0001110	○○○●●●○
15	0F	0001111	○○○●●●●
16	10	0010000	○○●○○○○
17	11	0010001	○○●○○○●
18	12	0010010	○○●○○●○
19	13	0010011	○○●○○●●

20	14	0010100	○○●○●○○
21	15	0010101	○○●○●○●
22	16	0010110	○○●○●●○
23	17	0010111	○○●○●●●
24	18	0011000	○○●●○○○
25	19	0011001	○○●●○○●
26	1A	0011010	○○●●○●○
27	1B	0011011	○○●●○●●
28	1C	0011100	○○●●●○○
29	1D	0011101	○○●●●○●
30	1E	0011110	○○●●●●○
31	1F	0011111	○○●●●●●
32	20	0100000	○●○○○○○
33	21	0100001	○●○○○○●
34	22	0100010	○●○○○●○
35	23	0100011	○●○○○●●
36	24	0100100	○●○○●○○
37	25	0100101	○●○○●○●
38	26	0100110	○●○○●●○
39	27	0100111	○●○○●●●
40	28	0101000	○●○●○○○
41	29	0101001	○●○●○○●
42	2A	0101010	○●○●○●○
43	2B	0101011	○●○●○●●
44	2C	0101100	○●○●●○○
45	2D	0101101	○●○●●○●
46	2E	0101110	○●○●●●○
47	2F	0101111	○●○●●●●
48	30	0110000	○●●○○○○
49	31	0110001	○●●○○○●
50	32	0110010	○●●○○●○
51	33	0110011	○●●○○●●
52	34	0110100	○●●○●○○
53	35	0110101	○●●○●○●
54	36	0110110	○●●○●●○

55	37	0110111	○●●○●●●
56	38	0111000	○●●●○○○
57	39	0111001	○●●●○○●
58	3A	0111010	○●●●○●○
59	3B	0111011	○●●●○●●
60	3C	0111100	○●●●●○○
61	3D	0111101	○●●●●○●
62	3E	0111110	○●●●●●○
63	3F	0111111	○●●●●●●
64	40	1000000	●○○○○○○
65	41	1000001	●○○○○○●
66	42	1000010	●○○○○●○
67	43	1000011	●○○○○●●
68	44	1000100	●○○○●○○
69	45	1000101	●○○○●○●
70	46	1000110	●○○○●●○
71	47	1000111	●○○○●●●
72	48	1001000	●○○●○○○
73	49	1001001	●○○●○○●
74	4A	1001010	●○○●○●○
75	4B	1001011	●○○●○●●
76	4C	1001100	●○○●●○○
77	4D	1001101	●○○●●○●
78	4E	1001110	●○○●●●○
79	4F	1001111	●○○●●●●
80	50	1010000	●○●○○○○
81	51	1010001	●○●○○○●
82	52	1010010	●○●○○●○
83	53	1010011	●○●○○●●
84	54	1010100	●○●○●○○
85	55	1010101	●○●○●○●
86	56	1010110	●○●○●●○
87	57	1010111	●○●○●●●
88	58	1011000	●○●●○○○
89	59	1011001	●○●●○○●

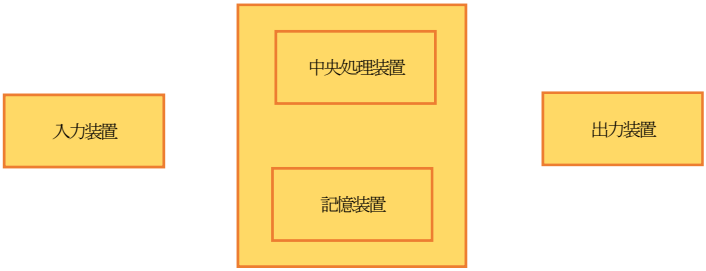
90	5A	1011010	●○●●○●○
91	5B	1011011	●○●●○●●
92	5C	1011100	●○●●●○○
93	5D	1011101	●○●●●○●
94	5E	1011110	●○●●●●○
95	5F	1011111	●○●●●●●
96	60	1100000	●●○○○○○
97	61	1100001	●●○○○○●
98	62	1100010	●●○○○●○
99	63	1100011	●●○○○●●
100	64	1100100	●●○○●○○
101	65	1100101	●●○○●○●
102	66	1100110	●●○○●●○
103	67	1100111	●●○○●●●
104	68	1101000	●●○●○○○
105	69	1101001	●●○●○○●
106	6A	1101010	●●○●○●○
107	6B	1101011	●●○●○●●
108	6C	1101100	●●○●●○○
109	6D	1101101	●●○●●○●
110	6E	1101110	●●○●●●○
111	6F	1101111	●●○●●●●
112	70	1110000	●●●○○○○
113	71	1110001	●●●○○○●
114	72	1110010	●●●○○●○
115	73	1110011	●●●○○●●
116	74	1110100	●●●○●○○
117	75	1110101	●●●○●○●
118	76	1110110	●●●○●●○
119	77	1110111	●●●○●●●
120	78	1111000	●●●●○○○
121	79	1111001	●●●●○○●
122	7A	1111010	●●●●○●○
123	7B	1111011	●●●●○●●
124	7C	1111100	●●●●●○○

