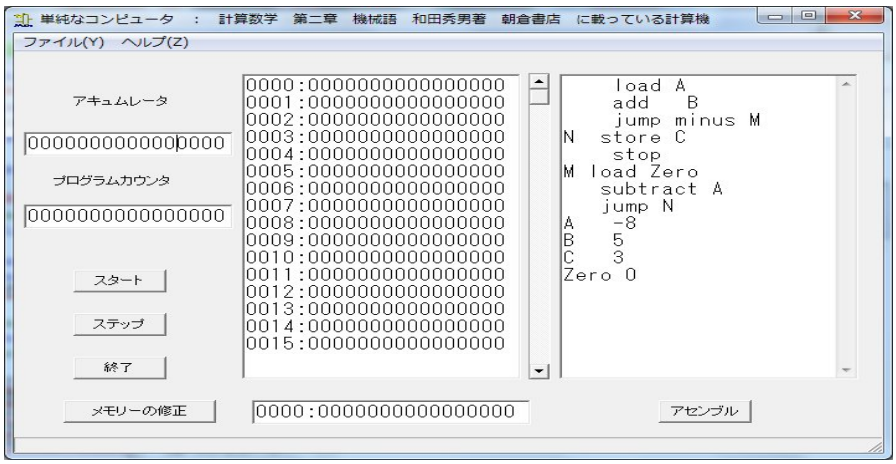
高知大学教育学部の情報数学のテキスト

文責　：　高知大学名誉教授　中村　治

機械語

和田秀男著「計算数学」朝倉書店　第二章「機械語」に載っている単純なコンピュータの仕組みをとうして、機械語とアセンブリ言語を見てみよう。



このコンピュータには**メモリ（記憶装置）**として８１９２語（１語＝１６ビット）があり、**演算装置**があり、演算装置で計算した結果を記憶する１６ビットからなる**アキュムレータ**と次にメモリの何処を実行するかを示す１６ビットからなる**プログラムカウンタ**がある。上のソフトには演算装置は表示されていない。アキュムレータとプログラムカウンタが左側に、メモリが中央にある。

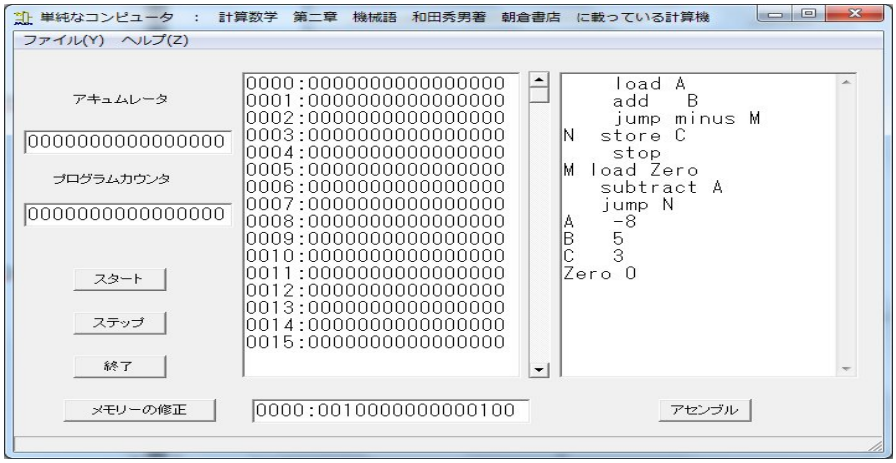
メモリにはコンピュータのプログラムとデータを入れる。コンピュータのプログラムをメモリに入れる方式が**プログラム内蔵方式**といわれ、フォン・ノイマンが考案したと言われている。

このコンピュータの命令は**オペコード**と呼ばれる命令の種類を表す数と**オペランド**と呼ばれるメモリの番地を示す数の組で表される。

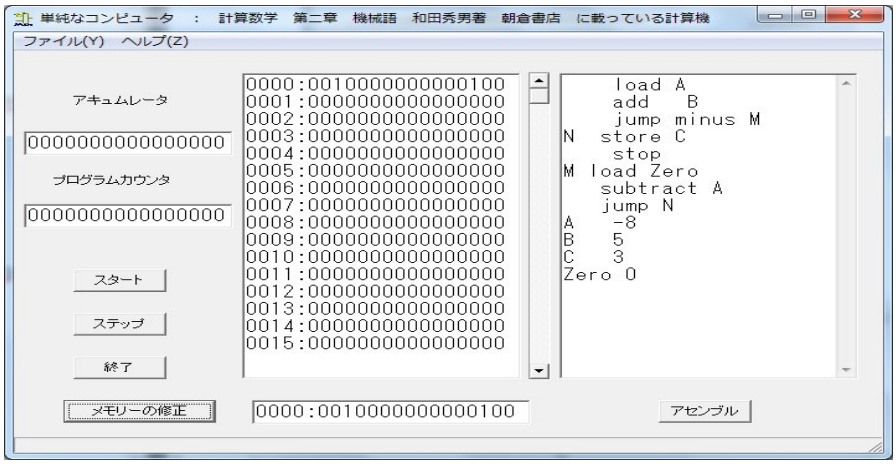
例えば１５と－７の和をこのコンピュータで計算したいときは次のようにする。

TinyCom を起ち上げる。

「メモリーの修正」ボタンを使って、メモリにデータをセットしていく。「メモリーの修正」ボタンの右のエディットに「0000:0010000000000100」と入力する。ここで、：の左の４桁が１０進数で番地を示し、：の右の１６桁が２進数でその番地の内容を示している。



「メモリーの修正」ボタンをクリックする。



メモリの一番上番地が「0000:0010000000000100」と変化した。ここで、：の左の４桁が１０進数で番地を示し、：の右の１６桁が２進数でその番地の内容を示している。

「メモリーの修正」ボタンの右のエディットに「0001:0110000000000101」と入力する。

「メモリーの修正」ボタンをクリックする。

メモリの２番目番地が「0001:0110000000000101」と変化した。

この操作を繰り返し、メモリが

```
0000:0010000000000100
0001:0110000000000101
0002:0100000000000110
0003:1010000000000000
0004:0000000000001111
0005:1111111111111001
0006:0000000000000000
```

となるようにする。

最初の４個が命令の列で、残りがデータです。データは１６ビットで整数を表し、正の整数は最初のビットを 0 とし残り１５ビットを使って２進数で表します。負の整数は**２の補数表示**という方式を使っていて、例えば -7 を表すには次のように変形して表現を求めます。

0000000000000111	7 を２進数表示する
1111111111111000	0 と 1 を入れ替える
1111111111111001	1 を足す

負の整数はこの操作で表現を求めます。これを**２の補数表示**といいます。**２の補数表示**の利点は数の表示が一意になることです。つまり、最初のビットが０なら正、最初のビットが１なら負とし、残りの１５ビットで絶対値を表現することにすると０の表現がの表現が 0000000000000000 と 1000000000000000　と二通り出来てしまい不便です。

