

T.T.L. T.T.L. T.T.L.

MZ-80K/C/1200/700/1500  
MZ-80B/2000/2200.X1

## 特集2 クリーンコンピュータと言語

# INTERPRETER

マイコン黎明期、4K BASIC、8K BASICなどという整数型のイン

タブリタが数々あったそう。多少不便ではあったが、メモリにゆとり

のない当時のマニアにとって、それはそれはステキな言語だったそう。

超巨大な、超肥大化したBASICやゲームが当たり前のように思ってい

る現代の人々に、あの当時のマニアたちの楽しみをもう一度……

## TTL VERSION 1.1 (C)MIKA ##

```
*READY  
*EXEC END  
*READY  
*EXEC END  
*READY  
1000  
1010  
1020  
1030  
1040  
1050  
1060  
1070  
1080  
1090  
1100  
1110  
1120  
1130  
1140  
1150  
1160  
1170  
1180  
1190  
1200  
1210  
1220  
1230  
1240  
1250  
1260  
1270  
1280  
1290  
1300  
1310  
1320  
1330  
1340  
1350  
1360  
1370  
1380  
1390  
1400  
1410  
1420  
1430  
1440  
1450  
1460  
1470  
1480  
1490  
1500  
1510  
1520  
1530  
1540  
1550  
1560  
1570  
1580  
1590  
1600  
1610  
1620  
1630  
1640  
1650  
1660  
1670  
1680  
1690  
1700  
1710  
1720  
1730  
1740  
1750  
1760  
1770  
1780  
1790  
1800  
1810  
1820  
1830  
1840  
1850  
1860  
1870  
1880  
1890  
1900  
1910  
1920  
1930  
1940  
1950  
1960  
1970  
1980  
1990  
2000  
2010  
2020  
2030  
2040  
2050  
2060  
2070  
2080  
2090  
2100  
2110  
2120  
2130  
2140  
2150  
2160  
2170  
2180  
2190  
2200  
2210  
2220  
2230  
2240  
2250  
2260  
2270  
2280  
2290  
2300  
2310  
2320  
2330  
2340  
2350  
2360  
2370  
2380  
2390  
2400  
2410  
2420  
2430  
2440  
2450  
2460  
2470  
2480  
2490  
2500  
2510  
2520  
2530  
2540  
2550  
2560  
2570  
2580  
2590  
2600  
2610  
2620  
2630  
2640  
2650  
2660  
2670  
2680  
2690  
2700  
2710  
2720  
2730  
2740  
2750  
2760  
2770  
2780  
2790  
2800  
2810  
2820  
2830  
2840  
2850  
2860  
2870  
2880  
2890  
2900  
2910  
2920  
2930  
2940  
2950  
2960  
2970  
2980  
2990  
3000  
3010  
3020  
3030  
3040  
3050  
3060  
3070  
3080  
3090  
3100  
3110  
3120  
3130  
3140  
3150  
3160  
3170  
3180  
3190  
3200  
3210  
3220  
3230  
3240  
3250  
3260  
3270  
3280  
3290  
3300  
3310  
3320  
3330  
3340  
3350  
3360  
3370  
3380  
3390  
3400  
3410  
3420  
3430  
3440  
3450  
3460  
3470  
3480  
3490  
3500  
3510  
3520  
3530  
3540  
3550  
3560  
3570  
3580  
3590  
3600  
3610  
3620  
3630  
3640  
3650  
3660  
3670  
3680  
3690  
3700  
3710  
3720  
3730  
3740  
3750  
3760  
3770  
3780  
3790  
3800  
3810  
3820  
3830  
3840  
3850  
3860  
3870  
3880  
3890  
3900  
3910  
3920  
3930  
3940  
3950  
3960  
3970  
3980  
3990  
4000  
4010  
4020  
4030  
4040  
4050  
4060  
4070  
4080  
4090  
4100  
4110  
4120  
4130  
4140  
4150  
4160  
4170  
4180  
4190  
4200  
4210  
4220  
4230  
4240  
4250  
4260  
4270  
4280  
4290  
4300  
4310  
4320  
4330  
4340  
4350  
4360  
4370  
4380  
4390  
4400  
4410  
4420  
4430  
4440  
4450  
4460  
4470  
4480  
4490  
4500  
4510  
4520  
4530  
4540  
4550  
4560  
4570  
4580  
4590  
4600  
4610  
4620  
4630  
4640  
4650  
4660  
4670  
4680  
4690  
4700  
4710  
4720  
4730  
4740  
4750  
4760  
4770  
4780  
4790  
4800  
4810  
4820  
4830  
4840  
4850  
4860  
4870  
4880  
4890  
4900  
4910  
4920  
4930  
4940  
4950  
4960  
4970  
4980  
4990  
5000  
5010  
5020  
5030  
5040  
5050  
5060  
5070  
5080  
5090  
5100  
5110  
5120  
5130  
5140  
5150  
5160  
5170  
5180  
5190  
5200  
5210  
5220  
5230  
5240  
5250  
5260  
5270  
5280  
5290  
5300  
5310  
5320  
5330  
5340  
5350  
5360  
5370  
5380  
5390  
5400  
5410  
5420  
5430  
5440  
5450  
5460  
5470  
5480  
5490  
5500  
5510  
5520  
5530  
5540  
5550  
5560  
5570  
5580  
5590  
5600  
5610  
5620  
5630  
5640  
5650  
5660  
5670  
5680  
5690  
5700  
5710  
5720  
5730  
5740  
5750  
5760  
5770  
5780  
5790  
5800  
5810  
5820  
5830  
5840  
5850  
5860  
5870  
5880  
5890  
5900  
5910  
5920  
5930  
5940  
5950  
5960  
5970  
5980  
5990  
6000  
6010  
6020  
6030  
6040  
6050  
6060  
6070  
6080  
6090  
6100  
6110  
6120  
6130  
6140  
6150  
6160  
6170  
6180  
6190  
6200  
6210  
6220  
6230  
6240  
6250  
6260  
6270  
6280  
6290  
6300  
6310  
6320  
6330  
6340  
6350  
6360  
6370  
6380  
6390  
6400  
6410  
6420  
6430  
6440  
6450  
6460  
6470  
6480  
6490  
6500  
6510  
6520  
6530  
6540  
6550  
6560  
6570  
6580  
6590  
6600  
6610  
6620  
6630  
6640  
6650  
6660  
6670  
6680  
6690  
6700  
6710  
6720  
6730  
6740  
6750  
6760  
6770  
6780  
6790  
6800  
6810  
6820  
6830  
6840  
6850  
6860  
6870  
6880  
6890  
6900  
6910  
6920  
6930  
6940  
6950  
6960  
6970  
6980  
6990  
7000  
7010  
7020  
7030  
7040  
7050  
7060  
7070  
7080  
7090  
7100  
7110  
7120  
7130  
7140  
7150  
7160  
7170  
7180  
7190  
7200  
7210  
7220  
7230  
7240  
7250  
7260  
7270  
7280  
7290  
7300  
7310  
7320  
7330  
7340  
7350  
7360  
7370  
7380  
7390  
7400  
7410  
7420  
7430  
7440  
7450  
7460  
7470  
7480  
7490  
7500  
7510  
7520  
7530  
7540  
7550  
7560  
7570  
7580  
7590  
7600  
7610  
7620  
7630  
7640  
7650  
7660  
7670  
7680  
7690  
7700  
7710  
7720  
7730  
7740  
7750  
7760  
7770  
7780  
7790  
7800  
7810  
7820  
7830  
7840  
7850  
7860  
7870  
7880  
7890  
7900  
7910  
7920  
7930  
7940  
7950  
7960  
7970  
7980  
7990  
8000  
8010  
8020  
8030  
8040  
8050  
8060  
8070  
8080  
8090  
8100  
8110  
8120  
8130  
8140  
8150  
8160  
8170  
8180  
8190  
8200  
8210  
8220  
8230  
8240  
8250  
8260  
8270  
8280  
8290  
8300  
8310  
8320  
8330  
8340  
8350  
8360  
8370  
8380  
8390  
8400  
8410  
8420  
8430  
8440  
8450  
8460  
8470  
8480  
8490  
8500  
8510  
8520  
8530  
8540  
8550  
8560  
8570  
8580  
8590  
8600  
8610  
8620  
8630  
8640  
8650  
8660  
8670  
8680  
8690  
8700  
8710  
8720  
8730  
8740  
8750  
8760  
8770  
8780  
8790  
8800  
8810  
8820  
8830  
8840  
8850  
8860  
8870  
8880  
8890  
8900  
8910  
8920  
8930  
8940  
8950  
8960  
8970  
8980  
8990  
9000  
9010  
9020  
9030  
9040  
9050  
9060  
9070  
9080  
9090  
9100  
9110  
9120  
9130  
9140  
9150  
9160  
9170  
9180  
9190  
9200  
9210  
9220  
9230  
9240  
9250  
9260  
9270  
9280  
9290  
9300  
9310  
9320  
9330  
9340  
9350  
9360  
9370  
9380  
9390  
9400  
9410  
9420  
9430  
9440  
9450  
9460  
9470  
9480  
9490  
9500  
9510  
9520  
9530  
9540  
9550  
9560  
9570  
9580  
9590  
9600  
9610  
9620  
9630  
9640  
9650  
9660  
9670  
9680  
9690  
9700  
9710  
9720  
9730  
9740  
9750  
9760  
9770  
9780  
9790  
9800  
9810  
9820  
9830  
9840  
9850  
9860  
9870  
9880  
9890  
9900  
9910  
9920  
9930  
9940  
9950  
9960  
9970  
9980  
9990  
10000  
10010  
10020  
10030  
10040  
10050  
10060  
10070  
10080  
10090  
10100  
10110  
10120  
10130  
10140  
10150  
10160  
10170  
10180  
10190  
10200  
10210  
10220  
10230  
10240  
10250  
10260  
10270  
10280  
10290  
10300  
10310  
10320  
10330  
10340  
10350  
10360  
10370  
10380  
10390  
10400  
10410  
10420  
10430  
10440  