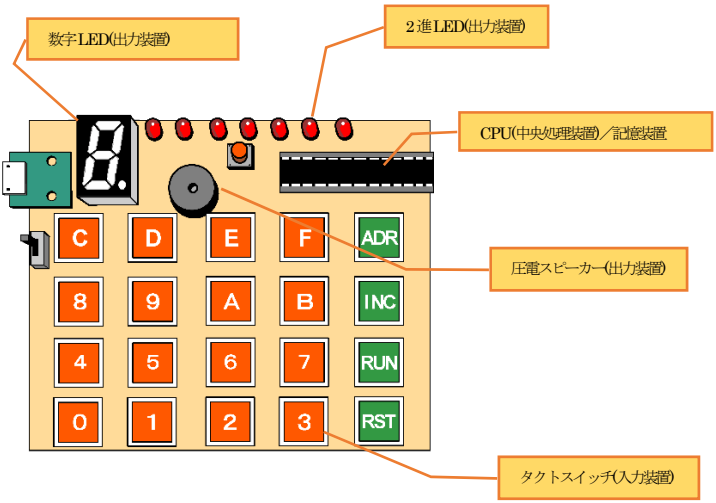
125	7D	1111101	●●●●●○●
126	7E	1111110	●●●●●●○
127	7F	1111111	●●●●●●●

2. ORANGE-4 の仕組み

●ORANGE-4 の構成●

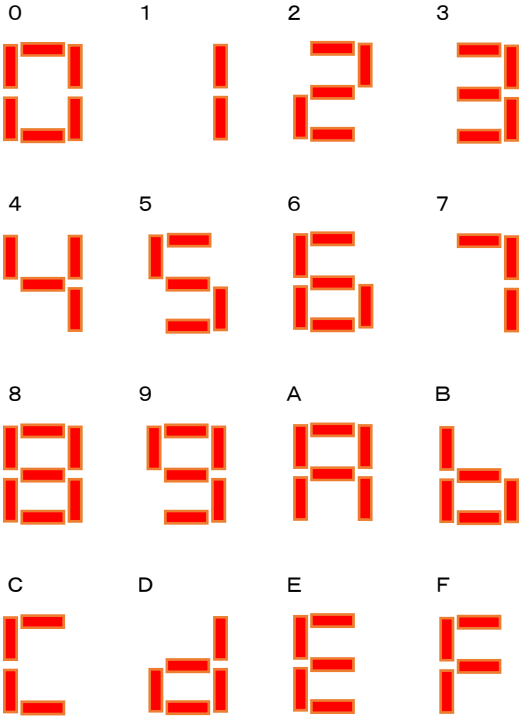
一般にコンピューターは中央処理装置と記憶装置、入力装置、出力装置で構成されています。





中央処理装置は、CPU(Central Processing Unit の略)のことで、データを移動したり、演算を行ったりする制御部です。記憶装置はメモリーのことで、データを保持するところです。ORANGE-4 では CPU とメモリーが一つの IC の中に入っています。

入力装置はデータは入力する装置で、ORANGE-4 では 20 個のタクトスイッチがこれにあたります。
出力装置はデータを出力する装置で、ORANGE-4 では 7 個の 2 進 LED、数字 LED、ブザーがこれにあたります。
数字 LED は次のような点灯状態で 1 から F の 16 進数を表わします。



●メモリーの中を見る●

メモリーの中は区画で区切られていて、それぞれの区画に番号が付いています。この番号のことをアドレスといいます。
ORANGE-4 のメモリーの中身を確認してみましょう。まず

RST

を押してください。

7 個の LED がすべて消灯し、数字 LED が F と表示されています。これはアドレス 00 番地の中身は F であることを意味します。7 個の LED はアドレスを(2 進数で)表しています。また、数字 LED がその番地のメモリーの中身を(16 進数で)表しています。

次に、

INC

を順次押してみます。初期状態では区画の中身はすべて F になっていることがわかります。

アドレス	アドレス(16 進数)	内容
000000	00	F
000001	01	F
000010	02	F
000011	03	F
000100	04	F
000101	05	F
000110	06	F
000111	07	F
001000	08	F
001001	09	F
001010	0A	F
001011	0B	F
001100	0C	F
001101	0D	F
001110	0E	F
001111	0F	F

●メモリーの中を書き換える●

それでは、次にメモリーの中身を書き換えてみましょう。

RST

を押します。ここで

0

を押すと、数字 LED の表示が F から 0 に変わります。この状態では、まだ仮決めですので、

INC

を押して確定します。

メモリー00 番地の内容は 0 に書き換えられました。

確定後、LED は次の番地である 01 番地を表示しています。数字 LED は 01 番地の中身である F を表示しています。

それでは続けて、01 番地から 0F 番地の中身をそれぞれ 1～F にしてみます。

1 INC

2 INC

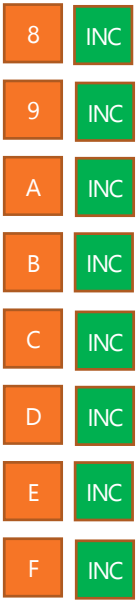
3 INC

4 INC

5 INC

6 INC

7 INC



メモリーの中身が書き換わっているかを確認してみましょう。

を押してから、

を順次押してみます。

初期状態では区画の中身はすべて F になっていたものが、次のように更新されていることがわかります。

アドレス	アドレス(16 進数)	内容
000000	00	0
000001	01	1
000010	02	2
000011	03	3

000100	04	4
000101	05	5
000110	06	6
000111	07	7
001000	08	8
001001	09	9
001010	0A	A
001011	0B	B
001100	0C	C
001101	0D	D
001110	0E	E
001111	0F	F