このプログラムは、最初の 4 行が命令で、次の 3 行がデータで

0000:0010000000000100	4 番地のデータをアキュムレータに入れる
0001:0110000000000101	5 番地のデータをアキュムレータに加える
0002:0100000000000110	6 番地にアキュムレータの内容を保存
0003:1010000000000000	停止する
0004:0000000000001111	15 を表す
0005:1111111111111001	-7 を表す
0006:0000000000000000	0 を表す

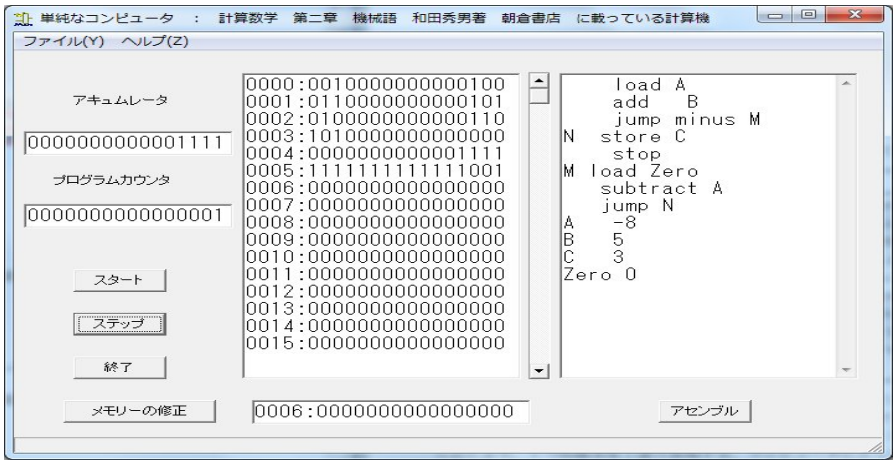
を意味します。

このプログラムを実行するには、プログラムカウンタが 0000000000000000 になっていることを確かめて、スタートのボタンをクリックします。

6 番地が 0000000000001000 となりました。すなわち、１５＋−7＝８です。プログラムカウンタが 0000000000000011 になりました。停止しろという命令で止まりました。

6 番地を 0000000000000000 にもどし、プログラムカウンタも 0000000000000000 に戻し、

ステップのボタンをクリックします。



アキュムレータが 0000000000001111 になり、プログラムカウンタが 0000000000000001 になりました。０番地の命令が実行されました。ステップのボタンをクリックするとプログラムカウンタが指している番地の命令が１つ実行されます。

このコンピュータの命令は７つしかないから１から７までの通し番号が付けられていて、２進法で表せば３ビットで表せ、メモリの番地は０から８１９１（$=2^{13}-1$）までなので１３ビットで表せる。それで、１語１６ビットのうち左３ビットに命令の番号を入れ、右の残り１３ビットにメモリの番地を入れることにより、１つの命令を１語で表現している訳です。データは１６ビットの２の補数で表現されています。

コンピュータにとってはこれでよいが、人間には２進数の数字の羅列を読むのは辛いものがあるので、人間に分かり易い言語としてアセンブリ言語が作られた。オペコードやオペランドを英単語や文字で表します。オペコードをアセンブリ言語で表したのが次の表の load, store, add, subtract, stop, jump および jump minus である。番地は文字や文字列で表す。

このコンピュータのオペコードは７つで、次の表の様になっている。

１０進数	２進数	アセンブリコード	説明
1	001	load	load M で M 番地の内容をアキュムレータにコピー
2	010	store	store M でアキュムレータの内容と M 番地にコピー
3	011	add	add M で M 番地の内容とアキュムレータの内容を加え、結果をアキュムレータにコピー
4	100	subtract	subtract M でアキュムレータの内容から M 番地の内容を引き、結果をアキュムレータにコピー
5	101	stop	stop でコンピュータの動作は停止する
6	110	jump	jump M で無条件で M の番地へ移動
7	111	jump minus	jump minus M でアキュムレータが負の時 M の番地へ移動

アセンブリ言語では上のプログラムを次のように表現する。

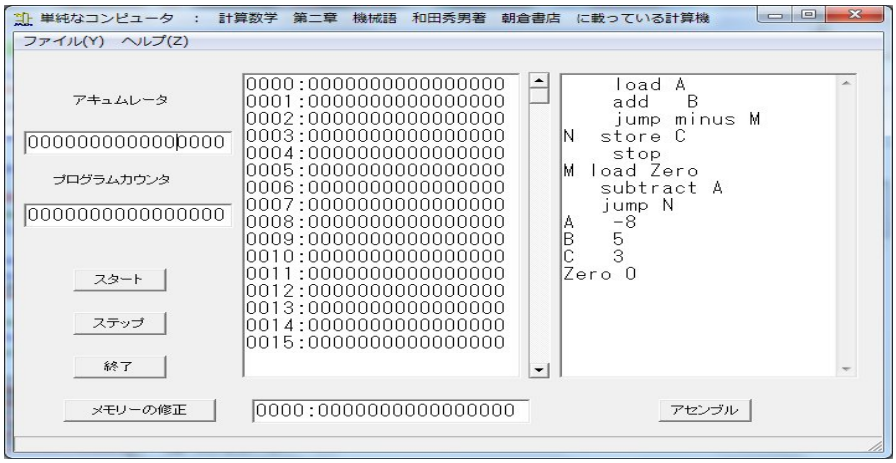
ラベル	オペコード	オペランド	対応する機械語
	load	A	0000:0010000000000100
	add	B	0001:0110000000000101
	store	C	0002:0100000000000110
	stop		0003:1010000000000000
A	15		0004:0000000000001111
B	-7		0005:1111111111111001
C	0		0006:0000000000000000

アセンブリ言語の設計はある程度の自由度があるが、**和田秀男著「計算数学」朝倉書店　第二章「機械語」**に載っている単純なコンピュータのアセンブリ言語、および TinyCom のアセンブリ言語の命令は

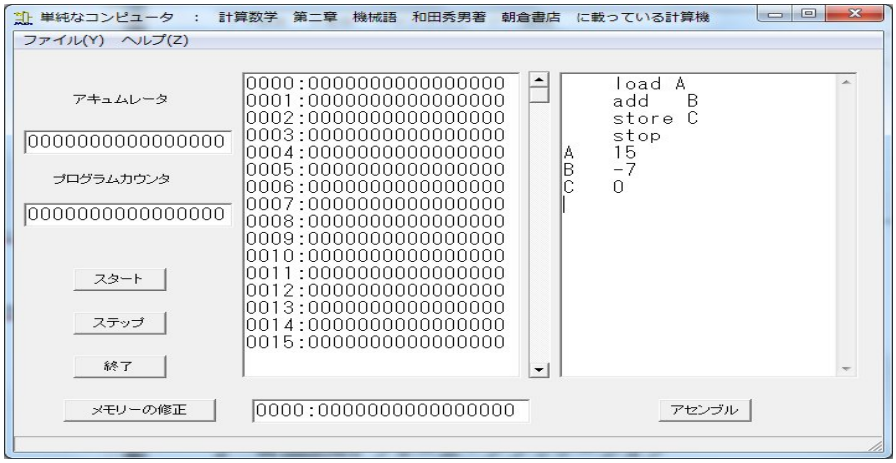
ラベル　オペコード　オペランド

からなり、ラベルやオペランドは空白のこともある。ラベルが空白でもラベル欄には半角スペースを入れる。**ラベルを記入するときはその前に空白を入れてはいけません !!!**