10450  
10460  
10470  
10480  
10490  
10500  
10510  
10520  
10530  
10540  
10550  
10560  
10570  
10580  
10590  
10600  
10610  
10620  
10630  
10640  
10650  
10660  
10670  
10680  
10690  
10700  
10710  
10720  
10730  
10740  
10750  
10760  
10770  
10780  
10790  
10800  
10810  
10820  
10830  
10840  
10850  
10860  
10870  
10880  
10890  
10900  
10910  
10920  
10930  
10940  
10950  
10960  
10970  
10980  
10990  
11000  
11010  
11020  
11030  
11040  
11050  
11060  
11070  
11080  
11090  
11100  
11110  
11120  
11130  
11140  
11150  
11160  
11170  
11180  
11190  
11200  
11210  
11220  
11230  
11240  
11250  
11260  
11270  
11280  
11290  
11300  
11310  
11320  
11330  
11340  
11350  
11360  
11370  
11380  
11390  
11400  
11410  
11420  
11430  
11440  
11450  
11460  
11470  
11480  
11490  
11500  
11510  
11520  
11530  
11540  
11550  
11560  
11570  
11580  
11590  
11600  
11610  
11620  
11630  
11640  
11650  
11660  
11670  
11680  
11690  
11700  
11710  
11720  
11730  
11740  
11750  
11760  
11770  
11780  
11790  
11800  
11810  
11820  
11830  
11840  
11850  
11860  
11870  
11880  
11890  
11900  
11910  
11920  
11930  
11940  
11950  
11960  
11970  
11980  
11990  
12000  
12010  
12020  
12030  
12040  
12050  
12060  
12070  
12080  
12090  
12100  
12110  
12120  
12130  
12140  
12150  
12160  
12170  
12180  
12190  
12200  
12210  
12220  
12230  
12240  
12250  
12260  
12270  
12280  
12290  
12300  
12310  
12320  
12330  
12340  
12350  
12360  
12370  
12380  
12390  
12400  
12410  
12420  
12430  
12440  
12450  
12460  
12470  
12480  
12490  
12500  
12510  
12520  
12530  
12540  
12550  
12560  
12570  
12580  
12590  
12600  
12610  
12620  
12630  
12640  
12650  
12660  
12670  
12680  
12690  
12700  
12710  
12720  
12730  
12740  
12750  
12760  
12770  
12780  
12790  
12800  
12810  
12820  
12830  
12840  
12850  
12860  
12870  
12880  
12890  
12900  
12910  
12920  
12930  
12940  
12950  
12960  
12970  
12980  
12990  
13000  
13010  
13020  
13030  
13040  
13050  
13060  
13070  
13080  
13090  
13100  
13110  
13120  
13130  
13140  
13150  
13160  
13170  
13180  
13190  
13200  
13210  
13220  
13230  
13240  
13250  
13260  
13270  
13280  
13290  
13300  
13310  
13320  
13330  
13340  
13350  
13360  
13370  
13380  
13390  
13400  
13410  
13420  
13430  
13440  
13450  
13460  
13470  
13480  
13490  
13500  
13510  
13520  
13530  
13540  
13550  
13560  
13570  
13580  
13590  
13600  
13610  
13620  
13630  
13640  
13650  
13660  
13670  
13680  
13690  
13700  
13710  
13720  
13730  
13740  
13750  
13760  
13770  
13780  
13790  
13800  
13810  
13820  
13830  
13840  
13850  
13860  
13870  
13880  
13890  
13900  
13910  
13920  
13930  
13940  
13950  
13960  
13970  
13980  
13990  
14000  
14010  
14020  
14030  
14040  
14050  
14060  
14070  
14080  
14090  
14100  
14110  
14120  
14130  
14140  
14150  
14160  
14170  
14180  
14190  
14200  
14210  
14220  
14230  
14240  
14250  
14260  
14270  
14280  
14290  
14300  
14310  
14320  
14330  
14340  
14350  
14360  
14370  
14380  
14390  
14400  
14410  
14420  
14430  
14440  
14450  
14460  
14470  
14480  
14490  
14500  
14510  
14520  
14530  
14540  
14550  
14560  
14570  
14580  
14590  
14600  
14610  
14620  
14630  
14640  
14650  
14660  
14670  
14680  
14690  
14700  
14710  
14720  
14730  
14740  
14750  
14760  
14770  
14780  
14790  
14800  
14810  
14820  
14830  
14840  
14850  
14860  
14870  
14880  
14890  
14900  
14910  
14920  
14930  
14940  
14950  
14960  
14970  
14980  
14990  
15000  
15010  
15020  
15030  
15040  
15050  
15060  
15070  
15080  
15090  
15100  
15110  
15120  
15130  
15140  
15150  
15160  
15170  
15180  
15190  
15200  
15210  
15220  
15230  
15240  
15250  
15260  
15270  
15280  
15290  
15300  
15310  
15320  
15330  
15340  
15350  
15360  
15370  
15380  
15390  
15400  
15410  
15420  
15430  
15440  
15450  
15460  
15470  
15480  
15490  
15500  
15510  
15520  
15530  
15540  
15550  
15560  
15570  
15580  
15590  
15600  
15610  
15620  
15630  
15640  
15650  
15660  
15670  
15680  
15690  
15700  
15710  
15720  
15730  
15740  
15750  
15760  
15770  
15780  
15790  
15800  
15810  
15820  
15830  
15840  
15850  
15860  
15870  
15880  
15890  
15900  
15910  
15920  
15930  
15940  
15950  
15960  
15970  
15980  
15990  
16000  
16010  
16020  
16030  
16040  
16050  
16060  
16070  
16080  
16090  
16100  
16110  
16120  
16130  
16140  
16150  
16160  
16170  
16180  
16190  
16200  
16210  
16220  
16230  
16240  
16250  
16260  
16270  
16280  
16290  
16300  
16310  
16320  
16330  
16340  
16350  
16360  
16370  
16380  
16390  
16400  
16410  
16420  
16430  
16440  
16450  
16460  
16470  
16480  
16490  
16500  
16510  
16520  
16530  
16540  
16550  
16560  
16570  
16580  
16590  
16600  
16610  
16620  
16630  
16640  
16650  
16660  
16670  
16680  
16690  
16700  
16710  
16720  
16730  
16740  
16750  
16760  
16770  
16780  
16790  
16800  
16810  
16820  
16830  
16840  
16850  
16860  
16870  
16880  
16890  
16900  
16910  
16920  
16930  
16940  
16950  
16960  
16970  
16980  
16990  
17000  
17010  
17020  
17030  
17040  
17050  
17060  
17070  
17080  
17090  
17100  
17110  
17120  
17130  
17140  
17150  
17160  
17170  
17180  
17190  
17200  
17210  
17220  
17230  
17240  
17250  
17260  
17270  
17280  
17290  
17300  
17310  
17320  
17330  
17340  
17350  
17360  
17370  
17380  
17390  
17400  
17410  
17420  
17430  
17440  
17450  
17460  
17470  
17480  
17490  
17500  
17510  
17520  
17530  
17540  
17550  
17560  
17570  
17580  
17590  
17600  
17610  
17620  
17630  
17640  
17650  
17660  
17670  
17680  
17690  
17700  
17710  
17720  
17730  
17740  
17750  
17760  
17770  
17780  
17790  
17800  
17810  
17820  
17830  
17840  
17850  
17860  
17870  
17880  
17890  
17900  
17910  
17920  
17930  
17940  
17950  
17960  
17970  
17980  
17990  
18000  
18010  
18020  
18030  
18040  
18050  
18060  
18070  
18080  
18090  
18100  
18110  
18120  
18130  
18140  
18150  
18160  
18170  
18180  
18190  
18200  
18210  
18220  
18230  
18240  
18250  
18260  
18270  
18280  
18290  
18300  
18310  
18320  
18330  
18340  
18350  
18360  
18370  
18380  
18390  
18400  
18410  
18420  
18430  
18440  
18450  
18460  
18470  
18480  
18490  
18500  
18510  
18520  
18530  
18540  
18550  
18560  
18570  
18580  
18590  
18600  
18610  
18620  
18630  
18640  
18650  
18660  
18670  
18680  
18690  
18700  
18710  
18720  
18730  
18740  
18750  
18760  
18770  
18780  
18790  
18800  
18810  
18820  
18830  
18840  
18850  
18860  
18870  
18880  
18890  
18900  
18910  
18920  
18930  
18940  
18950  
18960  
18970  
18980  
18990  
19000  
19010  
19020  
19030  
19040  
19050  
19060  
19070  
19080  
19090  
19100  
19110  
19120  
19130  
19140  
19150  
19160  
19170  
19180  
19190  
19200  
19210  

