EDITORIAL

Here we are again ! First, I must apologise for the delay in sending out Issue One: we didn't expect such a tremendous response to the User Club, and at one stage, to say we were on the verge of panic is an understatement !

However, we have been working like mad behind the scenes. All at Genpat, and Memotech, want this to be one of the best user groups around. I visited the factory last week, and I can tell you that we have in the pipe-line,and for exclusive release through the club, the following: **Light Pen, Speech Synthesiser, Modem, and a Cheepo Disc System.** Further news of these add-ons will be announced as they become available.

Between the pages of this edition you find all sorts of **goodies.** Those of you who have been slaving over a hot computer, trying to perfect a **Pixel Scroll** will be pleased to know that your prayers are answered ! We have an excellent one which is fully documented, and can easily be interfaced with your own programs. Also, when you have nothing better to do, type in the **MTX CLOCK....** it's deadly accurate.

We have available, a limited supply of Peter Goode's excellent book, **The Memotech MTX Program Book** at a special member price of £4.75 including P.P. The new **MTX Manual** will be available shortly at £5.50, inclusive, and any members requiring one can book their order now by sending a cheque to Genpat. The new manual includes the much sort after book by Spencer Bateson: **Advanced Programming with the Memotech.**

Don't forget that all **Continental Software** is available at a discount price of 15%, and any orders should be addressed to Genpat - quoting your membership number.

A new software house, **Tri-Soft** is about to release a suite of games for the MTX, and you will find selected reviews elsewhere in this magazine. Also, I can report that **Artic** have shown an interest in the MTX, and Mike Johnson,one of their free-lance programmers, is currently working on a new title (Mike is also a member of the Club). So you see, people are starting to realise, the MTX is one of the best computers on the market.

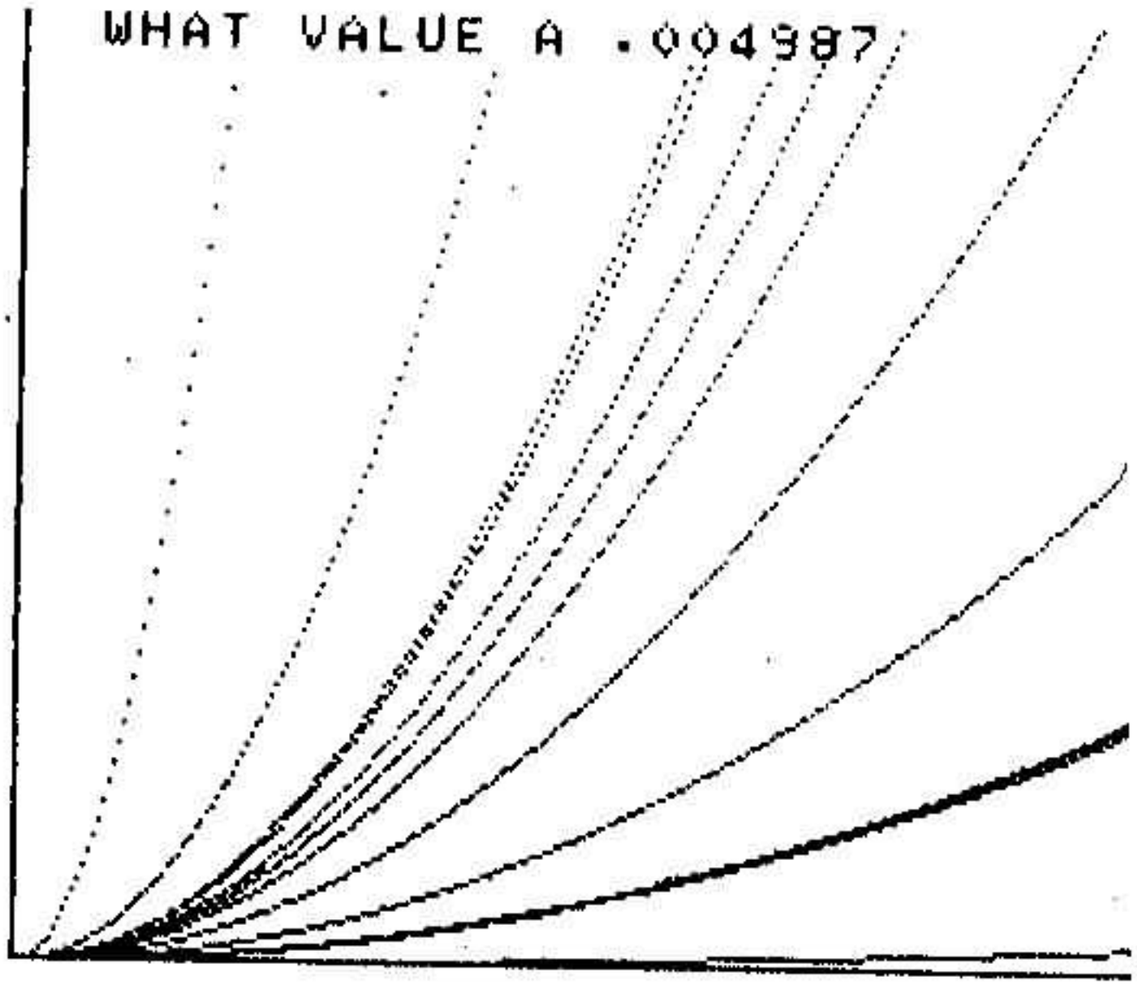
Finally, I would like to thank all those members who have shared their knowledge with us, and have contributed to this edition. Keep those programs and articles coming in - we need them !