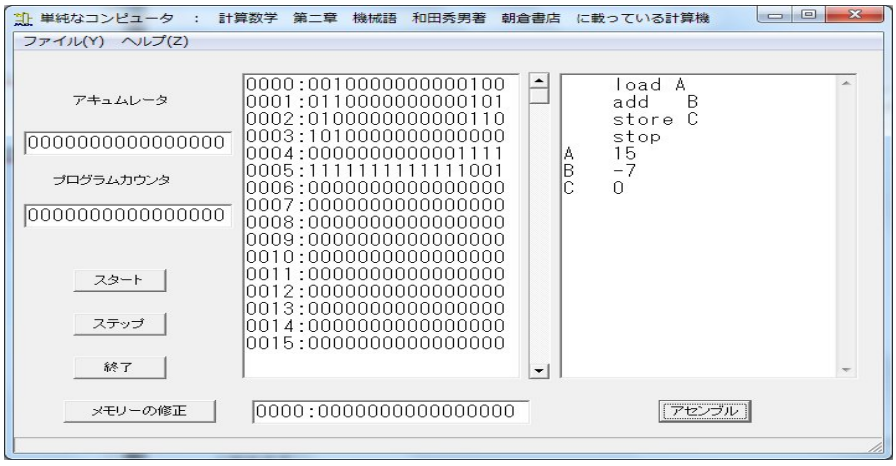
TinyCom を起ち上げ、



右側のメモを見るとアセンブリ言語のプログラムらしきものが書いてある。これはこのコンピュータの７つの命令をすべて使ったプログラムを書いているだけで無意味なプログラムです。これを消して、上のアセンブリ言語のプログラムを打ち込む。文字も空白もすべて半角文字や半角スペースを使います。**ラベルを記入するときはその前に空白を入れてはいけません !!!**



右下にある「アセンブル」ボタンをクリックします。



メモリに右側のアセンブリ言語のプログラムを機械語に翻訳されたプログラムがセットされました。アセンブリ言語のプログラムを機械語に翻訳するプログラムを**アセンブラ**と言います。コンピュータを使うのが随分楽になりました。このプログラムは、通常のコンピュータ言語では

```
C := A + B
```

と表現される命令を実行したことになります。

今後は TinyCom のアセンブリ言語のプログラムを調べます。

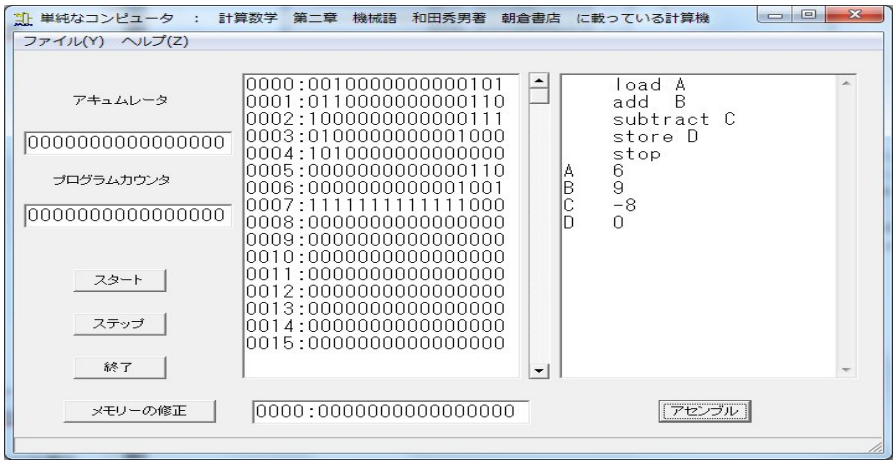
足し算をしましたが、引き算をするには subtract 命令を使います。subutract M と命令すれば、アキュムレータから M とラベル付けされた番地の内容を引き、その結果をアキュムレータに入れます。例えば、通常のコンピュータ言語では

```
D := A + B - C
```

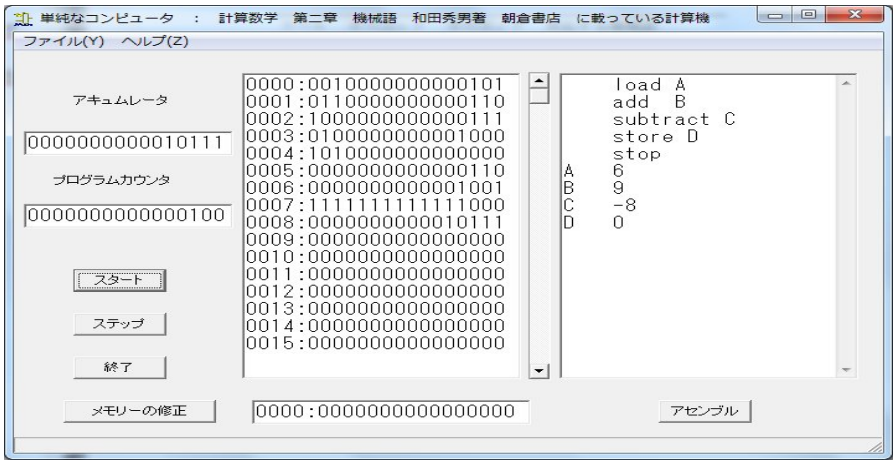
と表現される命令を実行するにはアセンブリ言語で次のようにプログラミングします。

ラベル	オペコード	オペランド	対応する機械語
	load	A	0000:0010000000000101
	add	B	0001:0110000000000110
	subtract	C	0002:1000000000000111
	store	D	0003:0100000000001000
	stop		0004:1010000000000000
A	6		0005:0000000000000110
B	9		0006:0000000000001001
C	-8		0007:1111111111111000
D	0		0008:0000000000000000

ラベル A, B, C, D を付けられた番地には好きな値を入れて下さい。今の場合は、$6+9-(-8)$ を計算しています。アセンブルし



スタートボタンをクリックすると



ラベル D で指示された 8 番地が 0000000000010111 になりました。$6+9-(-8)=23$ を計算しました。

問題：通常のコンピュータ言語では

```
A := 2*A - B - C
```

と表現される命令を実行するアセンブリ言語のプログラムを作れ。
　Ａ番地の内容を１６倍して、その結果を **A** 番地に保存する

```
A := 16 * A
```

