Index

wersja polska

Casio FX-700P - data formats

Character set

Code of the "space" character is $00.

BASIC tokens

```
$80 SIN     $90 LEN(    $A0 FOR     $B0 VAC     $C0 DEFM
$81 COS     $91 VAL(    $A1 NEXT    $B1 SET     $C1 SAVE
$82 TAN     $92 MID(    $A2 GOTO    $B2 PUT     $C2 LOAD
$83 ASN     $93 KEY     $A3 GOSUB   $B3 GET     $C3 VER
$84 ACS     $94 CSR     $A4 RETURN  $B4         $C4 LIST
$85 ATN     $95 TO      $A5 IF      $B5         $C5 RUN
$86 LOG     $96 STEP    $A6 PRINT   $B6         $C6 CLEAR
$87 LN      $97 THEN    $A7 INPUT   $B7         $C7
$88 EXP     $98         $A8 MODE    $B8         $C8
$89 SQR     $99         $A9 STOP    $B9         $C9
$8A INT     $9A         $AA END     $BA         $CA
$8B FRAC    $9B         $AB         $BB         $CB
$8C ABS     $9C         $AC         $BC         $CC
$8D SGN     $9D         $AD         $BD         $CD
$8E RND(    $9E         $AE         $BE         $CE
$8F RAN#    $9F         $AF         $BF         $CF
```

Internal data representation

Each variable (except the string variable **$**) occupies 8 bytes (i.e. 16 4-bit words) of RAM. Data are stored with least significant word first (i.e. at lower memory address).

Special case

Value 00 00 00 00 00 00 00 00 represents both numerical 0 and an empty string. The **VAC** command initialises all variables with this value. Both of the following statements write this value to the **A** or **A$** variable.

```
A = 0
A$ = ""
```

String variables

A string variable can hold up to 7 characters.
First 7 bytes (i.e. 14 4-bit words) hold the character codes. Spare locations are padded with 00.
15th word is not used and contains 0.
The last (16th) word contains the length of the string (non-zero). This distinguishes the string from the numerical variable, where the last word contains 0.
Examples:

```
A$ = "ABMN"          02 12 C2 D2 00 00 00 04
A$ = "1234567"       11 21 31 41 51 61 71 07
```

Numerical variables

Numerical values are stored in packed decimal floating point format.
First two 4-bit words contain the last two digits of the exponent, the least significant digit first. The stored exponent value is biased by adding an offset 100 to put it within an unsigned range.
Third 4-bit word contains the first digit of the biased exponent and the sign of the mantissa:

```
value = first_digit_of_the_exponent + 5 * sign_of_the_mantissa
```

List of all possible combinations:

```
0 - mantissa positive, exponent negative
1 - mantissa positive, exponent positive
5 - mantissa negative, exponent negative
6 - mantissa negative, exponent positive
```

Next twelve 4-bit words contain the mantissa in range 1.00000000000 to 9.99999999999, the least significant digit first.
The last (16-th) 4-bit word contains 0 to denote the numerical variable (as opposed to the string, where this value is non-zero).
Examples:

```
A = 1             00 10 00 00 00 00 00 10   (1.00000000000E00)
A = -1            00 60 00 00 00 00 00 10   (-1.00000000000E00)
A = 100           20 10 00 00 00 00 00 10   (1.00000000000E02)
A = -100          20 60 00 00 00 00 00 10   (-1.00000000000E02)
A = PI            00 10 63 56 29 51 41 30   (3.14159265360E00)
A = -PI           00 60 63 56 29 51 41 30   (-3.14159265360E00)
A = 0.01          89 00 00 00 00 00 00 10   (1.00000000000E-02)
A = -0.01         89 50 00 00 00 00 00 10   (-1.00000000000E-02)
A = 1/3           99 03 33 33 33 33 33 30   (3.33333333333E-01)
A = -1/3          99 53 33 33 33 33 33 30   (-3.33333333333E-01)
```

Special string variable $

The string variable **$** can hold up to 30 characters. The string stored in the memory is preceded by string length, and terminated by the $FF end marker.
Example:

```
$ = "ABCDEFG"
70 02 12 22 32 42 52 62 FF
```

BASIC program structure

BASIC line begins with a line number stored in 2 bytes (four 4-bit words) in packed decimal format, ends with an end marker $FF. BASIC keywords are stored as single byte tokens, numeric values as strings of characters, colons used as statements separators as $FE.
Example:

```
1234 FOR I=1 TO 49 STEP 1: NEXT I
43 21 0A 82 C0 11 59 41 91 69 11 EF 1A 82 FF
```