Keith Hook.
Editor.

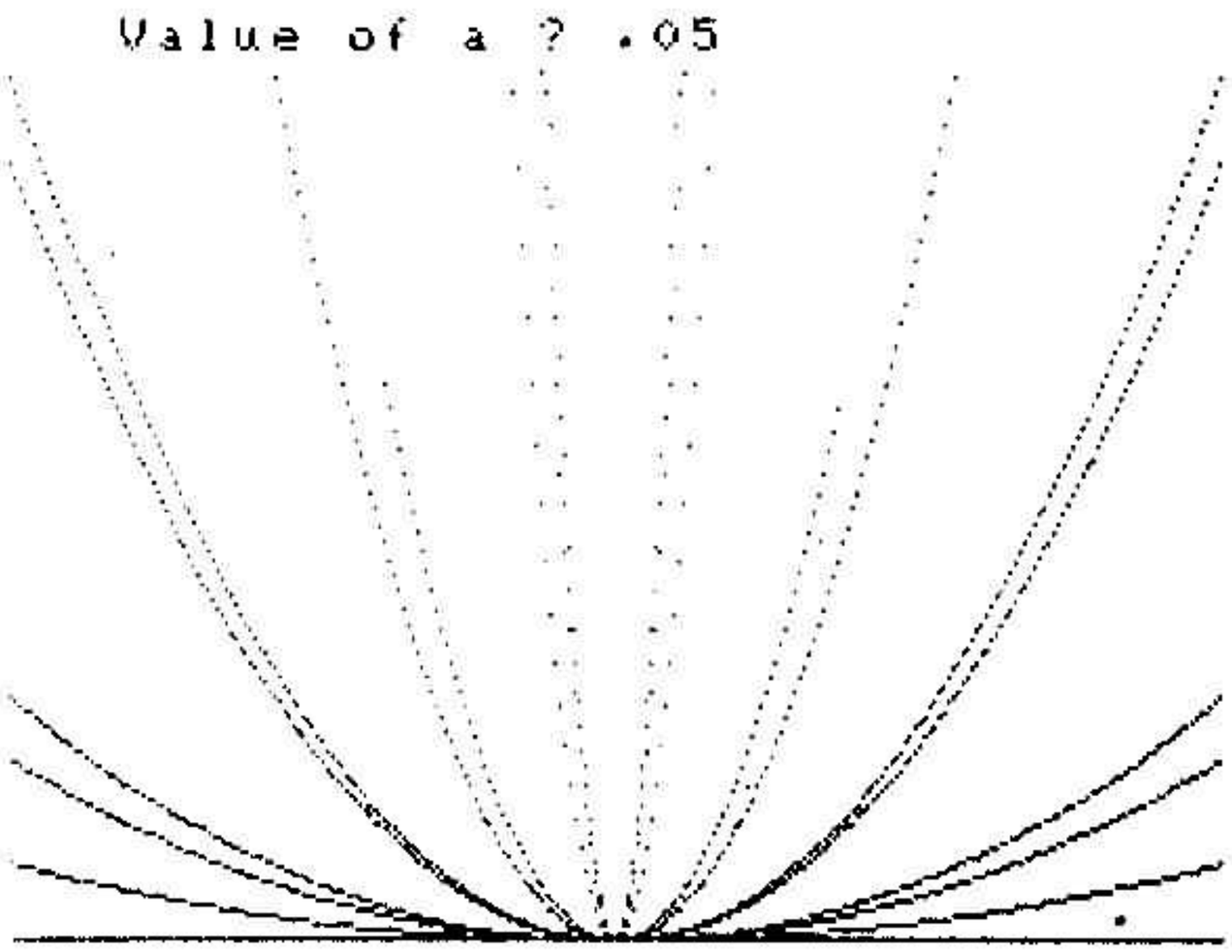
If you send a letter, and require a reply, PLEASE enclose a S.A.E