にはアセンブリ言語では次のようにプログラミングします。

ラベル	オペコード	オペランド
	load	**A**
	add	**A**
	store	**A**
	add	**A**
	store	**A**
	add	**A**
	store	**A**
	add	**A**
	store	**A**
	stop	
A	**-3**	

A には好きな数字を入れます。**add** を１５回使うよりはプログラムが短くなっています。
　A 番地の内容と **B** 番地の内容を入れ替えるには補助の番地 **C** を使って、次のようにします。これは通常のコンピュータ言語では

```
C := A
A := B
B := C
```

と表現される。

ラベル	オペコード	オペランド
	load	**A**
	store	**C**
	load	**B**
	store	**A**
	load	**C**
	store	**B**
	stop	
A	**27**	
B	**-18**	
C	**0**	

A, B, C には好きな数字を入れます。これを

ラベル	オペコード	オペランド
	load	**A**
	store	**B**
	load	**B**
	store	**A**
	stop	
A	**27**	
B	**-18**	

としては駄目です。これは通常のコンピュータ言語では

```
B := A
A := B
```

としたことになり、**A** も **B** も元の **A** の値になってしまいます。

問題：$A \leftarrow B \leftarrow C \leftarrow A$ のプログラムを作れ。つまり、３つの値を入れ替えるプログラムを作れ。これは通常のコンピュータ言語では、補助の変数 **D** を使い

```
D := A
A := B
B := C
C := D
```

とすれば良いです。

jump minus M という命令は、アキュムレータが負の時、プログラムの行の先頭に **M** と書かれているところ（ラベルが **M** である命令の行）を次に実行しなさいと言う、条件分岐命令です。**jump M** という命令は、無条件にプログラムの行の先頭に **M** と書かれているところ（ラベルが **M** である命令の行）を次に実行しなさいと言う、無条件分岐命令です。これらの命令を使うと **A** 番地の内容の絶対値を **B** 番地に保存する（$B \leftarrow |A|$）プログラムは次のように書くことが出来ます。これは通常のコンピュータ言語では

```
if A < 0 then
    B := -A
else
    B := A
```

と普通表現されます。

ラベル	オペコード	オペランド
	load	**A**
	jump minus	**L0**
L1	**store**	**B**
	stop	
L0	**load**	**Zero**
	subtract	**A**
	jump	**L1**
Zero	**0**	
A	**-95**	
B	**0**	

ラベル **Zero** 番地は **0** でなければなりませんが、**A, B** 番地は好きな値で良いです。

Ａ番地の内容が **-95** の様に負の数であれば、**load A** を実行すれば、アキュムレータに負の数がセットされるので次の **jump minus L0** の命令で、**L0** とラベル付けされた番地の命令 **load Zero** を次に実行します。一方、Ａ番地の内容が **95** の様に正の数であれば、**load A** を実行すれば、アキュムレータに正の数がセットされるので次の **jump minus L0** の命令は無視され、次の **store B** の命令が次に実行されます。また **jump L1** の命令を実行すると次は **L1** とラベル付けされた番地の命令 **store B** が次に実行されます。このように **jump minus** と **jump** だけがプログラムの流れを変えます。

A 番地の値と **B** 番地の値の小さい方を **C** 番地に保存するプログラムは次のようになります。これは通常のコンピュータ言語では

```
if A < B then
    C := A
else
    C := B
```

と普通表現されます。

ラベル	オペコード	オペランド
	load	**A**
	subtract	**B**
	jump minus	**L0**
	load	**B**
L1	**store**	**C**
	stop	
L0	**load**	**A**
	jump	**L1**
A	**-9**	
B	**-12**	

問題：**A** 番地の値と **B** 番地の値の大きい方を **C** 番地に保存するプログラムを作れ。

TinyCom は足し算と引き算を知っているので、正の整数 **A** と正の整数 **B** の積を **P** にセットするためには、 **A * B = A + A + A + ... + A (B** 個）なので、**P** に **A** を入れ、**P** に **A** を **B - 1** 回加えればいい。つまり、

```
P := A
B := B -1
while B  >= 1
   B := B - 1
   P := P + A
```

とすればいいので、アセンブリ言語に変換すると

```
     load A
     store P
     load B
     subtract One
     store B
L1   load B
     subtract One
     jump minus L2
     store B
     load P
     add A
     store P
     jump L1
L2   stop
One  1
A    6
B    4
P    0
```

とすればよい。

正の整数 **A** を正の整数 **B** で割った商を **Q** に、余りを **R** に入れるには、**A** から **B** を **Q + 1** 回引くと負になるので

```
Q := 0
R := A
while  R - B >= 0
   R := R - B
   Q := Q + 1
```

とすればいいので、アセンブリ言語に変換するとプログラムは

```
     load Zero
     store Q
     load A
```

```
     store R
L1   load R
     subtract B
     jump minus L2
     store R
     load Q
     add One
     store Q
     jump L1
L2   stop
Zero 0
One  1
A    99
B    8
Q    0
R    0
```

とすればいいです。

正の整数 **A** を正の整数 **B** の最大公約数を **G** にセットするには、ユークリッドの互除法を使って、

```
while B > 0
   R := A mod B
   A := B
   B := R
G := A
```

なので、上の割り算のプログラムを使って、アセンブリ言語に変換するとプログラムは例えば、

```
M0   load B
     subtract One
     jump minus M2
     load A
     store A1
     load B
     store B1
     jump L0
M1   load B
     store A
     load R1
     store B
     jump M0
M2   load A
     store G
     stop
```

```
L0   load Zero
     store Q1
     load A1
     store R1
L1   load R1
     subtract B1
     jump minus L2
     store R1
     load Q1
     add One
     store Q1
     jump L1
L2   jump M1
A    120
B    425
G    0
Zero 0
One  1
A1   99
B1   8
Q1   0
R1   0
```

とすればいいです。**A1, B1, Q1, R1, G** の初期値は何でも良い。
　割り算のプログラムをサブルーチンとして使うには、

```
L2   jump M1
```

の **M1** をそのつど書き換えます。**L2** のアドレスの内容を戻るべきアドレス **???** を計算して **jump ???** となるよう書き換えます。戻るべきアドレスが今の場合 **11** 番地なので、例えば、