●特定番地からのデータ修正●

メモリーの中身を書き換えるには、先の方法では 00 番地から順に書き換えて行かなくてはいけませんでした。特定番地からデータを書き換えるには、以下のようにします。05 番地のデータを 5 から 9 に変更してみます。

RST

を押してから

0 5 ADR

と押します。（アドレスの指定は 2 桁の 16 進数で指定します。）

この状態で 2 進 LED は 0000101 を表示し、数字 LED は、05 番地の内容である 5 を表示しています。

ここで、

9 INC

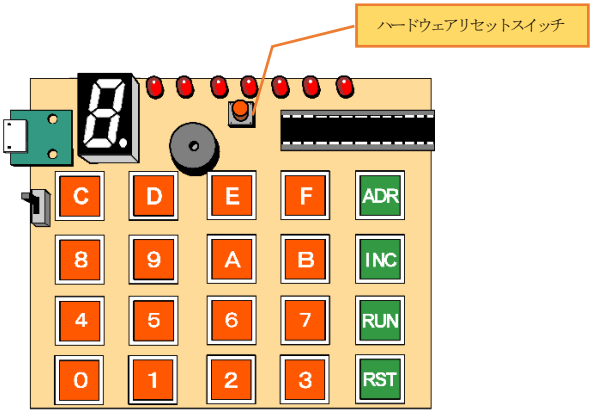
を押すと、05 番地のデータは 9 に更新されます。（2 進 LED は 0000110 を表示し、数字 LED は、06 番地の内容である 6 を表示しています。）

●データのクリア●

ブザーと CPU の間にあるオレンジ色の小さなスイッチはハードウェアリセットスイッチといい、このスイッチを押すとメリーの中のデータがすべて F になり、クリアされます。電源を入れ直すのと同じ効果があります。

RST

を押した場合は、メモリーの内容は消えずに 2 進 LED の表示が 00 番地に戻るだけです。



3. 機械語入門

●機械語とは？●

CPU はメモリーの 00 番地から順に読んで、その値に対応する動作をします。その数の羅列は、CPU のみに理解でき、人間から見ると、（通常は）全く理解できませんので、機械語とかマシン語と呼んでいます。
それでは、まず、（意味はわからないけれど）メモリーに機械語を入力してみましょう。

RST

A INC 6 INC E INC 1 INC

F INC 0 INC 0 INC

と入力します。入力し終えたら、

RST INC INC INC INC INC INC

と押して、00 番地から 06 番地までの内容が以下のようになっていることを確認します。

アドレス	アドレス(16進数)	内容
000000	00	A
000001	01	6
000010	02	E
000011	03	1
000100	04	F
000101	05	0
000110	06	0

入力確認ができたら、
メモリーの 00 番地からデータを順に実行するという指示をします。

RST 1 RUN

と入力します。一番左にある 2 進 LED が点灯します。

このメモリー内にあるA、6、E、1、F、0、0という16進数の羅列をCPUが解釈して、その結果2進LEDが点灯するのです。

ここで、

RST

を押すと、実行中のプログラムが停止しLEDが消灯します。

もう一度、

1 RUN

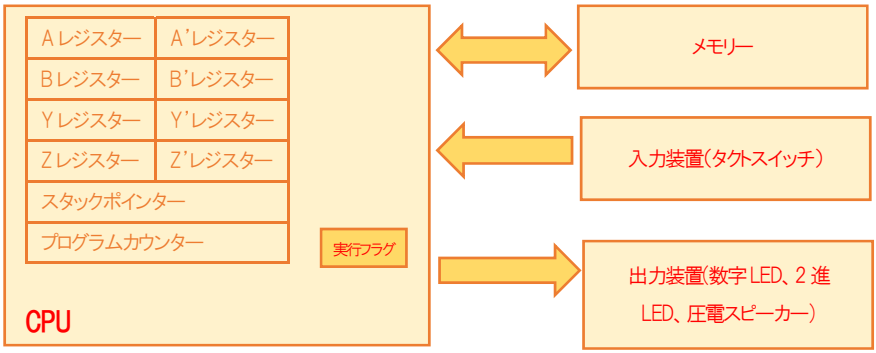
と入力すると、再び2進LEDが点灯します。

メモリー内のデータが変わっていないので、何度やっても同じ結果になります。

CPUはメモリーの00番地から順にA、6、E、1、F、0、0というデータを受け取ると、必ず一番左にある2進LEDを点灯します。

●CPUの中●

それでは、CPUの中を見て行きましょう。ORANGE-4のCPUの中には、レジスター、プログラムカウンター、スタックポインターなどがあります。

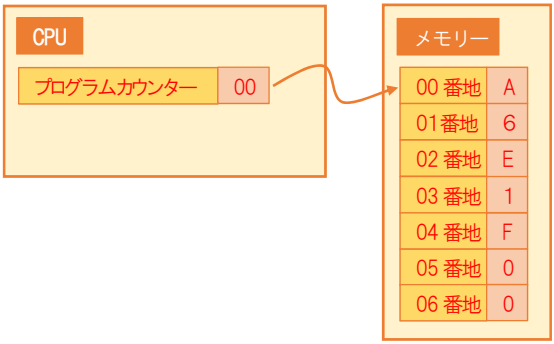


レジスターは、データを一時的に保持する場所です。ORANGE-4にはA、B、Y、Z、A'、B'、Y'、Z'という名前の付いたレジスターが合計8個あります。各レジスターは4ビット分の大きさなので0～Fのデータを蓄えておくことができます。