```



## 1. はじめに

TTL (Tiny Tiny Language) は基本的には整数型BASICを記号化してインタプリタ作成の負担を軽くしたものです。このような整数型の言語の生い立ちは、メモリが非常に高価であったところにさかのぼります。峰岸氏の連載でも書かれていますが、当時のマイコンマニアたちの夢は何よりもまず、自分の作った（そのころはまだ、マイコンの完成品などは売っていませんでした）マイコンで高級言語を走らせることでした。大学の研究所などで、お金を湯水のごとく注ぎ込めるような恵まれた環境の人を別にすると、当時のマニアに許されるメモリ容量はどんなに頑張ったとしても数Kバイトに過ぎませんでした。

この程度の容量ではもちろん、浮動小数点演算などを行うわけにはいきません。そこで生まれてきたのが、扱う数値を整数に限定し、命令も基本的なもの以外を取り払った、いわゆるTINY言語だったのです。

ダンプリストをもとに解説し、自分のマイコン用に書き換え、メモリが足りない自分で機能を削ったり、1バイトでも短くできるところはないかと必死の思いで探したりして、みな、自分のコンピュータで動かす喜びに浸っていたものです。

メモリも安価になり、マイコンが商売になるとみた大手メーカーが台頭し、実数型のBASICインタプリタ+32K以上のRAMの搭載が当たり前になるにつれてこのような言語を作っていたマニアたちもしだいに姿を消していき、TINY-BASICなどというものがこの世に存在していたことさえ忘れられてしまいました。メモリも安価になり、本格的な言語が動くのが当たり前になった現在、なにも時代の流れに逆らって整数型の言語を作成する意味があるのか疑問視する向きもあると思いますが、果たしてそうでしょうか。私個人の考えかもしれませんが、少なくともテープベースのクリーンコンピュータでのTINY言語にはTINYなりの良さがあると思うのです。

BASICでは数値の演算は通常、実数演算を行いますので、整数のみの演算に使う場合には無駄が多くなります。実数演算は学術計算には絶対に必要なものですが、ゲームやパズルを解く場合などでは使用頻度は非常に低いものでしょう。また、自分で作成したハードウェアのテストなどを行う場合にはメモリやI/Oの操作能力が問題となりますが、多くのBASICではPEEK関数、INP関数、POKE命令、OUT命令によって1バイトの受け渡しが許されているのみで、非常に貧弱なものです。また、Z80のようにメインメモリの空間の狭いCPUを使用する場合にはメモリ効率も問題となり、無用な実数演算ルーチンのためにメモリをくわれるのはあまり望ましいものではありません。

ここで気を付けて欲しいのは、このようなことから「だからBASICは駄目だ」と判断しないようにということです。BASICとTTLでFOR~NEXTループのようなものを組み、「TTLのほうが2倍ぐらい速い」と単純に考えるのは大きな間違いであるということです。たとえば、「そのループの中で、浮動小数点演算を行ってみなさい」と言われたらどうですか。BASICではあっさり追加できますが、TTLでは考え込んでしまいます。プログラムの完成までの時間はまさに桁違いです。同じことが機械語についてもいえます。I/Oやメモリの操作を機械語で置き換えるとBASICとは比べものにならないぐらいの動き方をすると、マイコンマニアが触れるプログラムのほとんどがリアルタイムのゲームであるということも手伝って、「機械語は速い」とか「機械語で組むのが上級者だ」、「機械語のプログラムは高級だ」といった間違いをしがちなのでしょう。少なくともこの記事をお読みの皆さんはこのような間違いがないようにしてください。

## 2. 入力について

インタプリタのサイズはワークエリアを含めて8Kバイトとなっており、その3分の2程度が演算用の第1スタックとサブルーチンコールやループ制御用の第2スタック用のエリアとして使われていますのでインタプリタ本体は約3Kバイトです。したがって、GOSUBやREPEATループのネスティングレベルをそれほど深くしないのであれば、4Kバイトでも動作は可能です。

私が最初に作成したものは8Kバージョンですが、アセンブラのORGを書き換えてC000hからアセンブルし直せば、4Kバージョンにすることもできます（この他のアドレスに配置するのはソースリストを入力した後、ORGや、ワークエリアなどを設定し直してから再アセンブルすることによって可能です）。

入力はモニターから、ダンプリストどおりにB000h~BBFFhまで打ち込みます。終わったら、スタートアドレスB000h、エンドアドレスBBFFh、ジャンプアドレスB002hとして、テープにセーブしておきましょう。TTLのコールドスタートはB002h番地、ホットスタートはB000h番地になっています。いずれの場合も、テキストエリアのクリアは行いませんので、プログラムを入力する前には%=0 (BASICのNEWに相当する命令) を実行しておく必要があります。

セーブが終わったら、試しに動かしてみましょう。8KバージョンではB002hにジャンプさせます。画面がクリアされて画面の一番上から

★★ TTL VERSION 1.1(C)MIKA ★★

\*READY



のようにタイトルが出力されて、入力待ちになります。これがTTLがコールドスタートした状態です。続いて%=0によってNEWを行ってからテキストの入力を行います。

```
1000 "DUMP FROM ? " ADRS=?
1010 ??=A " : "
1020 I=0 , #=8
1030 ?#=<A:I> " "
1040 +I @=I
1050 / ; !=0 A=A+B #=1010
1060 #=-1
```

入力が終わったら0 [CR] によっていったんリストをとってみます。今、入力したテキストがそのまま出力されます。間違いがなければ走らせてみましょう。実行は#=1で行います。

“DUMP FROM?”と聞いてきますので、\$B000を入力しますと、TTLインタプリタの機械語リストが出力され、何かキーを押すと終了します。きちんと出力されていれば、正常に動いたと考えて良いでしょう。あとでいくつかサンプルを載せますので動かしてみてください。