Here's some little graphic routines for you to play with..... try altering the COS/SIN values. Infact, try altering any value.

```
10 VS 4
20 COLOUR 4,0: COLOUR 2,3: COLOUR 0,3: CLS
30 INK 15: FOR I=0 TO 255: LET Y=0: PLOT I,Y: NEXT : FOR Y=0 TO 191
33 FOR Y=0 TO 191: LET X=0: PLOT X,Y: NEXT
35 INK 1: CSR 2,0: INPUT "WHAT VALUE A ";A
40 FOR X=0 TO 255
50 LET Y=INT(0+A*X*X)
60 IF Y>191 THEN GOTO 100
70 PLOT X,Y
80 NEXT X
100 GOTO 35
```

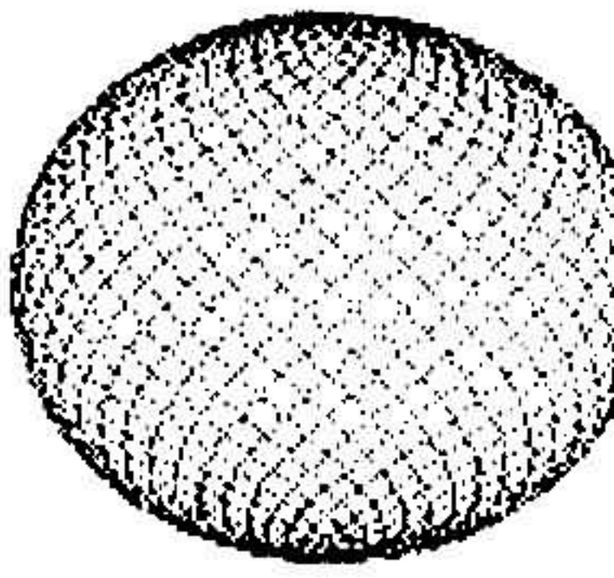


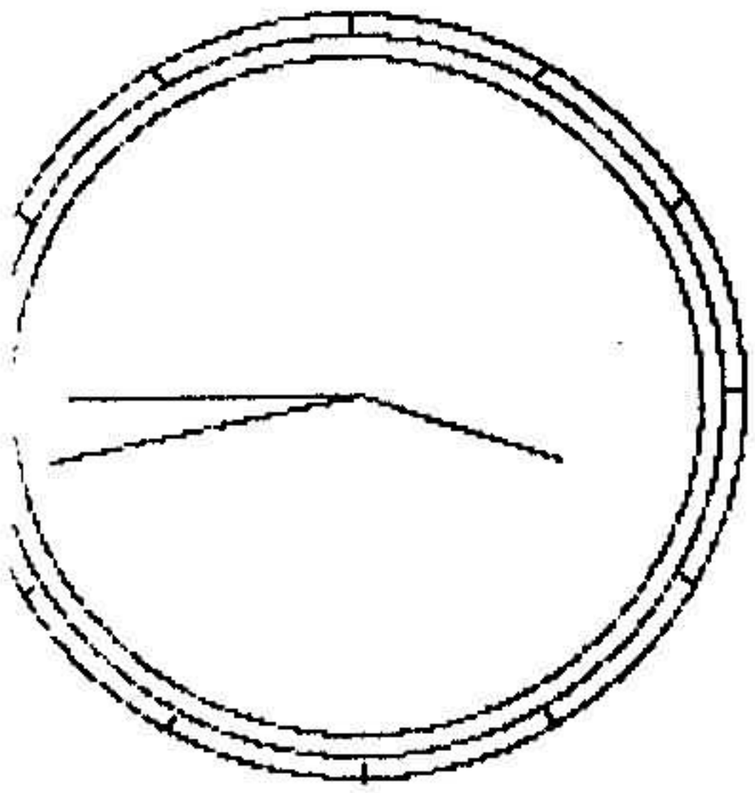
CURVE SCREEN




```
10 VS 4
20 CLS : INK 15
50 LET Y=0
60 FOR X=0 TO 254
70 PLOT X,Y
80 NEXT X: INK 1
85 CSR 2,0: PRINT CHR$(5);: INPUT " Value of a ? ";A;
90 FOR I=0 TO 127
100 LET Y=INT(A*I*I)
110 IF Y>191 THEN GOTO 150
120 PLOT (127-X),Y
130 PLOT (127+X),Y
140 NEXT
150 GOTO 85
```

```
10 VS 4: CLS
20 FOR A=0 TO 125.7 STEP .03
30 PLOT 128+(55*SIN(A)),96+(65*COS(A)*SIN(A*.95))
40 NEXT
```