スタックポインターとプログラムカウンターは 8 ビット分の大きさなので、00～FF(0～127)までのデータが入ります。実行フラグは 1 ビット分しかありませんので 0 か 1 かの状態しか保持できません。

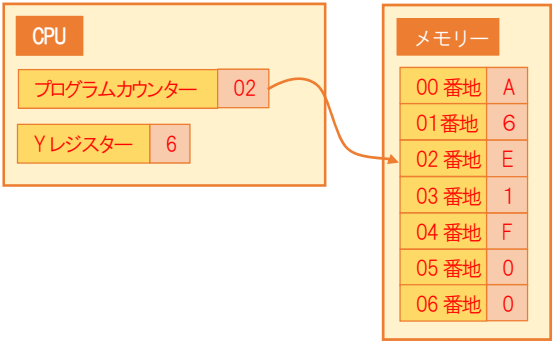
●CPU はどう動くの？●

電源 ON（またはハードウェアリセットスイッチが押されると）で、CPU 内のプログラムカウンターが 00 になります。次にプログラムカウンターの指しているメモリーの内容を取り込みます。すなわち、00 番地の内容である A を取り込みます。

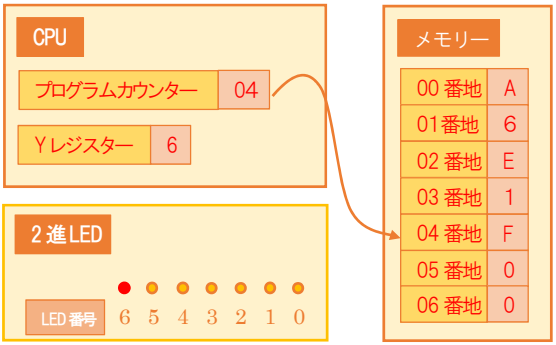


CPU は、A というデータを命令とみなして、決まった動作をします。(受け取った命令 A～F に対応して決まった動作をします。)

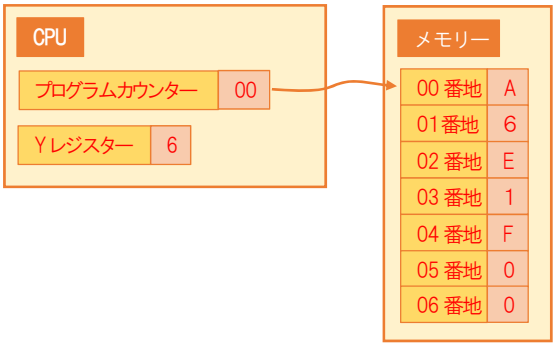
取り込んだ命令が A だった場合は、さらに次のデータを取り込みます。そして、取り込んだデータを Y レジスターに格納します。プログラムカウンターは 02 になります。「A、6」というデータの羅列は「Y レジスターの内容を 6 にする」という命令になります。



その次に02番地から命令を取り込みます。命令がEの場合は、さらに次のデータを取り込み、それが1だった場合は、Yレジスタの内容によって2進LEDを点灯します。現在のYレジスターの内容は6なので、それに対応する6番のLEDが点灯します。「E、1」というデータの羅列は「Yレジスターの内容に対応するLEDを点灯する」という命令になります。



さらに、04番地から命令を取り込みます。命令がFの場合は、05番地、06番地のデータを続けて取得し、その二つのデータを合わせてプログラムカウンターに設定します。「F、0、0」というデータの羅列は「プログラムカウンターを00にする」という命令になります。



プログラムカウンターが00になったということは、次に実行する命令は07番地からではなく、00番地からになります。すなわち、このプログラムは00番地から06番地までを繰り返し実行することになります。

4. アセンブリ言語入門

●アセンブリ言語とは？●

A というデータは、指定したデータを Y レジスターに格納する命令でした。「A、1」というデータ列なら、Y レジスターに 1 を格納するという命令になります。また、付録 1「ORANGE-4 の命令セット」を見ると、他にどんな命令があるかわかります。たとえば、8 というデータは、指定したデータを A レジスターに格納する命令です。「8、2」と続けば、A レジスターに 2 を格納するという命令になります。
付録 1「ORANGE-4 の命令セット」を見ながら、16 進数を打ち込んで行けば「機械語プログラミング」が可能です。
しかしながら、16 進数のデータ列では人間にはわかりにくいし覚えるのも大変です。(ORANGE-4 の命令セットは数が少ないのですべて覚えることは難しくありませんが、他の CPU は数多くの命令があり複雑です。)
そこで、通常はアセンブリ言語という人間にもわかりやすい略語で記述します。
たとえば、前章の機械語はアセンブリ言語で記述すると、以下のようになります。

```
①　　ldyi　　6
②　　scall　　1
③　　jmpf　　00
```