## リファレンス

### 1. TTLの特徴

TTLはBASICよりもメモリやI/Oの操作が強力であり、フリーエリアも8Kバージョンでも40Kバイトと非常に広がっているために、これまで機械語に頼っていたような仕事をTTLで行わせることもある程度可能となっています。特にメモリ操作においては操作が2バイト単位で行えるために、BASICに比べて非常に簡潔な表現ができます。ひとつ例を挙げましょう。次のプログラムはV、およびV+1番地に書かれているデータを2バイトの符号なしのデータと見なして取り込み、200を加えたあと元のアドレスに戻すものです。



```
BASIC:
1000 INPUT "ADDRESS=";V
1010 A=PEEK(V+1)*256+PEEK(V)
1020 A=A+200
1030 POKE V+1,INT(A/256)
1040 POKE V,A MOD 256
```

```
TTL (1):
1000 "ADDRESS="V=?
1010 A=<V:1>#256+<V:0>
1020 A=A+200
1030 <V:1>=A/256
1040 <V:0>=#
```

```
TTL (2):
1000 "ADDRESS="V=?
1010 <V:0>=<V:0>+200
```

TTL(1)はBASICに忠実に書いてみたもので、TTL(2)はTTLの2バイト型メモリアクセス機能をフルに使用してみたものです。最初のBASICで書いたものに比べ、非常に簡潔になっているのがわかります。これはかなり極端な例ですが、いずれにしても書き込みに2バイト型が使用できることはかなりの効果があると思います。

TTLの命令セットはHuBASICはおろか、S-BASICと比べても貧弱なものではありますが、用途によってはこのようにTTLがBASICを圧倒する場合もあるのです（逆に多次元配列や文字列処理などではTTLがBASICに圧倒されてしまいます）。TTLは機械語に非常に近いBASICとも言えるでしょう。このため、TTLを使ってシステムプログラムを書くことも可能です。実際、TTLのアルゴリズムはベースとなったGAMEインタプリタを使って開発したのですから、言ってみればTTLはTTL自身によって開発されたということになるでしょう。

## 2. サンプルプログラム

TTLを使ってサンプルプログラムを作ってみました。TTLによるプログラミングの参考にしてください。なお、プログラム中↑となっているものは、MZ-80B/2000/2200, X1ユーザーの方は↑と書き換えてください。

### (1) Oh! MZ式チェックサム

```
900 ,=1 @=#! "PRINTER ON ? (Y/N)"/
1000 ,=0 ,=1 N=! @=N ;=N="Y" ,=1
1010 "START ADRS = " S=?
1020 " END ADRS = " E=?
1030 "## OH! MZ CHECK SUM SYSTEM V1.2 ##"/
1040 "ADR. +0 +1 +2 +3 +4 +5 +6 +7 :SUM. "/
1050 A=0 B=0 K=0 ,=1
1060 PAGESUM=0
1070 J=0 ,=B
1080 ??=S " " LINESUM=0
1090 I=0 ,=B
1100 DATA=<S:I> ?#=D " " L=L+D
1110 +I @=I
1120 " : " ?(1)=#L.#F " " ?#=L /
1130 P=P+L S=S+B
1140 +J @=J
1150 "-----"
1160 ??=P / ;=A>(A+P) +B
1170 A=A+P
1180 ;=! ,=1 @=#!
1190 @=(S>E)
1200 ??=B ??=A /
1210 .=0
```

### (2) 再帰的関数表現

TTLの特徴のひとつである、再帰的な関数の表現のサンプルを作ってみました。再帰を使うと、漸化式をそのまま使用することができるようになります。

うになります。サンプルは、ある自然数Nを与えたときに1からNまでの自然数の和を返すものです。たとえばNが2ならば3, Nが5ならば15を返してきます。これを漸化式を使って書くと次のようになります。

$$y = f(x) = \begin{cases} 1 & \text{if } x = 1 \\ f(x-1) + x & \text{if } x > 1 \end{cases}$$

これをTTLで記述すると次のようになります。

```
1000 "A=" A=?
1010 :=2000,A
1020 ?=Z
1030 #=-1
1999-----
2000 ;=A=1 Z=1 ↑
2010 :=2000,A-1 Z=A+Z ↑
```

2000行が漸化式で言う  $f(x) = 1$  の部分に相当し、次の2010行が  $f(x-1) + x$  に相当します。TTLではPASCALのように関数が値を返す機能がないために変数Zを使って値の受け渡しを行っています。つまり、2000行を :=2000, A という形で呼び出すと1からAまでの自然数の和を変数Zに代入してくるわけです。

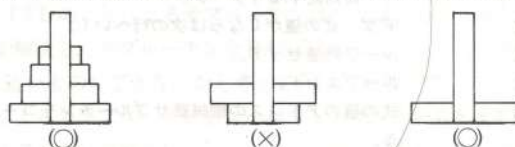
### (3) ハノイの塔

ハノイの塔というのはかなり古典的なパズルのひとつですが知らない方もいらっしゃると思いますので、簡単に説明しておきましょう。

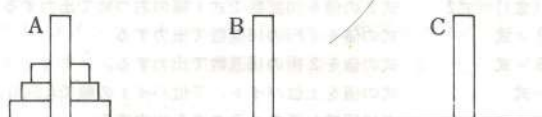
A, B, Cと3つの柱があり、柱Aに大きい円板から順次小さい円板が重なっています。この円板を次の規則に従って円板全部をAからCに動かすのを目的とします。

①円板は1度には1枚しか動かさせません。

②大きい円板を小さい円板の上に重ねることは許されません。



例として円板が3つの場合の回答を示しましょう。



例ではAに3枚のプレートが重ねられています。これをCに移すには A→C A→B C→B A→C B→A B→C A→C となります。わかりにくい人は紙を切って重ねて動かしてみるとよいでしょう。

```
1000----- TOWER OF HANOI -----
1010 "HOW MANY PLATES ? " N=?
1020 :=2000,1,2,3,N
1030 /"FINISH!"/
1040 #=-1
1999-----
2000 ;=D<2 #=2100
2010 :=2000,A,C,B,D-1
2020 $=#40+A "->" $=#40+C " "
2030 :=2000,B,A,C,D-1
2040 ↑
2100 $=#40+A "->" $=#40+C " "
2110 ↑
```

TTLではローカル変数を6つまで持つことが可能ですから、パラメータを6つ持ち、多数のデータを返すような関数まで記述することが可能となっています。ここまで来ると言語を作った私自身にも、どのように使ったものか、考え込んでしまいます。みなさんのアイデアで使いこなしてください。



### 3. 文法について

TTLの設計にあたり、GAMEとの互換性がある程度意識していましたが、GAMEをご存じの方にはすぐにはみこめるとは思いますが、さすがに二十数回にわたってバージョン変更を行っている、自分自身にも記憶しかねる部分が出て来てしまったので、リファレンスマニュアルを作ってみました。もともと個人用に使うつもりだったのですが、考えてみると、TTLを使うのは私だけではなくるのですから、やはり公開することにしました。一般に公開するということで、かなり書きかたはみただけですが、もともと自分用に作ったものですからややわかりにくい箇所があるかもしれません。各コマンドの詳細については次の章の解説を参照してください。

ここで項というのは定数、変数、およびカッコでくくられた式のことを指し、式というのは少なくとも1つ以上の項からなり、項が2つ以上の場合には項と項の間に演算子が入ります。

(項) A, 1234, \$ABCD, !, "ABC", (123+4\*6), (2\*(3+2)), etc.

(式) A, 12345, A+B, 12+(3\*3), !\*2, etc.