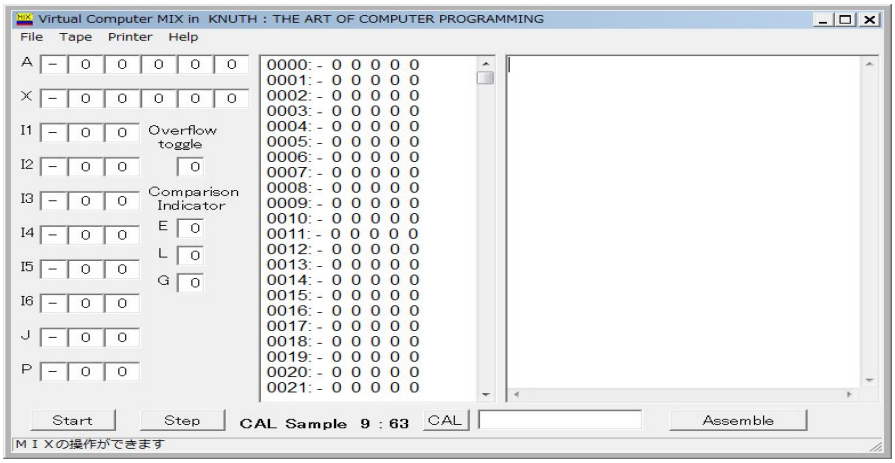
```
M0   load B
     subtract One
     jump minus M2
     load A
     store A1
     load B
     store B1
     load JMP
     add ELEVEN
     store L2
     jump L0
M1   load B
     store A
     load R1
```

```
     store B
     jump M0
M2   load A
     store G
     stop
L0   load Zero
     store Q1
     load A1
     store R1
L1   load R1
     subtract B1
     jump minus L2
     store R1
     load Q1
     add One
     store Q1
     jump L1
L2   jump M1
JMP  49152
ELEVEN 11
A    120
B    425
G    0
Zero 0
One  1
A1   0
B1   0
Q1   0
R1   0
```

とすればいいです。**A1, B1, Q1, R1, G** の初期値は何でも良い。

このように色々なことが出来ます。もっと複雑なプログラミングの例が知りたければ和田秀男著「計算数学」朝倉書店　第二章「機械語」を読んで下さい。この本には **TinyCom** の物理的な作り方も載っていて面白いです。

現実のコンピュータは構造が非常に複雑です。**Knuth** の **THE ART OF COMPUTER PROGRAMMING** で説明されている仮想のコンピュータは次の図の様になっています。

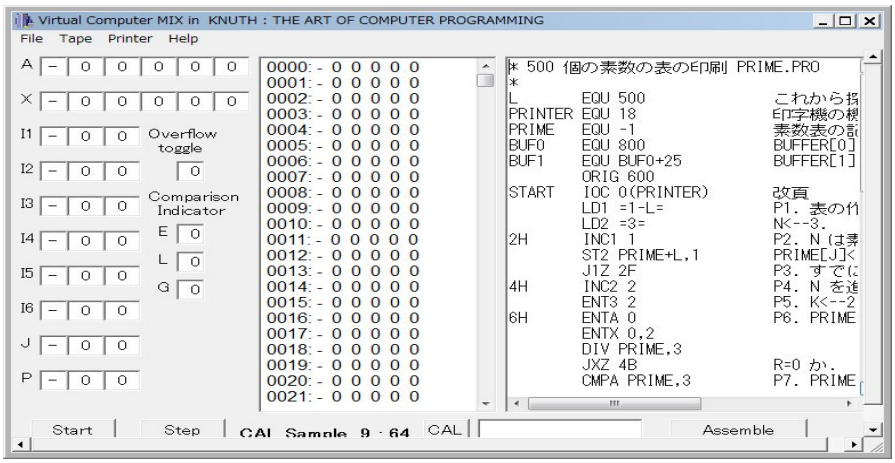


このコンピュータで 500 個の素数を求めるプログラムは

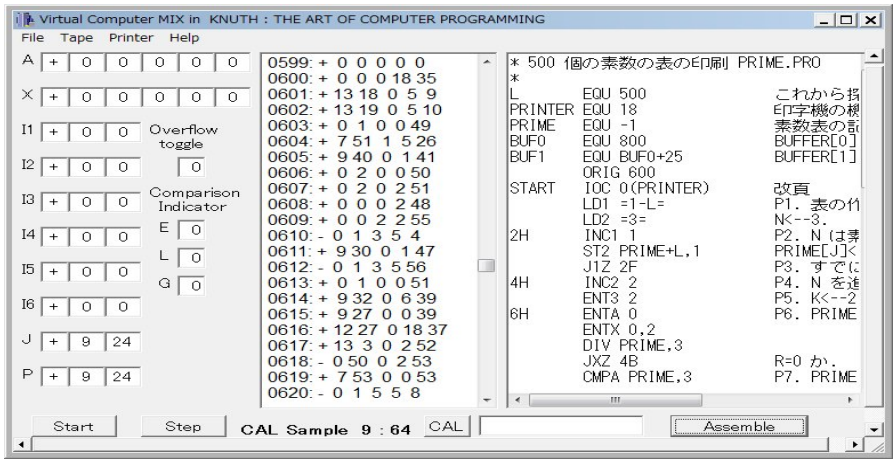
```
* 500 個の素数の表の印刷 PRIME.PRO
*
L       EQU 500             これから探す素数の個数
PRINTER EQU 18              印字機の機番
PRIME   EQU -1              素数表の記憶領域
BUF0    EQU 800             BUFFER[0] の記憶領域
BUF1    EQU BUF0+25         BUFFER[1] の記憶領域
        ORIG 600
START   IOC 0(PRINTER)      改頁
        LD1 =1-L=           P1. 表の作成開始. J<--1.
        LD2 =3=             N<--3.
2H      INC1 1              P2. N は素数, J<--J+1.
        ST2 PRIME+L,1       PRIME[J]<--N.
        J1Z 2F              P3. すでに 500 個見つけたか.
4H      INC2 2              P4. N を進める.
        ENT3 2              P5. K<--2.
6H      ENTA 0              P6. PRIME[K] | N か.
        ENTX 0,2
        DIV PRIME,3
        JXZ 4B              R=0 か.
        CMPA PRIME,3        P7. PRIME[K] の方が大きいか.
        INC3 1              P8. K を進める.
        JG 6B               Q>PRIME[K] なら飛び越し,
        JMP 2B              さもなければ N は素数.
2H      OUT TITLE(PRINTER)  P9. 見出しの印刷
        ENT4 BUF1+10        Set B<--1.
        ENT5 -50            Set M<--0.
2H      INC5 L+1            M を進める.
4H      LDA PRIME,5         P10. 一行分の印刷内容を入れる.
```

```
        CHAR
        STX 0,4(1:4)
        DEC4 1
        DEC5 50             (正の範囲で rI5 の内容が 50 個ずつ減る)
        J5P 4B
        OUT 0,4(PRINTER)    P11. 一行印刷.
        LD4 24,4            緩衝領域の切替.
        J5N 2B              rI5=0 なら完了.
        HLT
* INITIAL CONTENTS OF TABLES AND BUFFERS
        ORIG PRIME+1
        CON 2               最初の 2.
        ORIG BUF0-5
TITLE   ALF FIRST           表題の文字情報.
        ALF  FIVE
        ALF  HUND
        ALF RED P
        ALF RIMES
        ORIG BUF0+24
        CON BUF1+10         各緩衝領域が他の緩衝領域を参照.
        ORIG BUF1+24
        CON BUF0+10
        END START           終わり.
```