アセンブリ言語の 1 行は、「ニーモニック」という命令にあたる部分と「オペランド」というパラメーターにあたる部分からなります。

①の場合で言えば、ldyi がニーモニックで、オペランドが 6 です。そして、ldyi はオペランドで指定した値を Y レジスタに格納するという意味になります。したがって、Y レジスタの値は 6 となります。

②では、scall がニーモニックで、オペランドが 1 です。scall はオペランドで指定した番号のサービスを呼び出すという意味になります。ORANGE-4 では、サービスコールの 1 番は、Y レジスターに格納されている値に対応する 2 進 LED を点灯させます。

③では、jmpf がニーモニックで、これはオペランドで指定した値をプログラムカウンターに格納します。その結果、00 番地にジャンプすることになります。

●ハンドアセンブル●

アセンブリ言語でプログラムを書いたら、機械語に変換してメモリーに入力しないと動かしてみることができません。アセンブリ言語で書いたプログラムを手作業で機械語に変換することを「ハンドアセンブル」といいます。

付録1「ORANGE-4 の命令セット」を参照しながら、ハンドアセンブリで機械語に変換してみます。

①　　ldyi　　6

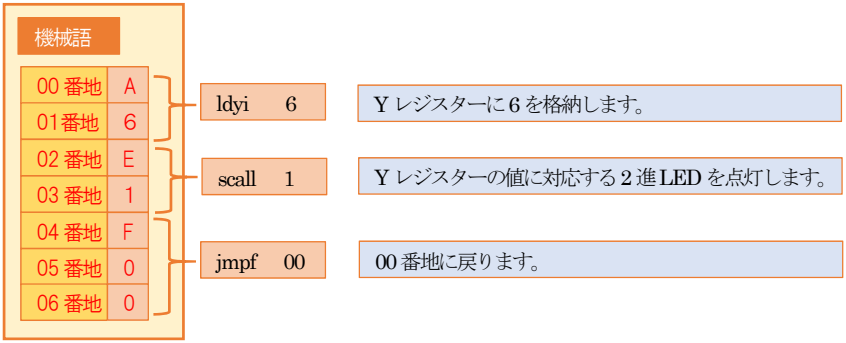
ニーモニック ldyi に対応する機械語（オペコード）はA なので、00 番地はA となります。オペランドは6 なので、01 番地は6 となります。

②　　scall　　1

ニーモニック scall に対応するオペコードはE なので、02 番地はE となります。オペランドは1 なので、03 番地は1 となります。

③　　jmpf　　00

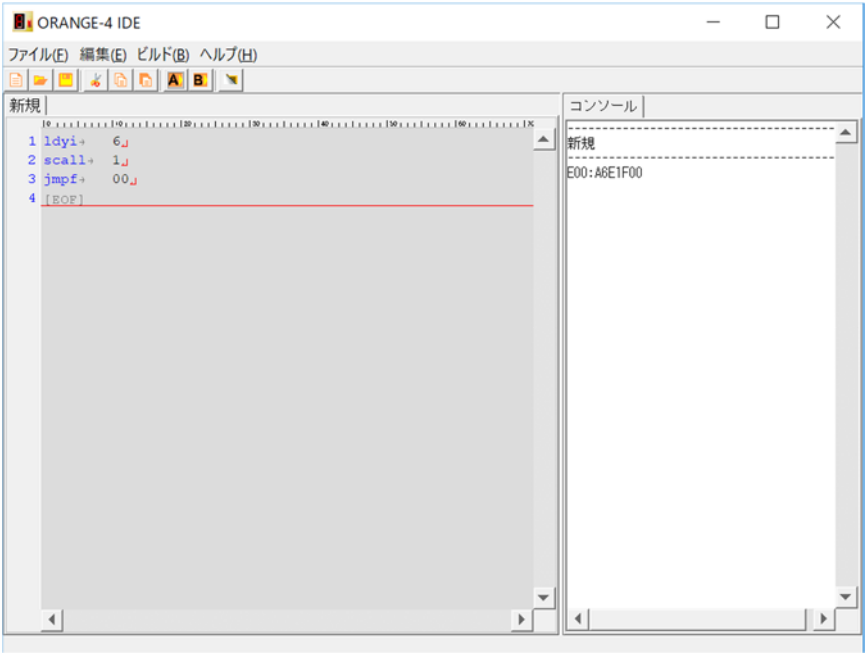
ニーモニック jmpf に対応するオペコードはF なので、04 番地はF となります。オペランドは00 なので、05 番地と 06 番地が 0 になります。



●アセンブラとは？●

アセンブリ言語のことを単にアセンブラと言うこともありますが、正しくはアセンブラとは「アセンブリ言語で書いたプログラムを機械語を自動で変換するツール」のことです。
ORANGE-4 のようなトレーニングキットでは自身の上で動作するアセンブラが付属していません。
他のコンピューターで動作するアセンブラをクロスアセンブラと言いますが、ORANGE-4 では、パソコン上で動作するクロスアセンブラが Web に公開されています。（http://www.picosoft.co.jp/ORANGE-4/）
パソコンを使用できる環境があれば、アセンブラを使用することができます。

アセンブラの起動には、あらかじめ Java8 をインストールしておく必要があります。（パソコンによっては、購入時からインストールされている場合があります。）
アセンブラ（ORANGE 4 .jar）を起動すると、ウインドウが表示されますので、左側のウインドウ内にアセンブリ言語で書いたプログラムを入力し、A をクリックすると右側に機械語が表示されます。先頭に E00: が付加していますが、実際の機械語はそれに続く「A6E1F00」の部分です。これを ORANGE-4 のメモリーに入力すれば実行できます。