When you have nothing to put on the computer.....type this in. It's deadly accurate... you'll be amazed. Single numbers need a leading zero when following the prompts.

PROGRAM

```
1 REM*************************************************************************
2 REM  MEMOPAD 12 HOUR CLOCK  K.HOOK
3 REM*************************************************************************
10 VS 4: CLS
20 DIM HR$(2),SEC$(2),MIN$(2): LET E=128: LET F=98: LET R=PI/30: LET COR=-15
30 LET TSX=0: LET TSY=0: LET LSY=0: LET LSX=0: LET MZ=0: LET Z=0
35 COLOUR 4,10: PRINT CHR$(4);CHR$(10): INK 1
60 CSR 2,22: INPUT "HOURS ";HR$: LET HR=VAL(HR$)
65 IF HR>12 THEN GOTO 60
70 CSR 2,22: PRINT CHR$(5);: INPUT "MINUTES ";MIN$: LET MIN=VAL(MIN$)
75 IF MIN>60 THEN GOTO 70
80 CSR 2,22: PRINT CHR$(5);: INPUT "SECONDS ";SEC$: LET SEC=VAL(SEC$)
90 IF SEC>60 THEN GOTO 80
100 LET T$=LEFT$(HR$,2)+LEFT$(MIN$,2)+LEFT$(SEC$,2)
110 LET HR=HR*5+INT(MIN/12): REM  CORRECT  HOURS TO SIN & COS
120 CLOCK "000000": CLOCK T$
130 ATTR 2,0
140 CLS : INK 15: CIRCLE E,F,90: CIRCLE E,F,85: CIRCLE E,F,80
150 COLOUR 1,8: FOR I=1 TO 12
160 LET NM=I/6*PI
170 LINE E+86*SIN(NM),F-86*COS(NM),E+90*SIN(NM),F-90*COS(NM)
180 NEXT
190 LET Z=(HR+COR)*R
200 LET LHX=E+50*COS(Z)
210 LET LHY=F-50*SIN(Z)
220 LET MZ=(MIN+COR)*R
230 LET LMX=E+65*COS(MZ)
240 LET LMY=F-65*SIN(MZ)
280 LET SEC1=VAL(RIGHT$(TIME$,2)): IF SEC=SEC1 THEN GOTO 280
290 LET SEC=SEC1: LET SAN=(SEC+COR)*R
300 LET LSX=E+72*COS(SAN)
310 LET LSY=F-72*SIN(SAN)
320 ATTR 2,1: LINE E,F,TSX,TSY: ATTR 2,0
325 INK 5
330 LINE E,F,LSX,LSY: LET TSX=LSX: LET TSY=LSY
340 LINE E,F,LMX,LMY
350 LINE E,F,LHX,LHY
360 IF SEC<>00 THEN GOTO 280
370 ATTR 2,1: LET MIN=MIN+1: LINE E,F,LMX,LMY
380 IF (MIN-INT(MIN/12)*12)<>0 THEN ATTR 2,0: GOTO 220
390 LET HR=HR+1: LINE E,F,LHX,LHY: ATTR 2,0: GOTO 190
```

Tip

A request for help posed a problem earlier this week: ' How do you simulate the **Instring$** function on the 512 ?'. Yes, well, er.....

```
B$ = Search $tring
A$ = $string to search
10 LET B = LEN(B$)
20 LET L = LEN(A$)-B
30 FOR I = 1 TO L
40 IF A$(I,B) = B$ THEN GOTO Found
50 NEXT
```


SPECIAL NOTICE RST 10 CALL NUMBER 18

I have had lots of calls about the RST 10 information sheet that was posted to those members who requested it. I am pleased to announce that we have now solved the elusive bug.