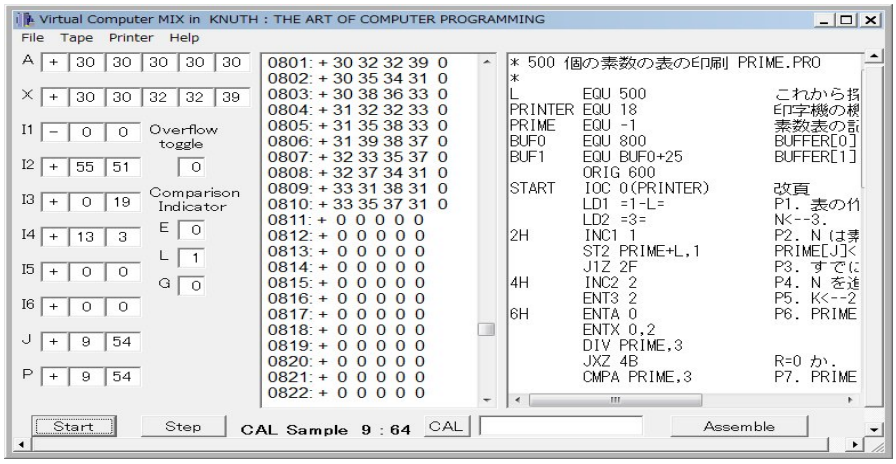
となります。このプログラムは **Knuth** の原本の翻訳とは異なりメモリ **1000** 語で計算できるように修正しています。修正した **MIX** のソフトはメモリ **4000** 語ですからそのまま使えます。これを **MIX** にロードし



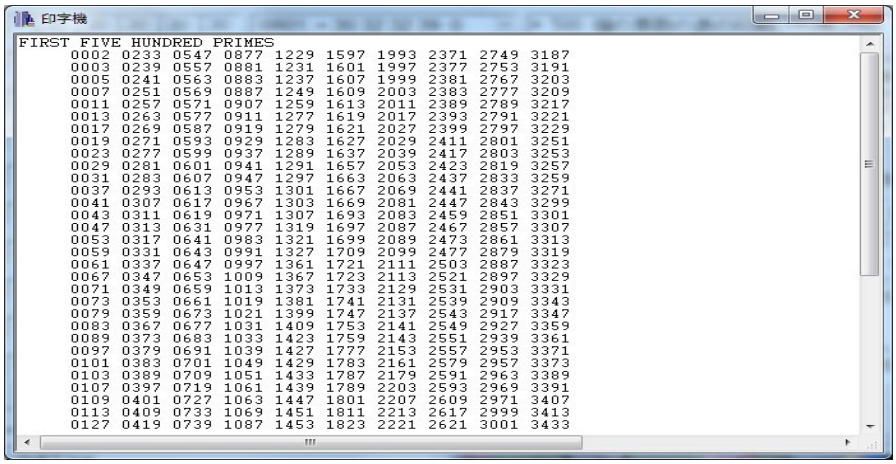
アセンブルし



実行し



Printer を表示すると



となります。

これも今となっては古いコンピュータです。**MIX** のプログラミングは、**Donard E. Knuth**「**The Art of Computer Programming Volumes 1-4A**」 にあります。この本はコンピュー

タの世界では最も有名な本です。翻訳もあります。アルゴリズムを集大成した本で、計算機科学者のバイブルです。**Donard E. Knuth** は、世界中の人が使っている複雑な数式や化学式を含む文章や楽譜や色々な言語の文章を作るためのソフト **TeX** の作者としても有名です。

現実に作られたパソコンの **CPU** の解説は、「はじめて読む８０８６」アスキー出版局。１９８７や「はじめて読む４８６」アスキー出版局。１９９４などがありますが、これも古いです。昔はこんな本を読んで、私もコンピュータの勉強をしましたが、今では機械語やアセンブリ言語の勉強をする必要はほとんどなくなりました。唯、機械語を知っていれば、**C** 言語の関数の引数やポインタや **CPU** のコアやスレッドの仕組みがそれなりに理解できます。