FOR stack

Each time a FOR statement is executed, a FOR control structure described below is pushed on the stack. The stack entry is freed by marking it with an $F when the loop is terminated. There's no stack pointer used, the FOR statement scans the stack for the first unused location instead.

```
 2 words        TO value, two last digits of the exponent
 1 word         TO value, first digit of the exponent and sign
10 words        TO value, mantissa
 2 words        index of the control variable
                (00 = variable A, 10 = variable B and so on)
 1 word         F - free entry, 2 - occupied entry
 2 words        STEP value, two last digits of the exponent
 1 word         STEP value, first digit of exponent and sign
10 words        STEP value, mantissa
 3 words        address of the first character after the FOR
                statement, it's the place where the NEXT iteration
                loop resumes execution
```

Example:

```
1234 FOR I=1 TO 49 STEP 1: NEXT I

10              TO value = 4.90000000E01
1               mantissa and exponent positive
0000000094
80              index of the I variable
2               denotes an occupied entry
00              STEP value = 1.0000000E00
1               mantissa and exponent positive
0000000001
662             points to the colon after the STEP 1 statement
```

GOSUB stack

Executing a GOSUB statement pushes a 4-word structure described below on the GOSUB stack. RETURN frees the top stack location by marking it with a $F word. The concept doesn't use any stack pointer either, similar to the way the FOR stack is implemented.

```
3 words        address of the first character after the GOSUB
               statement (the RETURN point)
1 word         F - free entry, 3 - occupied entry
```

Memory map

```
$0000-$007F     128 words   general purpose buffer
$0080-$00FF     128 words   general purpose buffer
$0100-$010F     16 words    variable ANS
$0110-$011F     16 words    RAN# seed, initial value 0.43429448190
$0120-$012F     16 words    character position selected by MID( function
$0132-$0133     2 words     data sent to the printer
$0140-$01BF     128 words   FOR stack, holds 4 entries, grows upwards
$01C0-$01DF     32 words    GOSUB stack, holds 8 entries, grows upwards
$01E0-$021F     64 words    the special string variable $
$0220-$0E5F     3136 words  BASIC programs
$0E60-$0E6F     16 words    variable Z
$0E70-$0E7F     16 words    variable Y
   .               .           .
   .               .           .
   .               .           .
$0FD0-$0FDF     16 words    variable C
$0FE0-$0FEF     16 words    variable B
$0FF0-$0FFF     16 words    variable A
```

File format

A file consists of a name segment followed by one or more data segments.

File name segment

A name segment begins with a byte $Dx (file created with **SAVE**), $Ex (file created with **PUT**), or $Fx (file created with **SAVE A**), where x is the file name length (up to 8 characters), or $F for a file without a name.
Next 8 bytes contain the file name characters.
Last 2 bytes contain the first line number of the BASIC program stored with **SAVE**.

Data segments

A data segment begins with a $02 byte, ends with a $F0 byte (when it is the last segment), or a $F1 byte (when more data segments will follow). The number of the data bytes between these pair of characters cannot exceed 63, because the contents of the segment is loaded to the 64-byte buffer at address $0000. This limit doesn't apply to files saved with **SAVE A** which contain only a single data segment.

A BASIC program consists of a list of BASIC lines. When the file was created with **SAVE A** each program is followed with a $E0 byte. An empty program is stored as $E0 alone.

File stored with **PUT** consists of a list of variables separated by $FF bytes. No variable names are stored.

Examples:

```
1.  BASIC Program stored with SAVE "PROG"

    File name segment:
    $D4 $2F $31 $2E $26 $07 $FF $10 $FF $10 $00

    Data segment:
    $02
    $10 $00 $A0 $28 $0C $11 $95 $11 $10 $FF   10 FOR I=1 TO 10
    $20 $00 $A6 $28 $FF                       20 PRINT I
    $30 $00 $A1 $28 $FF                       30 NEXT I
    $F0

2.  Variable $ and a numeric variable stored with PUT "*-+" $,A

    File name segment:
    $E3 $03 $02 $01 $07 $09 $5D $20 $FF $00 $F8

    Data segment:
    $02
    $0D $11 $12 $13 $14 $15 $16 $17       $="1234567890ABC"
    $18 $19 $10 $20 $21 $22 $FF $00
    $01 $00 $00 $00 $00 $00 $00 $00
    $01 $00 $00 $00 $00 $00 $00 $FF
    $00 $01 $36 $65 $92 $15 $14 $03       A=3.14159265360
    $F0

3.  BASIC Programs stored with SAVE A "*"

    File name segment:
    $F1 $03 $07 $FF $00 $00 $00 $00 $07 $00 $68
```

```
Data segment:
$02
$01 $00 $A6 $FF $E0                 P0: 1 PRINT
$10 $00 $A2 $11 $10 $FF $E0         P1: 10 GOTO 10
$E0 $E0 $E0 $E0 $E0 $E0 $E0 $E0
$F0
```