●アセンブラの文法●

ハンドアセンブルのときは、独自のルールで書いても問題ありませんが、アセンブラを使用するときは、そのアセンブラのルールに沿って記述する必要があります。
たとえば、先のプログラムは以下のように書きます。

```
        org     0x00
start:
        ldyi    6       ;Y レジスターに 6 を格納
        scall   1       ;Y レジスターの値に対応する 2 進 LED を点灯
        jmpf    start   ;start に戻る
```

org は CPU への命令ではなく、アセンブラ疑似命令で指定番地から格納せよと言うアセンブラへの命令です。数字はそのまま書くと 10 進数と見なされます。先頭に 0x を付けると 16 進数と見なされます。上記の場合は 0x を付けても付けなくても同じで 00 番地から格納せよと言う意味になります。また、指定がない場合はアセンブラは 00 番地から格納しますので、この行をすべてしても省略してもかまいません。

strat:のようにコロン(:)で終わるものをラベルと言います。
セミコロン(;)から行末まではアセンブラが無視しますので、コメントとしてプログラムの説明を書いておくことができます。

jmpf のオペランドには、0x00 と書いてもかまいませんが、先ほどのラベル名を書くことができます。(start:ではなく start だけを記述します。)
先頭にジャンプするなら 00 番地だとすぐにわかりますが、途中にジャンプするときは何番地かすぐにわからないので、ラベルを使った方が便利です。

●エル L チカ●

LED を点滅させることを「L(エル)チカ」(LED チカチカの省略) と呼びます。LED の点灯と消灯を繰り返せば、L チカが実現できます。LED を消灯させるには、「scall」のオペランドに 2 を指定します。
点滅させる 2 進 LED も一番左の 6 番の LED から一番右の 0 番の LED に変えてみます。

```
start:
        ldyi    0
        scall   1
        scall   2
        jmpf    start
```

このプログラムでは、LED の点灯と消灯を高速で繰り返してしまい、「L チカ」になりません。
点灯と消灯の間に待ち時間を入れます。待ち時間はA レジスターで時間を指定して scall のオペランド指定 C を使います。

```
        org     0x00
start:
        ldyi    0       ;Y レジスターに 0 を格納
        scall   1       ;Y レジスターの値に対応する 2 進 LED を点灯
        ldi     9       ;A レジスターに 9 を格納
        scall   0xc     ;1 秒間処理間待ち
        scall   2       ;Y レジスターの値に対応する 2 進 LED を消灯
        ldi     9       ;A レジスターに 9 を格納
        scall   0xc     ;1 秒間処理間待ち
        jmpf    start   ;start に戻る
```

「待ち時間は (A レジスターの値 +1) × 0.1 秒です。」となっていますので、A レジスターに 9 を格納した場合はちょうど 1 秒間待つことになります。A レジスターに値を設定する命令は「ldi」です。
2 進 LED の点灯や消灯は Y レジスター、処理待ち時間はA レジスターと、使うべきレジスターが決められている場合があります。ORANGE-4 ではA レジスターと Y レジスターがよく使われ、その他のレジスターは補助的に使用します。
ところで、このプログラムの繰り返しの中でA レジスターの値は常に 9 で、Y レジスターの値は常に 0 です。したがって、これらの処理はループ（繰り返し）から外に出して、以下のように最適化できます。

```
        org     0x00
        ldyi    0       ;Y レジスターに 0 を格納
        ldi     9       ;A レジスターに 9 を格納
loop:
        scall   1       ;Y レジスターの値に対応する 2 進 LED を点灯
        scall   0xc     ;1 秒間処理間待ち
        scall   2       ;Y レジスターの値に対応する 2 進 LED を消灯
        scall   0xc     ;1 秒間処理間待ち
        jmpf    loop    ;loop に戻る
```

機械語　A089E1ECE2ECF04

これでLチカプログラムの完成です。ハンドアセンブルするか、アセンブラにかけて機械語に変換します。「A089E1ECE2ECF04」を ORANGE-4 のメモリーに入力すれば実行できます。LED の場所や点滅間隔を変更するのも簡単でしょう。

●数字 LED に 1~9 を表示する●

今度は、数字 LED を点灯させてみましょう。数字 LED を点灯させる命令を調べてみますと、「outn」という命令が「A レジスタの値（0～F）を数字 LED に表示します。」となっています。
数字 LED に 1～9 を表示するには、A レジスターに 1～9 を順に格納しながら「outn」を実行すれば実現できそうです。

```
ldi     1
outn
ldi     2
outn
～      ～
ldi     9
outn
```

このプログラムはA レジスターの値が一つずつ増えて行くことに着目しれば、「addi」命令を使用して

以下のように書き換えることができます。

```
        ldi     0

loop:
        addi    1
        outn
        jmpf    loop
```

「addi」命令は、「A レジスターにオペランドで指定した値を足します。」ということなので、オペランドに 1 を指定すればA レジスターが 1 ずつ増えて行くことになります。やがて、Aレジスターが 9 になり、A になり、Bになり、、、止まりません。
そこで、Aレジスターが 9 になったら止める処理を入れます。

```
        ldi     0

loop:
        addi    1
        outn
        cpi     9
        jmpf    loop

end:
        jmpf    end
```