次は、昔私が作った **Borland** の **Turbo Assembler TASM 5.0J** による **8** 人の女王の問題を解くプログラムです。左側が **8086** の機械語で右側がアセンブリ言語によるプログラムです (勿論 **PENTIUM** でも同じアセンブリ言語のプログラムで動きます) 。対称性は考慮していません。**8** 人の女王の問題を拡張した **N** 人の女王の問題を **8** 人以下で解けます。

```
 1 0000                                 .MODEL  SMALL
 2 0000                                 .STACK  200H
 3 0000                                 .DATA
 4 0000  09*(??)                MM      DB      9 DUP(?)
 5 0009  09*(00)                JJ      DB      9 DUP(0)
 6 0012  10*(00)                KK      DB      16 DUP(0)
 7 0022  10*(00)                LL      DB      16 DUP(0)
 8 0032  ????                   I       DW      ?
 9 0034  ????                   N       DW      ?
10 0036  ????                   M       DW      ?
11 0038                                 .CODE
12 0000                        QUEENSSTART:
13 0000  B8 0000s                       MOV     AX,@DATA
14 0003  8E D8                          MOV     DS,AX
15 0005  B4 02                          MOV     AH,2
16 0007  B2 3F                          MOV     DL,'?'
17 0009  CD 21                          INT     21H
18 000B  B4 01                          MOV     AH,1
19 000D  CD 21                          INT     21H
20 000F  2C 30                          SUB     AL,'0'
21 0011  B4 00                          MOV     AH,0
22 0013  A3 0036r                       MOV     [M],AX
23 0016  B4 02                          MOV     AH,2
24 0018  B2 0D                          MOV     DL,0DH
25 001A  CD 21                          INT     21H
26 001C  B2 0A                          MOV     DL,0AH
27 001E  CD 21                          INT     21H
28 0020  C7 06 0032r 0001               MOV     [I],1
29 0026  C7 06 0034r 0000      L3:      MOV     [N],0
30 002C  FF 06 0034r           L4:      INC     [N]
```

```
31 0030  A1 0034r                      MOV     AX,[N]
32 0033  3B 06 0036r                   CMP     AX,[M]
33 0037  77 7F                         JA      L6
34 0039  BB 0009r                      MOV     BX,OFFSET JJ
35 003C  03 1E 0034r                   ADD     BX,[N]
36 0040  80 3F 01                      CMP     BYTE PTR [BX],1
37 0043  74 E7                         JE      L4
38 0045  BB 0012r                      MOV     BX,OFFSET KK
39 0048  03 1E 0032r                   ADD     BX,[I]
40 004C  2B 1E 0034r                   SUB     BX,[N]
41 0050  03 1E 0036r                   ADD     BX,[M]
42 0054  80 3F 01                      CMP     BYTE PTR [BX],1
43 0057  74 D3                         JE      L4
44 0059  BB 0022r                      MOV     BX,OFFSET LL
45 005C  03 1E 0032r                   ADD     BX,[I]
46 0060  03 1E 0034r                   ADD     BX,[N]
47 0064  4B                            DEC     BX
48 0065  80 3F 01                      CMP     BYTE PTR [BX],1
49 0068  74 C2                         JE      L4
50 006A  A1 0034r                      MOV     AX,[N]
51 006D  BB 0000r                      MOV     BX,OFFSET MM
52 0070  03 1E 0032r                   ADD     BX,[I]
53 0074  88 07                         MOV     [BX],AL
54 0076  BB 0009r                      MOV     BX,OFFSET JJ
55 0079  03 1E 0034r                   ADD     BX,[N]
56 007D  C6 07 01                      MOV     BYTE PTR [BX],1
57 0080  BB 0012r                      MOV     BX,OFFSET KK
58 0083  03 1E 0032r                   ADD     BX,[I]
59 0087  2B 1E 0034r                   SUB     BX,[N]
60 008B  03 1E 0036r                   ADD     BX,[M]
61 008F  C6 07 01                      MOV     BYTE PTR [BX],1
62 0092  BB 0022r                      MOV     BX,OFFSET LL
63 0095  03 1E 0032r                   ADD     BX,[I]
64 0099  03 1E 0034r                   ADD     BX,[N]
65 009D  4B                            DEC     BX
66 009E  C6 07 01                      MOV     BYTE PTR [BX],1
67 00A1  FF 06 0032r                   INC     [I]
68 00A5  A1 0032r                      MOV     AX,[I]
69 00A8  3B 06 0036r                   CMP     AX,[M]
70 00AC  77 03                         JA      L5
71 00AE  E9 FF75                       JMP     L3
72 00B1  8B 0E 0036r           L5:     MOV     CX,[M]
73 00B5  E8 004B                       CALL    PRINTRESULT
```

```
 74 00B8  FF 0E 0032r           L6:     DEC     [I]
 75 00BC  83 3E 0032r 00                CMP     [I],0
 76 00C1  74 3C                         JE      ENDQUEENS
 77 00C3  BB 0000r                      MOV     BX,OFFSET MM
 78 00C6  03 1E 0032r                   ADD     BX,[I]
 79 00CA  8A 07                         MOV     AL,[BX]
 80 00CC  32 E4                         XOR     AH,AH
 81 00CE  A3 0034r                      MOV     [N],AX
 82 00D1  BB 0009r                      MOV     BX,OFFSET JJ
 83 00D4  03 1E 0034r                   ADD     BX,[N]
 84 00D8  C6 07 00                      MOV     BYTE PTR [BX],0
 85 00DB  BB 0012r                      MOV     BX,OFFSET KK
 86 00DE  03 1E 0032r                   ADD     BX,[I]
 87 00E2  2B 1E 0034r                   SUB     BX,[N]
 88 00E6  03 1E 0036r                   ADD     BX,[M]
 89 00EA  C6 07 00                      MOV     BYTE PTR [BX],0
 90 00ED  BB 0022r                      MOV     BX,OFFSET LL
 91 00F0  03 1E 0032r                   ADD     BX,[I]
 92 00F4  03 1E 0034r                   ADD     BX,[N]
 93 00F8  4B                            DEC     BX
 94 00F9  C6 07 00                      MOV     BYTE PTR [BX],0
 95 00FC  E9 FF2D                       JMP     L4
 96 00FF                        ENDQUEENS:
 97 00FF  B4 4C                         MOV     AH,4CH
 98 0101  CD 21                         INT     21H
 99 0103                        PRINTRESULT     PROC
100 0103  BB 0000r                      MOV     BX,OFFSET MM
101 0106                        PRINTLOOP:
102 0106  43                           INC     BX
103 0107  B4 02                        MOV     AH,2
104 0109  8A 17                        MOV     DL,[BX]
105 010B  80 C2 30                     ADD     DL,'0'
106 010E  CD 21                        INT     21H
107 0110  B2 20                        MOV     DL,' '
108 0112  CD 21                        INT     21H
109 0114  E2 F0                        LOOP    PRINTLOOP
110 0116  B4 02                        MOV     AH,2
111 0118  B2 0D                        MOV     DL,0DH
112 011A  CD 21                        INT     21H
113 011C  B2 0A                        MOV     DL,0AH
114 011E  CD 21                        INT     21H
115 0120  C3                           RET
116 0121                        PRINTRESULT     ENDP
```

```
117                                    END      QUEENSSTART
```

プログラミングの歴史を「闘うプログラマー」[上] **G** ・パスカル・ザカリー著山岡洋一訳　日経 **BP** 出版センターから引用してみよう。

「初期のコンピュータ・プログラムは、特定のコンピュータを補助するものにすぎなかった。第二次世界大戦以前には、機械式の計算機がほとんどで、スイッチを切り替えたり、配線を変えたり、歯車をうごかすのがプログラミングの仕事にあたっていた。**1930** 年代には、当時の最強の機械式計算機、微分解析機で新しい問題を解く準備をするのに、何日もかかった。それから十年たって、デジタル・コンピュータができたが、むずかしい問題を解くには、セットアップに何日もかかった。 パンチ・カードや紙テープに記録された要求を読みとれるコンピュータができたが、カードやテープは、人間の手で機械に送り込まなければならなかった。プログラミングはまだ初期の段階にあり、進歩が必要だった。 画期的な進歩がおとずれたのは、**1944** 年であった。ハンガリー生まれのアメリカの数学者、ジョン・フォン・ノイマンが、プログラム記憶方式という考えを発展させた。プログラムを記憶させるという考え方自体は、この分野ではだれでも考えていたことだが、ノイマンはとくにその重要性をはっきりととらえていた。プログラム記憶方式では、コンピュータへの命令は、コンピュータ自体のメモリーに保存され、データとおなじようにあつかわれる。こうすれば、ごく短時間でプログラムを立ち上げられ、プログラムを修正したり、いくつかのプログラムを切り替えるのも簡単になる。 プログラム記憶方式は、初期のコンピュータ文化の中に広がり、プログラミングが爆発的に増え、この方式を支持する人が急増した。しかし、道のりはけわしかった。デジタル・コンピュータには、オンかオフの二つの状態しかないため、**1**（オン）と **0**（オフ）でできた二進法のメッセージにしか応答できない。 プログラムはすべて、この二つの数字だけで表さなければならないので、 ごく普通の数式が、すぐに、目がくらむほど複雑になる。**1940** 年代後半には、 コンピュータのプログラミングは「気が狂うほどむずかしい」と言われた。 まもなく、二進法の文字列を、もっと簡単につくる方法が考え出された。最初に発明されたのは、二進コードを自動的に打ち出す特殊なタイプライターだ。つぎに、もっと応用範囲の広い「アセンブリー」言語ができた。これは、文字や記号で **0** と **1** を表すものだ。アセンブリーでプログラムを書くようになったのは進歩だが、それでも、融通のきかないコンピュータの命令セットに忠実に従って書く必要がある。アセンブリー・コードを書きこなすには、命令セットを完璧に知っていなければならない。それに、命令セットは、プロセッサーの設計に依存するため、コンピュータの機種によってちがっている。つまり、苦労して覚えたアセンブリー言語の知識も、そのコンピュータが使われなくなれば、無駄になる可能性がある。 **1950** 年代初期には、コンピュータの利用が多い組織、とくにアメリカの陸海空軍は、ソフトウエアが頭痛のタネであり、コストがかかることに気づくようになった。一流のプログラマーは、プログラムを書きやすくする方法を模索していた。**1951** 年、アメリカ海軍予備役補給局の数学者、グレース・マレー・ホッパーが、コンパイラを開発した。コンパイラは、プログラマーの命令を、コンピュータを制御する **1** と **0** の文字列、つまりマシン語に翻訳するプログラムである。コンパイラで、ハードウエアの束縛や頭の痛い二進コードから、プログラマは解放されるようになった。 ホッパーが突破口を開いた後、プログラミングを簡単にするために、さまざまな努力が行われるようになった。その中でおそらくもっとも重要なものは、 **IBM** が開発したコンパイラ、**FORTRAN** だろう。**FORTRAN** には、**PUNCH, READ DRUM, IF DIVIDE CHECK** など三十二種類の命令があり、それぞれに、コンピュータに必要な正確なマシン語が対応している。**1950** 年代後半、**FORTRAN** の影響力は絶大であった。あるコンピュータ史研究家はこう述べている。「論理的に考えられる人なら、だれでも、やる気があればコ