#### <コマンド一覧>

#=式	式で指示される行番号へジャンプする。式の値が32768以上であればプログラムの実行を終了する(BASICのENDと同じ)。
!=式	式で指示される行番号のサブルーチンをコールする。サブルーチンからのリターン。
] =式	値の引渡しと変数の退避を行うサブルーチンコール。
↑ (^)	:=に対応するリターン。
; =式	IF文。式の値が0ならば次の行へいく。
: =式	ループ終値セット。
@ =式	ループエンド。
> =式	式の値のアドレスの機械語サブルーチンをコールする。
? =式	式の値を10進数の5桁の右づめで出力する。
? (式) =式2	式2の値を10進数で式1幅の右づめで出力する。
? ? =式	式の値を4桁の16進数で出力する。
? \$ =式	式の値を2桁の16進数で出力する。
\$ =式	式の値を上位バイト、下位バイトの順でASCIIコードに変換してキャラクタを出力する。
"文字列"	文字列を出力する。
/	改行する。
'数字列'	各数字に対応するディスプレイコントロールを順に実行する。
+ 変数	変数の値を1増やす。
- 変数	変数の値を1減ずる。
* 変数	変数の値の上位バイトと下位バイトを交換する。
10進定数 /	定数で指示される行番号の行からプログラムリストを出力する。ダイレクトモード時のみ有効(プログラム中では使用不可能)。

#### <変数・定数>

A-Z	単純変数。冗長形が許される。
¥	最後に行った除算の余りを返す。
<項: (式)	メモリ変数 (1バイト)
<項 (式)	メモリ変数 (2バイト)
[項: (式)	I/O変数 (1バイト)
[項 (式)	I/O変数 (2バイト)
"文字列"	文字列の右から2文字をASCIIコードと見てデータにする(ex. "ABC"→4243 <sub>16</sub> )。
!	キーボードを見て押されていなければ0、押されていれば該当するキーのASCIIコードを返す。

?	キーボードから式を1行入力し、その値をとる。
π	カセットからファイルを読む時の先頭アドレスを格納する(ただしπが0ならばセーブ時と同じアドレスから読み込まれる)。
(ピリオド)	出力制御用変数。プリンタの出力のON-OFF(第0ビットが1ならON)や画面コントロールコードの有効、無効(第1ビットが1なら無効)、および画面出力のON-OFF(第2ビットが1ならOFF)を決定する。
&	テキスト開始番地を格納する。
%	テキストの終了番地を格納する。プログラムの実行が終了し、"*READY"が出力された時点で自動的に文末探索が行われ、変数%の値も変更される。ただし、%=0はBASICのNEWに相当する。

※10進定数は通常使用するとおり(ex. 1234)。取り得る値は0から65535までの整数。負の数は補数表現であり、-1と65535が等しくなるため、たとえば-1>2は真となる。

※16進定数は最初の文字を\$にする(ex. \$12AF)。取り得る値は\$0000から\$FFFFである。

#### <二項演算子>

+, -, *	加算, 減算, 乗算
/	除算(余りは自動的に変数¥に格納される)
(ピリオド)	論理積(AND)
;	論理和(OR)
!	排他的論理和(XOR)
>, <, =, #	大小比較(#……等しくない)

#### <単項演算子>

#	項の値が0ならば1、それ以外の場合は0をかえす。
/	項の値の行番号の行の先頭のアドレスをあたえる。
-	0から項の値を引いた値にする。
*	項の値の上位バイトと下位バイトを交換する。

※行番号の直後にスペースをあげない場合、その行は注釈文と見なされ、読み飛ばされます。

※プログラムの入力方法は基本的にはBASICとほとんど変わりありません。RENUMやDELETEのようなコマンドもありませんが、やや不自由に感じることがあるかも知れませんが何せTINY言語ですので我慢してください。

※プログラムのSAVEルーチンはBB00<sub>h</sub>から、LOADルーチンはBB70<sub>h</sub>から設けてあります。>=\$BB00や>=\$BB70のようにしてCALLするとSAVE、LOADを行うことができます。

## マニュアル

次にTTLの文法をコマンド1つひとつについて説明していきましょう。なお、|はそれが省略可能であることを示しています。

### 1. コマンド

#### (1) # = 式

式で指定される行番号にジャンプします。BASICのGOTO文と同様に考えることができます。飛び先の指定に式が許されていますので、たとえば#=Aのようなことも許可されますが、プログラムが見づらくなりますから、あまり使用しないほうがよいでしょう。該当する行番号がない場合には式の値を越える最小の行番号にジャンプします。

```
1000 #=2000
1999 "L1999"
2001 "L2001"
```



1000行で2000行へのジャンプを指定していますが、プログラム中には2000行は存在しません。このプログラムを実行すると“L2001”と表示されます。BASICで同様のことを行うと“UNDEFINEND LINE NUMBER”などのエラーが出ます。TTLを作るとき、エラーを出すか否か迷ったのですが、0で除算を行ってしまったといったような場合と異なり、プログラムの実行が不可能になるようなエラーではない場合が多いので、エラーは出さないようにしました。

プログラム中で使用されている最大の行番号よりも大きい値を指定すると、プログラムの実行は打ち切れ、最後に実行した行を表示してコマンド待ちに戻ります。行番号として許されるのは1から\$7FFFまで、つまり32767までですから、それ以上の値を指定すると必ず実行が終了することになります。TTLで扱う数値は基本的には符号なしの2バイトの整数ですからたとえば1は65535と同じになります。したがって、 $\# = -1$ とするとBASICのENDと同等になります。

(2)  $! = \text{式1} \{ : \text{式2} \}$  および

$!$ は式1で表される行番号から始まるサブルーチンをコールし、サブルーチンからは]によって元に戻ります。BASICのGOSUB~RETURN文と同様です。該当する行がないときは $\# = 0$ のときと同様、式の値を越える最小の行をコールします。式の後にコロンの(:)をつけて式2を書くとその値から始まるテキスト中のサブルーチンをコールします。TTLではメモリの許す限り、複数のテキストをメモリ上に置くことが可能です。TTLをコールドスタートさせた直後にはテキストは\$7000からとなっていますが、後述する $\& =$ によって変更することが可能です。このことを頭にいて、次のようなプログラムを実行してみましょう。まず、 $\& = \$A000 \% = 0$  (CRをお忘れなく)とやって、\$A000番地からテキストを格納するようにしてから次のプログラム(サブルーチン)を入力します。

```
1000 --- SUB ROUTINE ---
2000 "THERE IS $A000 "
2010 J
```

次に $\& = 7000 \% = 0$ とやって、\$7000番地から次のプログラム(メインルーチン)を入力します。

```
1000 --- MAIN ROUTINE ---
1010  $! = 2000 : \$A000$ 
1020  $\# = -1$ 
2000 "THERE IS $7000"
2010 J
```

入力終了したら $\# = 1$ によって実行を開始させます。“THERE IS \$A000”と表示されたでしょう。\$A000から入力したプログラムの2000行からのサブルーチンがコールされたわけです。次に1010行の $! = \$A000$ を削除して再び実行してみます。今度は“THERE IS \$7000”が出力されたでしょう。これはBASICのGOSUB文と同じものとみなせるでしょう。TTLにはBASICのMERGEに相当するようなコマンドは存在しませんが、この例のようにプログラムを分割して互いに相手呼び出すことが可能なため、プログラムを各モジュールごとに開発して、最後にそれをまとめあげていくといったプログラム作成を行うことで、それほどの不自由はないと思うのですがいかがでしょうか。

(3)  $: = \text{式} \{ , \text{式} , \dots \} \{ : \text{式} \}$  および ↑  
(↑はMZ-80B/2200/2000, X1では^。以下同様に読み替えてください)

BASICにはTTLの $: =$ に相当する命令はありません。あえて類似を探せば、 $!$ を拡張したものということが可能かもしれません。

BASICを使用していて困ることのひとつにサブルーチン内でのみ有効なローカル変数がないということがあげられると思います。次のような

サブルーチンを組んだとしましょう。

```
1000 REM --- CALC SUM(A) ---
1010 S=0
1020 IF A<0 THEN 1060
1030 FOR B=1 TO A
1040 S=S+B
1050 NEXT B
1060 RETURN
```

このサブルーチンは1からAまでの自然数の和を求めるものです。このサブルーチンが正常に動作することは疑いがないところでしょう。ところが、このサブルーチンを次のように使うと問題が発生します。

```
100 REM --- MAIN ROUTINE ---
110 FOR B=1 TO 10
120 FOR C=0 TO 9
130 A=B*10+C
140 GOSUB 1000
150 PRINT A:S
160 NEXT C
170 PRINT
180 NEXT B
```

この例のようにメインルーチンでサブルーチンと同じ変数を使用していると、アルゴリズムとしては正しいにもかかわらず、プログラムとしては正常に動作しないという事態が発生します。ここであげた例はごく短いプログラムなのでかなりわざとらしい感じがするかもしれませんが、少し長いプログラムを組んだことがある方でしたら経験のあるところでしょう。TTLの $: =$ によるサブルーチンコールはこのようなトラブルをなんとか解決し、サブルーチンをPASCALで言うところのPROCEDUREに近似的にできないかと考えて作った命令です。

$: =$ によってサブルーチンをコールするとき、AからFまでの6つの変数はスタックに退避され、↑によってリターンするときに復帰されます。したがって、AからFまでの変数はサブルーチン内で値が変化してしまっても、メインルーチンに戻ったときに、サブルーチンを呼び出す前の値に戻りますから、お互いに変数の重複を気にすることなくプログラムを組むことができます。