The answer lies in the fact that this command expects two byte words for the x,y data. The format for using this instruction is :-

```
RST 10
DB £8A
DB P,LSB FOR X,MSB FOR X,LSB FOR Y,MSB FOR Y
DB LSB X1,MSB X1,LSB Y1,MSB Y1,COL
```

We have tested this routine extensively, and in every situation it was found to perform satisfactorily.

MEMOPAD

ASSEMBLY LANGUAGE

PART 2

In the old days, the only way to write **assembly language** programs was to **hand assemble** each instruction. As you can imagine, this was a laborious and time consuming task. Today, **Assemblers** have made the task far easier.

The **assembler** translates your code, written with mnemonic instructions, and called the **source program**, into the **object program**, a machine language program which the MTX executes when loaded into the computer.

INPUT TO ASSEMBLER ===> SOURCE CODE

OUTPUT FROM ASSEMBLER ==> OBJECT CODE

With the MTX assembler, there is no need to save separate files of **source code and object code** because the Memotech automatically displays the source code when the **List** command is given, and each line is assembled internally as the mnemonics are entered from the keyboard.

Before you can attempt to use assembly language, you must learn the instructions - as you did with Basic - and how to use them. With Basic you stored your values in **variables** e.g. X, NUM,X1,Y$, etc. In assembly language programming values are stored in memory locations, or in **registers.**

The Z80 processor contains two sets of internal, general registers, and six special purpose registers. Take a look at the following:-

	A	F	A'	F'	ALTERNATE
GENERAL	B	C	B'	C'	GENERAL
REGISTERS	D	E	D'	E'	REGISTERS
	H	L	H'	L'	

Z80 REGISTERS

IX	IY	SP	PC	I	R

SPECIAL PURPOSE REGISTERS

A,B,C,D,E,F,H,and L are the normal general registers and the registers designated ' are the **alternate** register set, which can only be accessed by the two instructions **EX AF,AF'** AND **EXX** - these two instructions exchange the contents of the main set with that of the alternate set. Only one set of registers can be used at one time. Following the two sets of **8 Bit** registers are four **16 Bit** registers : **IX,IY,SP,PC.**

Registers **I & R** are very seldom used in most normal programming applications.

The **A** register is also referred to as the **accumulator** because all of the arithmetic instructions, and most of the other instructions use the contents of the **A** register as an operand. In fact, this is where most of the transfers take place.

The **F** register is also called the **Flag** register. The **F** register sets or re-sets bits internally to indicate a **true or false** type of condition, and is **never** used for computations.

The **Program Counter** or **PC** is a **16 bit** register that points to the current memory location which hold the instruction to be executed.

Another 16 bit register is the **SP** or **Stack Pointer**. This register keeps a check on the position of the **STACK** in RAM. Two 16 bit registers with very powerful programming possibilties are the **index registers; IX & IY.**

Each of the 8 Bit registers can be used separately or in set pairs [BC,DE,HL] and treated as 16 Bit registers.

Assemblers have their own set of rules, but they aren't difficult to learn:-

```
DB or DEFB  =====>    Define Byte.......DB £FA,'T','ONE'
DW or DEFW  =====>    Define Word.......DW £4007 or Label
DS or DEFS  =====>    Define Space......DS 245  reserve  245 bytes
DM or DEFM  =====>    Define Message....DM 'ANOTHER GAME ?'
```

All the above are known as **Pseudo operations**, and are used by the assembler, **not the CPU**, to carry out predefined functions.

LABELS are used to reference one instruction to another. For example: **JP Z,AGAIN.** A label can be compared with a line number in Basic; e.g **IF A = 0 THEN GOTO 100.**

The semi-colon **;** is used in the same way as the REM statement in Basic, and the assembler ignores all that follows. It is good programming practice to get into the habit of documenting your program. Believe me, when you look at your code after a few months, you will find it hard to understand what you had in mind at the time you wrote the program.