「cpi」という命令は、「A レジスターの値とオペランドで指定した値が等しい場合は実行フラグが 0 になります。一致しない場合は実行フラグが 1 になります。」ということです。
この場合で言えば、A レジスターが 9 になると実行フラグが 0 になります。

実は、ORANGE-4 の「jmpf」命令は無条件ジャンプではなく、条件ジャンプと言われるものです。「jmpf」命令は「実行フラグが 1 の場合は、オペランドで指定したアドレスに制御を移します。実行フラグが 0 のときは、次のアドレスに制御を移し、実行フラグは 1 になります。」となっています。以前のプログラムで jmpf が無条件ジャンプのような振る舞いに見えたのは、jmpf の直前で実行フラグが常に 1 だったからです。

今回の場合は、jmpf 命令の直前が cpi 命令ですので、A レジスタが 1 から 8 までの間はラベル loop のアドレスに制御を移しますが、A レジスタが 9 になった瞬間、次の命令に制御を移します。
最後の
end:
　　jmpf　　end
の部分は、常に実行フラグが 1 の状態ですので、同じアドレスにジャンプし続けています。数字 LED が 9 になったまま、何も変化しないままになります。

このプログラムを実行すると、確かに数字 LED が 1 から 9 に高速で変化してしまうので、単に数字 LED に 9 を表示しただけのプログラムと何ら変わらないように見えるでしょう。
そこで、outn の前に処理待ち時間を入れれば、数字 LED が 1 から 9 まで変化するのが見えるようになります。しかし、困ったことがあります。
1 秒間の処理待ち時間を入れるには、
　　ldi　　9
　　scall　　0xc
を入れればいいわけですが、元のプログラムは A レジスターが 1 から 9 に変化しながら、ループの中で動作していました。ところが、「ldi　　9」を実行した瞬間に A レジスターが 9 になり、この前提がくずれてしまいます。
A レジスターの内容を破壊しないで処理待ち時間を入れるには、その前後で A レジスターを退避します。どこに退避するかが問題ですが、このプログラムでは Y レジスターを壊しても影響ないので、Y レジスターに退避します。「ay」命令が「A レジスターの値と Y レジスターの値を交換します。」となっています。
　　ay
　　ldi　　9
　　scall　　0xc
　　ay
この処理をループの先頭に入れれば、A レジスターを破壊しないで 1 秒間の処置待ちを実現できます。

```
        org     0x00
        ldi     0       ;A レジスターに 0 を格納

loop:
        ay              ;A レジスターを退避
        ldi     9       ;A レジスターに 9 を格納
        scall   0xc     ;1 秒間処理間待ち
        ay              ;A レジスターを復元
        addi    1       ;A レジスターを+1
        outn            ;A レジスターの内容を数字 LED に表示
        cpi     9       ;A レジスターが 9 になったか?
        jmpf    loop    ;A レジスターが 9 でないときは loop に戻る

end:
        jmpf    end     ; 常に end に戻る
```

機械語　80389EC3911C9F02F10

●10 秒タイマーを作る●

最後にプログラムを修正して 10 秒タイマーを作ります。最初に 10(A)を表示し、1 秒毎に音を刻みながら 9、8、7、6、5、4、3、2、1、0 と表示します。0 になったら終了音を出します。

音を出すのは scall を使用するだけなので簡単ですが、A レジスターの値を一つずつ減らすことが難関です。付録 1 「ORANGE-4 の命令セット」を見渡しても A レジスターから指定した値を引く命令が見つかりません。足し算はあっても引き算はないのです。

答えはオーバーフローを利用します。A レジスターの内容が F だった場合に+1 すると、桁上がりで 10(16 進数)になりますが、A レジスターは 4 ビット分しかないので、0 だけが残ります。

A レジスターに F を足した結果は以下のようになります。

A レジスターの値	A レジスターに F を加えたときの値
0	F
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
A	9
B	A
C	B
D	C
E	D
F	E

すなわち、F を加えることは 1 を引くことと同じになります。

このことを利用して、最終的なプログラムは以下のようになります。

```
        org     0x00
        ldi     0xa     ;A レジスターに 10 を格納
        outn            : 10(A)を表示

loop:
        ay              ;A レジスターを退避
        ldi     9       ;A レジスターに 9 を格納
        scall   0xc     ;1 秒間処理間待ち
        ay              ;A レジスターを復元
        addi    0xf     ;A レジスターを-1
        outn            ;A レジスターの内容を数字 LED に表示
        scall   9       ; ショート音を鳴らす
        cpi     0       ;A レジスターが 0 になったか?
        jmpf    loop    ;A レジスターが 0 でないときは loop に戻る
        scall   7       ; エンド音を鳴らす

end:
        jmpf    end     ; 常に end に戻る
```