ンピュータのプログラミングを学べるようになった。コンピュータの内部構造やむずかしいアセンブリー言語を熟知したスペシャリストである必要はなくなった。**FORTRAN** の簡単なコマンドだけを使えば、コンピュータを命令どおりにうごかせる。コンパイラーが自動的に、命令を正しいマシン・コードに翻訳してくれる」 **FORTRAN** によって、おなじ命令セットを使って、何種類ものコンピュータ用にプログラミングできるようになったが、種類のちがうマシンで使うには、プログラムを変更しなければならないことが多かった。それに、**FORTRAN** は科学技術の問題を解決する事を目的としている。他の用途のために、**COBOL** などの言語が開発された。やがて、コンピュータ言語のジャングルが、プログラマにとって、新たな頭痛のタネになった。どの言語を得意とするかによって、キャリアが決まるようになった。」

私がコンピュータを勉強しだした４０年ぐらい前は、コンピュータを勉強することは **FORTRAN**（**FORmulaTRANslater**）を勉強することでした。最初に実用化されたプログラミング言語は **FORTRAN** ですが、これは数値計算のための言語で、私が最初に勉強し、**20** 年間使っていた言語ですが、今では古い言語です。宇宙開発のソフトなど開発している「真のプログラマ」は **FORTRAN** とアセンブリ言語であらゆるプログラムを作るといいます。

FORTRAN は科学技術計算に使われ、事務計算には **COBOL** という言語が使われていました。**LISP (LISt Processor)** は **1960** 年ごろ **J.McCarthy** によって開発されたリストを基本データ構造とする関数型の言語です。**LISP** は人工知能のためのアセンブラといわれリスト処理に優れています。論理型のプログラミング言語 **PROLOG (PROgramming in LOGic)** は **1972** 年に **A. Colmerauer** によってフランスで開発されます。他の言語のように手続きを記述するのではなく、事実（物事の論理的関係）を記号論理に基づいて記述し、パターン・マチィングとバックトラッキングでプログラムを実行します。**LISP** と同じくリスト処理に優れ人工知能のための言語で、日本の第五世代コンピュータの開発用言語としても使われました。しばらく忘れ去られていましたが、**Bruce A. Tate** 著「**7** つの言語　**7** つの世界」 **Ruby, Io, Prolog, Scala, Erlang, Clojure, and Haskell** 平成 **23** 年　**3200** 円の影響で再び注目されています。

1975 年 **1** 月、米「ポピュラー・エレクトロニクス」誌で紹介された８ビット・マイコンキットである「アルテア」が経営難に陥っていた **MITS** 社から通信販売されます。**397** ドルのキットに購入申し込みが殺到し、１年分と見込んだ個数を初日で突破します。非常に高価で、専門教育を受けた一部のエリートしか触ることのできなかったコンピュータを個人が我が手にすることができるということで、医者や弁護士といった人たちが競って購入したそうです。若き日のビル・ゲイツと友人のポール・アレンがこのアルテア用の **BASIC** インタープリタを開発します。これが圧倒的人気を得ます。これより一般の人がマイコン（当時はパソコンのことをマイコンと言っていました）で使うプログラミング言語は **BASIC** になりました。高等学校の数学 **A**，**B**，**C** の教科書にも **BASIC** のプログラムが大量に載っていましたが、**BASIC** の処理系もなくなり、センター試験のコンピュータの問題も実際の **BASIC** の処理系でなく、 **JIS** 規格を基準として問題を作るようになり、情報という教科の導入も相まって、指導要領が改訂され、数学の教科書から **BASIC** は無くなりました。**BASIC** の次にマイコンのプログラミング言語として出てきたのは、**GOTO** 文有害論が出て、それの対策として開発された構造的プログラミング言語 **PASCAL** のマイコン版 **PASCAL(TURBO PASCAL)** です。大学では、**PASCAL** でプログラミング教育をしていた時代もありますが、純粋な **PASCAL** の処理系も現在では無く（**FREE PASCAL** と言うのがありますが、オブジェクト指向言語の **DELPHI** の代用品です）、ワークステーションの言語として生まれ、すべてのコンピュータに普及した **C** を最初の言語として教えているところが日本の大学では多いです。アメリカの大学では、例えば、 **MIT** では **LISP** の方言である **Scheme** で教育しています。構造的プログラミング言語の次に、オブジェクト指向言語と言われる **SMALLTALK**

や **C++** や **Java** が出てきます。計算機でできることはすべて **C** または **C++** でプログラミングできますが、コンピュータの性能が良くなったので、インターネットがらみで、**Perl** や **Ruby** や **Python** や **Erlang** や **Clojure** や **Scala** や **Haskell** といた言語ができます。

この授業では、中学校・高等学校の数学の先生になるためのプログラミング教育なので、本格的なプログラミング教育ではなく、代表的なプログラミング言語達の紹介と数学 **A, B, C** の教科書に載っていた **BASIC** のプログラムのアルゴリズムが理解でき、どれかのプログラミング言語でプログラミングできることを目指します。面白いプログラミング言語が見つかれば自分で勉強して下さい。

現在の主流のプログラミング言語には手続き型プログラミング言語、関数型プログラミング言語、論理型プログラミング言語、オブジェクト指向プログラミング言語があります。オブジェクト指向プログラミング言語は難しいので、手続き型プログラミング言語の代表として **C** 言語（一部 **C++** の機能を使います)、関数型プログラミング言語の代表として **Scheme**、論理型プログラミング言語の代表として **Prolog** の基本を学びます。