さらに、 $: =$ 式の後に、式の形を続けて書いておくと(6つまでです)、変数の退避を行ったあと、式の値が変数AからFまで順に代入されていきます。次のプログラムを実行させるとよくわかるでしょう。

```
1000 A=1 B=2 C=3 Z=7
1010  $: = 2000, A+B$  "MAIN: "
1020 " A=" ?(1)=A " B=" ?(1)=B
1030 " C=" ?(1)=C " Z=" ?(1)=Z
1040 //  $\# = -1$ 
1999 -----
2000 B=5 Z=0 " SUB: "
2010 " A=" ?(1)=A " B=" ?(1)=B
2020 " C=" ?(1)=C " Z=" ?(1)=Z
2030 / ↑
```

少し込み入っていますが、順に見ていって下さい。1010行でサブルーチンコールを行い、AにA+B(=3)を代入しています。サブルーチン内ではBを5に、Zを0にしています。このPRINT文の結果からもわかるようにAは3( $: =$ で引き渡された値)になり、BとZは2000行で設定された値、Cはメインの最初で設定した値そのままになっています。↑によってメインに戻った時点ではAからCまでの変数は1010行の $: =$ を行う前と同じ値になっており、ZはBASICでGOSUBを行ったときと同様にサブルーチン内で変更されたままの値を持って来ていることが



わかります。

AからFまでの変数はこのように特殊な変数であり、サブルーチンの中のループカウンタなどにはこれらの変数のみを使うようにすることで、プログラムをブロックごとに開発して、動作が確認されたものについてはその内部で使用されている変数名などに気を使うことなく、ブラックボックス的に扱うことが可能になります。

さらに変数の退避や復帰をスタックで行い、ローカル変数への値の受け渡しが行えるために、簡単な再帰を使ったようなプログラムもTTLで扱うことができます(具体的な例はサンプルプログラムを見てください)。

この命令の場合にも! =と同様に、:に続いて式を書き加えることで、他のアドレスから始まるテキスト中のサブルーチンを呼び出すことが可能です。

#### (4); =式

式の値が0ならば次の行へ、それ以外の場合は次の文へいきます。BASICのIF文と同じものと考えられます。

#### (5), =式, @ =式

ループ制御です。 , =はループのリミットを設定するものです。ループの終わりは@ =です。もし, @ =の後の式の値が, =で設定した値以上になればループは終了し、それ未満であれば再びループを繰り返します。 , =1としておいて, @ =条件式とすれば、BASICのREPEAT ~ UNTIL文と同じことになります。

```
1000---- SAMPLE OF LOOP ----
1010 I=0 ,=100
1020 ?(4)=I
1030 +I @=I
```

#### (6)> =式

式の値で示されるアドレスから始まる機械語のサブルーチンをコールします。BASICのCALL文に相当します。

#### (7)? =式

式の値を10進数で5桁幅で右づめて出力します。

#### (8)? (式1) =式2

式2の値を10進数で式1の桁幅の右づめて出力します。

#### (9)? \$ =式

式の値を16進数に変換し、下位バイト(16進数2桁)を出力します。

#### (10)? ? =式

式の値を16進数の4桁で出力します。

#### (11)\$ =式

式の値を16進数に変換し、上位バイト、下位バイトをそれぞれASCIIコードとみなし、その順序に出力します。たとえばAが\$4142のときに? \$ = Aを実行すると"AB"と出力されます。

#### (12)"文字列"

文字列をそのまま出力します。BASICのPRINT"文字列";と同じです。文字列出力後の改行は行われませんので、改行が必要な場合は次に出てくる/による改行を付加します。

#### (13)/

改行する。後で説明する, . =によって画面コントロールを禁止しても、この命令による改行は実行されます。改行の禁止が可能になるようにしたい場合には, \$ = \$Dによって改行を行ってください。

#### (14)'数字列'

各数字に対応する画面コントロールを順に実行します。数字は続けて記述できます。各機種のコントロールコードが異なりますので、よく使うコントロールを1~6の数字に割り当てました。移植を考えると, \$ = 式や"コントロールキャラクタ"よりも'数字列'を使ったほうがよいと思

われます。コードとコントロールの対応は以下のとおりです。

- 1 .....カーソルを下へ移動
- 2 .....カーソルを上へ移動
- 3 .....カーソルを右へ移動
- 4 .....カーソルを左へ移動
- 5 .....カーソルをホームポジションへ移動
- 6 .....画面クリア

たとえば,'633311'を実行すると、画面クリアしたあと、カーソルをホームポジションから右へ3つ、下へ2つ移動させます。

#### (15)+変数

変数の値を1増やす。オーバーフローのチェックは行っていないので、変数が\$FFFF (=65535。ただし、変数が1バイト型の場合は\$FF =255)であるときに+変数を行うと変数の値は0になります。

#### (16)-変数

変数の値を1減ずる。+変数の場合と同様に、変数の値が0のときにこの命令を実行すると変数の値は\$FFFF (変数が1バイト型の場合には\$FF =255)になります。

#### (17)\*変数

変数の値を16進数とみなし、上位バイトと下位バイトを交換します。たとえばAが\$0102 (=256 + 2 =258)であるときに\*Aを実行するとAの値は\$0201 (=256 \* 2 + 1 =513)になります。1バイト型の変数(メモリやI/Oの1バイト型の指定型)の場合には上位4ビットと下位4ビットが交換されます。

#### (18)その他

※NEWは%=0, LISTは0 **CR**です。ある行からのLISTを出力したいときは行番号/によって行うことができます。この場合の行番号指定に使えるのは10進数のみで、16進数や変数、式などを使うことは許されていません。

※プログラムの実行は#=1によって行います。結局はGOTO文と同じですから変数のクリアは行われません。

## 2. 変数、定数

### (1)<項:式>, <項(式)> (メモリ変数)

TTLではメモリと普通の変数(AやBなど)とを区別せずに使用することができます。項と書いてあるところは、変数、定数、または(\$D000+40)のようにカッコでくくられた式のいずれかがきます。

メモリ変数には1バイト型と2バイト型があり、項と式の間が:で区切られているのが1バイト型、(で区切られているのが2バイト型です。1バイト型の場合にはメモリの(項+式)の値のアドレスが、2バイト型の場合には(項+2\*式)を下位バイト(項+2\*式+1)を上位バイトとしてアクセスされます。メモリ変数の内容を読み出すことはBASICのPEEKに相当し、メモリ変数に代入することはBASICのPOKEに相当します。たとえば,<\$D000:B>=AはBASICでいうとPOKE \$D000+B, Aに相当し、A=<\$D000(B)>はBASICで書くとA=PEEK(\$D000+2\*B)+256\*PEEK(\$D000+2\*B+1)になります。※2バイト型のメモリへのアクセスはアドレスの小さい方(下位バイト)、大きい方(上位バイト)の順序で行っています。タイミング的にまずい場合には1バイト型を2回使用して書き込むようにしてください。

### (2)[項:式], [項(式)] (I/O変数)

TTLではメモリと同時にI/Oを扱うことができ、メモリと同様に1バイト型と2バイト型をもちます。I/O操作はTTL内部ではCレジスタ間接モードで行っていますので、X1シリーズのようにI/O空間にV-RAMを割り付けたようなマシンの場合にも機械語のお世話にならずにI/O空間をアクセスすることが可能です。2バイト型の場合の読み書き



きの順序はアドレスの小さい方 (下位バイト), 大きい方 (上位バイト) の順序で行っています。

A=[B:0] は BASIC の A=INP(B), [B(1)]=A は BASIC の OUT B+2, A MOD 256:OUT B+3, INT(A/256) に相当します。

(3)&と% (テキスト開始番地, 終了番地)

TTLではテキスト(プログラム)はメモリ上のどこにあってもかまいませんので, テキストの開始番地は任意に決定することができます。テキストの開始番地を格納するのが変数&で, テキストの終わりを格納しているのが変数%です。??=&や, ??=%とすることで現在のテキストの開始番地やテキストの最後がどこになっているかを知ることができます。また, &=式としてテキストの開始番地を変更しながら, テキストを書くことで複数のプログラムを同時にメモリ上に存在させることができますから, BASICのようにいちいちカセットにセーブしてからNEWを行って再び入力するといった手間をかける必要もほとんどなくなります。