機械語　8A1389EC39F1E9C0F03E7F15

ORANGE-4 の命令セット

オペコード (16 進数)	ニーモニック	オペランド	実行フラグ	機能
0	ink	-	0 または 1	押された数字キー（0~F）を A レジスターに格納します。キーが押されていない場合は実行フラグが 0、押されている場合は 1 になります。
1	outn	-	1	A レジスタの値（0~F）を数字 LED に表示します。
2	abyz	-	1	A レジスターの値と B レジスターの値を交換します。さらに、Y レジスターの値と Z レジスターの値を交換します。
3	ay	-	1	A レジスターの値と Y レジスターの値を交換します。
4	st	-	1	A レジスターの値を（50 + Y レジスタの値）番地に格納します。
5	ld	-	1	（50 + Y レジスタの値）番地の値を A レジスターに格納します。
6	add	-	0 または 1	A レジスターに（50+Y レジスタの値）番地の値を足します。足し算の結果、桁上がりがある場合は実行フラグが 1 になり、それ以外のときは 0 になります。
7	sub	-	0 または 1	（50 + Y レジスタの値）番地の値から A レジスターの値を引いた結果を A レジスター格納します。引き算の結果が負数になる場合は実行フラグが 1 になり、それ以外のときは 0 になります。
8	ldi	x	1	A レジスタにオペランドで指定した値を格納します。

9	addi	x	0 または 1	A レジスターにオペランドで指定した値を足します。足し算の結果、桁上がりがある場合は実行フラグが 1 になり、それ以外のときは 0 になります。
A	ldyi	x	1	Y レジスタにオペランドで指定した値を格納します。
B	addyi	x	0 または 1	Y レジスターにオペランドで指定した値を足します。足し算の結果、桁上がりがある場合は実行フラグが 1 になり、それ以外のときは 0 になります。
C	cpi	x	0 または 1	A レジスターの値とオペランドで指定した値が等しい場合は実行フラグが 0 になります。一致しない場合は実行フラグが 1 になります。
D	cpyi	x	0 または 1	Y レジスターの値とオペランドで指定した値が等しい場合は実行フラグが 0 になります。一致しない場合は実行フラグが 1 になります。
E	scall	x	0 または 1	オペランドで指定した機能番号のサービスコールを呼び出します。(別表参照)
F	jmpf	xx	1	実行フラグが 1 の場合は、オペランドで指定したアドレスに制御を移します。実行フラグが 0 のときは、次のアドレスに制御を移し、実行フラグは 1 になります。
F60	call	xx	1	(実行フラグにかかわらず) オペランドで指定したアドレスに制御を移します。
F61	ret	-	1	呼び出した call 命令の直後に制御を移します。

F62	pushA	-	1	A レジスターの値をスタックトップに積みます。
F63	popA	-	1	スタックトップから値を取り出し、A レジスターに格納します。
F64	pushB	-	1	B レジスターの値をスタックトップに積みます。
F65	popB	-	1	スタックトップから値を取り出し、B レジスターに格納します。
F66	pushY	-	1	Y レジスターの値をスタックトップに積みます。
F67	popY	-	1	スタックトップから値を取り出し、Y レジスターに格納します。
F68	pushZ	-	1	Z レジスターの値をスタックトップに積みます。
F69	popZ	-	1	スタックトップから値を取り出し、Z レジスターに格納します。
F70	ioctrl		1	Y レジスターで指定したポートの設定を行います。ポートの設定は A レジスターの値により決定します。 0: デジタル出力 2: デジタル入力
F71	out		1	Y レジスターで指定したポートに A レジスターの値(0 または 1)を出力します。ポートは ioctrl 命令でデジタル出力の設定をしておく必要があります。
F72	in		1	Y レジスターで指定したポートの状態を A レジスターに読み込みます。ポートは ioctrl 命令でデジタル入力の設定をしておく必要があります。

サービスコール

機能番号	実行フラグ	機能
0	1	数字 LED を消灯します。
1	1	Y レジスターの値（0～6）に対応する 2 進 LED を点灯します。
2	1	Y レジスターの値（0～6）に対応する 2 進 LED を消灯します。
4	1	A レジスターの全ビットを反転します。
5	1	A レジスター、B レジスター、Y レジスター、Z レジスターの値を、それぞれ、A’ レジスター、B’ レジスター、Y’ レジスター、Z’ レジスターの値と交換します。
6	0 または 1	A レジスターの値を 1 ビット右にシフトします。シフト前の値が偶数だった場合は実行フラグが 1 になり、奇数だった場合は実行フラグが 0 になります。
7	1	エンド音を鳴らします。
8	1	エラー音を鳴らします。
9	1	「ピッ」という短い音を鳴らします。
A	1	「ピー」という長い音を鳴らします。
B	1	A レジスターで指定した音階（1~E）の音を鳴らします。
C	1	A レジスターで指定した時間だけ処理を待ちます。待ち時間は（A レジスターの値 + 1）× 0.1 秒です。
D	1	5E 番地、5F 番地の値に対応する 2 進 LED を点灯します。5E 番地に LED 番号の下位 4 ビット、5F 番地に LED 番号の上位 3 ビットを設定しておきます。
E	1	（50 + Y レジスタの値）番地の値から A レジスター値の値を 10 進減算します。結果を（50 + Y レジスタの値）番地に格納します。繰り下がりがあれば（Y - 1)番地が 1 になります。繰り下がりがない場合は(Y - 1)番地が 0 になります。命令実行後は Y レジスターの値は 1 を引かれます。
F	1	（50 + Y レジスタの値）番地の値と A レジスターの 10 進加算を行い、結果を（50 + Y レジスタの値）番地に格納します。繰り上がりがあれ

		ば上位(Y － 1 番地)の桁に 1 を加えて行きます。命令実行後は Y レジスターの値は 1 を引かれます。