The convention used by most assemblers is as follows:-

Label	**Op Code**	**Operand**	**Remarks**
START:	LD A,	(DE)	;Put score in A reg.

MTX512

If you have ever re-configured VRAM to your own format, you will know what a pain it is to design, and load all the Ascii characters back into your own **Pattern Generator Table.** Well! Toil no more. Read on.....

The MTX ROM stores its **Ascii characters** from location £35B3 on page zero. Special precautions should be taken as the characters are stored in **5 bytes** which means that they must be rotated to allow them to be printed to the screen. The coding below does all the hard work for you. Obviously, immediately before this code you should have sent the VRAM address to the VDP chip. **ASCII = £35B3 and PORTDO = 1.** In the listing I have had to code each step, but you can replace the **LD E,(HL) :INC HL:LD D,(HL):INC HL** with **RST 8** because this is exactly what the **RST8** call does: it loads the E reg with the contents of the memory location pointed to by HL then incs HL , and LDs reg D with the contents of memory loccation pointed to by HL then increments HL again before returning to caller.

```
TITLE   ARCADIAN        ;DATE 30.5.84
;AUTHOR:-   KEITH HOOK
;
ASEG
;
                        EXT ASCII

        PORTA0  EQU  2
        PORTD0  EQU  1

        ;
        ; SEND DATA TO VRAM
        ; HIDDEN IN 1900H (TEXT RAM)
        ;
SENASC:
        LD  HL,5900H
        CALL  ADDOUT
        LD HL,ASCII
        LD B,96
SENAS1:
        PUSH  BC
        PUSH  HL
        LD  C,(HL)
        INC  HL
        LD   E,(HL)
        INC  HL
        LD  D,(HL)
        INC  HL         ; RST 8
        LD  A,(HL)
        INC  HL
        LD  H,(HL)
        LD  L,A
        LD  B,08        ; LOAD REGISTERS WITH 5 CHARS
SENAS2:
        XOR  A
        RL   H
        RR   A
        RL   L
        RRA
        RL   D
        RRA
        RL   E
        RRA
        RL   C
        RRA
        OUT (PORTD0),A
        DJNZ SENAS2
        POP  HL
        LD  C,5
        ADD  HL,BC
        POP  BC
        DJNZ SENAS1

ADDOUT:
        LD   A,L
        OUT  (PORTA0),A
        LD   A,H
ADOUT2: OUT  (PORTA0),A
        RET
```

MEMOPAD

SYSTEM VARIABLES £FA85 USYNT & £FA89 USER

These two addresses are used to interface your own routines with MTX basic. £FA89 is loaded with a jump to your own routine. The syntax for your new command must be **USER <NAME>** where NAME can be a new basic command e.g FILL.=====> USER FILL <parameters>.

When Basic encounters your new command it will then check with £FA85 to check what the syntax should be for the new command. Basic will check for various syntax depending on what has been loaded into £FA88 down to £FA85. The syntax bytes are as follows:

0 => Expect a numeric expression
1 => Expect a string expression
2 => Expect an arithmetic expression
3 => Expect a list of expressions separated by ',' or ';'
4 => Expect a list of numbers separated by "," from 0 - 64K: FILL 93,2,1
5 => Expect a list of arithmetic expressions
6 => Expect a single number in range 0 - 64K
7 => Check Nothing
There are various other values for the byte but those listed above are the main ones.

If your command was going to check nothing you would load £FA88 with 7 and £FA87 with £C9 (RET).....the last byte must **always** be a RET instruction.

Suppose the parameters for your routine required a series of numbers along the lines of the GENPAT statement e.g

USER FILL 3,5,234,8,8

Then you would set up USYNT as follows

FA88 3
FA87 £C9

Step One: Load USER with JP to your routine.
Step Two: Load USYNT starting at last byte £FA88 with synta checking bytes.
Step Three: Make sure last location of USYNT is loaded with a RET instruction.

Also note that £FA89 is loaded LSB/MSB notation

Any one not quite sure how to go about this can send for a full listing from Genpat.

The easiest way is to use syntax byte 7 until you have debugged the routine then insert the syntax checking.

MEMOPAD