&と%はこのような用途のほか, セーブのときのセーブエリアの指定にも使用されます。>=\$BB00によってセーブルーチンをコールすると, &から%までのエリアがセーブされます。TTLでは"\*READY"が出力されるごとに自動的にプログラムエンド探索が行われ, %の値が更新されていますので, メモリの任意のエリアをセーブしたい場合には&=先頭アドレス %=エンドアドレス >=\$BB00と1行にズラズラとかいて一度に実行することで行います。

例外としては%=0があります。%=0はBASICのNEWに相当するものです。もっともNEWとは言っても, やっていることはテキストの先頭にFF<sub>h</sub>, 00<sub>h</sub>を書き込み%ポインタを&ポインタと等しくするだけです。たとえばA=先頭行番号 \*A <&(0)=Aとでもやれば簡単に復活することができますので, %=0をうっかり行ってしまったと言って嘆く必要はありません。

(4)π (ロードアドレス)

TTLでは一度作ったプログラムはメモリ上のどこに配置しても動作します。一度作ったプログラムをアドレスが合わないというだけの理由で再びキーインし直すというのは非常に馬鹿らしいことです。TTLでは一度セーブしておいたプログラムをロードするときに任意のアドレスから読み込むことができるようにしてあります。変数πがカセットからロードするときの先頭アドレスを指定するものです。例外として, πが0のときにはセーブしたときと同じアドレスにロードされますから, 通常はπ=0を実行してからロードするようにしましょう(ロードは>=\$BB70によって行います)。

(5)!

BASICで言うところのINKEY\$でしょうか。押されているキーがあればそのキーのASCIIコードを, 押されていないキーがなければ0を返します。2つ以上のキーが押されていても1つしか判別できません。また, 画面へのエコーバックも行われません。

(6)?

キーボードから1行分, 式を入力し, その値を返します。文字列に関してはダブルクォーテーションで囲むことで2文字(2バイト分)までは入力することができます。これ以上の文字列の入力が必要な場合には1行入力バッファ(LNBUF1)を利用してください。

(7)"文字列"

文字列の右から2文字をASCIIコードとみなし, 左側の文字を上位バイト, 右側を下位バイトとして値を返します。ちょうど\$=式の逆を行うと思えば良いでしょう。もし, 文字列が1文字しかないときは上位バイトは0に設定され, 文字のASCIIコードが下位バイトに設定されます。また, 文字列がヌルストリングのときは下位バイトも0になります。

### 3. 二項演算子

二項演算子は2×3のように次に書かれている項との間で行う演算を指定するものです。これに対して次の項の値のみに対して演算を行うのは単項演算子と言い, BASICでいうとLOG(X)とかINT(X), RND(X)などがそれにあたります。

(1)四則演算(+, -, \*, /)

それぞれ, 加算, 減算, 乗算, 除算です。いずれの場合にもオーバーフロー(0から65535の間にあるか否か)のチェックは行っていないので, たとえば?=\$100\*\$100をやると0に, ?=300\*400とすると54464といった具合に妙なことになります。

除算を行ったときの余りは変数%に代入されます。

(2)論理演算(. ; !)

それぞれ, AND, OR, XORです。TTLでは16ビット単位で処理される点が違うぐらいで機械語の論理演算と似たようなものです。ここで論理演算についてよくご存じでない方のために少し説明を加えておきましょう。たとえばA=\$4E, B=\$7Cとします。

①論理演算は2進数で行われます。

A=\$4E=01001110

B=\$7C=01111100

②両者の同じ位置にあるビットを比べ, 演算を行います。

ANDは, 両者のビットが共に1の場合にのみ1, その他の場合には0にします。ビット単位での掛算のようなものということで, 論理積とも言われます。AとBのANDをとると, 01001100=\$4Cとなります。

ORは, 少なくともどちらか一方のビットが1であれば1になり, どちらも0なら0になります。論理和ともいわれます。AとBのORをとると, 01111110=\$7Eとなります。

XORは両者のビットが同じであれば0に, 違っていれば1になります。一般に, 排他的論理和と呼ばれています。AとBのXORをとると, 00110010=\$32となります。

(3)比較(<, >, =, #)

大小関係を判別するもので, 関係式が成立すれば(真ならば)1, 成立しなければ(偽ならば)0となります。#=両者が等しくないという条件です。たとえば, 1=2の値は0, 1#2は1になります。比較は2バイトの符号なしの場合の値で行われますので, -1は65535とみなされます。したがって, -1>2は真となってしまいますから注意してください。

### 4. 単項演算子

単項演算子は次にくる項に対して演算を行うもので, 二項演算よりも優先的に処理されます。

(1)#

どちらかという, 算術演算よりも論理演算用の単項演算子で, 意味としてはNOTのつもりで作ってみました。次の項の値が0ならば1, それ以外ならば0になります。#に続けて論理演算式をカッコでくくって記述すると, 論理演算式の否定がえられます。;(BASICのIF文に相当)文で使うことが多くなるでしょう。

!=#(A=B) "NOT EQUAL!"

(AがBと等しくなければ"NOT EQUAL!"と出力する。)

(2)-

TTLは基本的には負の数は持っていませんが, これではV-RAMのアクセスのときなどにやや不便を感じるので, 単項演算子として用意することで不自然なく利用することができるようにしました。10進の定数が



きたときの形は-1といったふうになるので一見演算子でないようにみえますが、TTLの内部では演算子として扱われています。-はその前に0があるのと同じ値を返します（たとえば-1は65535、-2は65534といった具合になります）。

(3)/  
次の項で与えられる値の行番号を探し出してそのアドレスを返します。該当する行番号がないときには項の値を越える最小の行番号のあるアドレスになります（#=などの飛び先と同じことです）。返ってきたアドレスと次のアドレスに格納されている2バイトの値が行番号になっています。

```
1004 A=0 B=1 C=3
1010 W=/1000 Z=*W(0) ?=Z
```

（#=1000を行ったときの飛び先、この場合には1000行がないので1004行を見つけ、その行番号を表示します。）

(4)\*  
項の値を16進数4桁に直し、その上位2桁と下位2桁を交換した値にします。コマンドの\*と同じようなことを行う演算子ですがコマンドが単体で用いられるのに対し、演算子の\*は必ず式の中で使用される点が異なります。

```
1000 A=$1234
1010 ??=A / *A ??=A / ?=*A //
1020 B=5**A ?=B
```

（二項演算子の\*とは式の中で置かれる位置が異なるので判別できます。）

## おわりに

### 移植について

TTLに限らず、ほとんどのシステムソフトウェアは、使用するCPUさえ同じならば入出力ルーチンを書き換えるのみで動作することはCP/Mなどを使用している方ならばよくわかることでしょう。CP/M上で動作するプログラムは機種に依存せず、たとえばPCで動いているPASCALをそのままMZのメモリ上にもってくと（ディスクのフォーマットを変えたりするだけで）、そのまま何の手直しも行わずに動かすことができます。世の中にはパソコンと称するものが数多く出回り、それぞれがそれなりの特徴を持っています。このことは、ユーザー側には目的に合う物を選択できることにもなり、好ましいことではありますが、反面、あまり機種の特性に依存しないプログラム、たとえば、アセンブラなどのシステムソフトなどでは非常にムダな労力を払うことにもなります。

CPUが同じで、同じ機械語のプログラムが同じアドレスにあれば、同じことをやるのは当然ですが、外から見ているユーザー（オペレータ）にとっては同じことにはなりません。たとえばMZ-700でD000番地に1を書く画面の左上にAと表示されますが、PCで同じことを行ってみると、そのようなことはまったく起こりません。CPUにとっては同じ「D000番地に1を書く」ことであっても、システム全体では同じことにはならないことになります。

このようにどうしても機種に依存してしまうのはその機種独自の入出力部だけです。逆にいえば、機種に依存している入出力を書き換えれば、他はまったくいじらずに移植が完了することになります。CP/Mではこのような入出力関係のサブルーチンのエントリポイント（入口）や入力条件（パラメータの与え方）などが決められており、それを使うようにする限り、まったく機種に依存しないプログラムになります。

TTLを作るときにこのようなことも頭の片隅においていました。MZの場合、CP/MなどのようなグローバルなOSは標準で付いてこないのですが、幸いクリーン思想を正しく堅持してきたシャープさんがMZ-80KのP-ROMタイプ（試作段階）以来、最新のMZ-1500にいたるまでROMの主要サブルーチンのエントリポイントやワークエリアをまったく変更することなく、またカセットのフォーマットなどについても共通のまま、バージョンアップをしてきてくれたおかげで、ROMがMZ-80K/C/K2/K2E、1200、711/721/731、1500のすべてを標準化するキーになっています。ROM内のサブルーチンを使う限り、シャープの1号機でも最新の1500でもまったく同じプログラムが使えるということは驚異でもあります。某社のようにコンパチビリティをうたってこそいせんが、MZは統一された設計思想に基づいて作られた素晴らしいシリーズであるといえます。

話をTTLに戻しましょう。TTLはMZ-700（MZ-731）で作成したことはすでに述べましたが、このとき、700の特徴であったファンクションキーやカラーコントロールは無視し、MZ-80Kシリーズと共通なROM内ルーチンのみ使用しました。このため、TTLは一度カセットを作っておけば、MZ-80KからMZ-1500に至るまでのMZ-80K系統の機種すべてでロードし、実行することができます。もちろん、コマンドや動作にもまったく変化はありません。機種別の特徴は多少そこなわれることはいなめませんが、機種に依存しないことのメリットを考えれば当然ともいえるでしょう。

TTLを他機種に移植する場合は少し手間がかかります。幸い、編集室にX1とMZ-80B、2000に詳しい方がおられますので、MZ-80B、2000、およびX1への移植は完了していますが、その他の機種や自作の機械に移植したい方のために簡単に移植の方法を書いておきましょう。まず、TTLのソースリストをすべて入力し、入出力部の書き換えを行い、さらに必要があれば、ORGやワークエリアを変更します。書き換えが終了したら、アセンブルしなおしてオブジェクトプログラムを得ることになります。

ソースを入力するのが嫌な場合にはパッチをあてるという方法もあります。0番地からRAMになるモードにして、MZ-700のROMと同じエントリポイントから同じことを行うプログラムを組み込み（ワークエリアもそろえておきます）、TTLのオブジェクトはそのまま入力して使います。

どちらの方法でも動作するようになりますが、機械語のプログラムのソースというものは持っているとか何とか便利なものですから、MZ-80K系統以外の方はもちろん、MZ-80K系統の機械を使う場合でもできればソースを入力しておくほうが良いと思います。次にTTLで使ったROM内ルーチンとその動作を説明しておきましょう。

ラベル名	アドレス	動作
? ? CR (AF以外は保存)	0006	改行する。
? ? NL (AF以外は保存)	0009	同上。ただしカーソル位置が左端なら何もしない。
? ? PRNT (AF以外は保存)	0012	Accの内容を画面出力。
? ADCN (AF以外は保存)	0BB9	AccのASCIIコードをディスプレイコードに変換する。
? PRNTX (AF以外は保存)	0DB5	Accのディスプレイコードを出力。
? BRK (AF以外は保存)	001E	SHIFT+BREAKが押されているかを検出する。押されていればZフラグをたてる。
? GETKY (AF以外は保存)	001B	キーボードを読み、押されていればそのASCIIコードを、押されていなければ00をAccに収めてくる。エコーバックは行わない。



?GETL (全レジスタ保存)	0003	キーボードから1行入力を行う。入力した行はDレジスタで指定されるアドレスから格納される。行の終わりには0D(CRコード)がセットされる。入力文字数は0Dを含めて最大80文字。SHIFT+BREAKが押されたら先頭に0B(BREAKコード)次に0Dがセットされる。これらのコードは機種により異なることがあるので、ソース中では、INPEND、BRKEYというラベル名で定義してある。
DSPXY	1171	モニタのワークエリア。現在のカーソルのX座標が格納されている。INPUT文のときにキーから入力した部分のみをとりだすときに、?Nルーチン(??NLではない)の処理で使われる。

その他、システムに依存するのはプリンタへの出力ルーチンです(LPTと書いてあるところ)。これはソースをみると最後のところにおいてありますので、必要に応じて書き換えてください。仕様としてはAccに収められたASCIIコードをプリンタに出力するだけのものを組めばよいようにしてあります。

カセット関係のルーチンはうしろに付加しただけになっていますので(SAVEとLOADルーチン)、各自のシステムに応じてカセット、フロッピーディスク、QD用に書き換えても良いでしょう。

出力関係で、ASCIIコードを与えるだけで出力するルーチンがあるのに、ディスプレイコードへの変換を行うルーチン、ディスプレイコードを出力するルーチンを使っていることを不思議に思う方もおられるでしょう。ほとんどのキャラクタはどちらを使用しても同じ結果になるのですが、問題はコントロールコードです。MZではカーソルの移動や画面のクリアを行うキャラクタ(矢印やH、Cの反転文字)を持っています。プログラムの実行時にはコントロールコードとして動作して良いのですが、LISTをとるときにカーソルコントロールをやられたのでは使いものになりません。そこで、CALL ?ADCNに続いてCALL ?PRNTXを行うとコントロールキャラクタは反転文字で表示され、画面のコントロールは行われないのを利用してLIST出力などを行うのです。

したがって、テキストの中にコントロールコードが入らないシステムでは?ADCN ?PRNTXは不要で、?PRNTで書き換えて構いません。

SHIFT+BREAKは他の機種では単にBREAK、STOP、ABORTなどとよばれていると思いますが、いずれにせよプログラムの実行を打ち切るキーをセンスするようにしてください。

#### ●ちょっと注意していただきたいこと

ここではTTLと機械語を併用するときに必要な情報を書いておきましょう。

(1)変数(AからZまで)はワークエリアのVARTから2バイトずつ、A、B、C、D……Zの値を格納してあります。アドレスの小さいほうが下位バイトとなっていますので、例えば、LD HL, (VART)を実行すれば、HLレジスタに変数Aの値が転送されます。また、LD(VART+2), HLなどとすれば、TTLの変数(この場合は変数B)に値を引き渡すことができます。

(2)TTLの1行入力バッファは2つあります(LNBUF1のLNBUF2)。ひとつはテキストの入力のためのもの(LNBUF2)。もうひとつはA=?などを行うときの1行入力用のバッファです(LNBUF1)。最初、気付けずに両者を兼用していたら、ダイレクトモードでA=?をやったときにおかしなことになってしまいました(考えてみれば、あたりまえのことで、実行している行を壊してしまったのです)。ダイレクトモードで、1行入力を行うということをしないのであれば、LNBUF1とLNBUF2に同じ値を与えてもかまいません。

(3)TTLのスタックは3本あります(CPUのスタック、およびSTACK1とSTACK2)。TTLがスタートすると、CPUのスタックポインタはD000hに設定されます。この値はソース中ではCPUSTKで表わされています。TTLでは一部、サブルーチンの中やスタックに値を退避させた状態から強引にエラー処理に飛んだりしていますので、スタックのイニシャライズを行う所は5箇所もあります。ダンプを直接書き換える人はソースをよく見てください。

STACK1は演算式で、カッコの処理を行うときに使用されるもので、STACK2はループやサブルーチンコールの際に、ループの終値、返り先の番地や変数の退避などを行うのに使用されます。どちらもリミットチェックが行われており、スタックに積みすぎたり、掘りすぎたりすると、[?STACK1]、[?STACK2]のエラーメッセージが出力されます。

STACK1はこのほか、1行演算が終了した時点で、もとの値に戻っているか、つまり、(の数と)の数と同じであったかどうかのチェックも行われています。

#### ●メモリマップ

機械語とTTLを併用する場合に便利のようにTTLの簡単なメモリマップを書いておきます。編集室のご好意によって、ソースリストが掲載できるようになったので、細かなサブルーチンの配置などについてはそちらを見てください。

B000	ホットスタート	TTL本体
B002	コールドスタート	
	行実行ルーチン	TTLワーク
B8FF	テキストエディタ	
BA60	入出力ブロック	
BB00	カセット出力ルーチン	
BB70	カセット入力ルーチン	
BBE6	ワークエリア	
BC4B	ラインバッファ1	
BD00	第2スタック	
CDF7	第1スタック	
CE00	ラインバッファ2	
CEFF	CPUスタック	

