# SAVE-ETC

T\_ADDR = 1 (fonction : load)

BC = 34 (place réservée dans le workspace)

DE = IX = adresse réservée dans le workspace (BC-SPACES)

écriture de 11 espaces au début du workspace réservé

(1 file type indicator + 10 file name)

écriture de FF en position 1 : file name = null

routine STK-FETCH fetches the filename parameters from the calculator stack.

- length of string in BC.

- start of string in DE

nom de fichier vide interdit pour save

couper la longueur de nom de fichier à 10

si string non nul : copier le nombre de caractères limité depuis l'adresse dans le workspace (si moins de 10 caractères, complété par des espaces)

GET-CHAR :

token DATA ? (E4)

...

token SCREEN$ ? (AA)

écriture dans le descripteur de fichier

- index 0 : 3 (file type 3)

- index 11 : length\_low = $00, index 12 : length\_high = $1B

- index 13 : start\_low = $00, index 14 : start\_high = $40

HL = start

token CODE ? (AF)

si paramètres start adress + length en BASIC (sinon 0, 0 par défaut)

écriture dans le descripteur de fichier

- index 0 : 3 (file type 3)

- index 11 : length\_low, index 12 : length\_high

- index 13 : start\_low, index 14 : start\_high

HL = start

=> continue à SA-ALL

token LINE ? (CA) (pour save basic)

écriture dans le descripteur de fichier

- index 0 : 0 (file type 0)

- index 13 : auto start line\_low, index 14 : auto start line\_high

(auto start line = 32768 par défaut)

HL = E\_LINE

DE = PROG : total length = PROG - E\_LINE

- index 11 : total length\_low, index 12 : total length\_high

HL = VARS : program length = total length - VARS

- index 15 : program length\_low, index 16 : program lenght\_high

HL = start, DE = length

# SA-ALL

Load header from tape into second descriptor of workspace (17 bytes, loop until loading fails)

Compare loaded header type to expected (first descriptor) => type mismatch, loop

Affiche le nom de fichier et compare le nom de fichier attendu => si mismatch, loop

Si type lu = 3 (code, 2ème buffer) => VR-CONTRL

* Lire l’adresse cible soit dans le premier buffer (instruction LOAD), soit dans le deuxième buffer (header sur cassette)
* Lire la taille des données dans le deuxième buffer
* Charger les données du bloc en mémoire par LD-BYTES

Sinon => LD-CONTRL

* Calcule adresse début programme basic et charger
* Positionne la variable système de l’adresse des variables
* Positionne l’adresse de la prochaine ligne à exécuter si autostart
* Charge les données basic en mémoire par autostart

# 0556 : LD-BYTES

*A : 0x00 for header / 0xFF for data*

*CF : set to load, reset to verify*

*IX : start of the receiving location*

*DE : length of the bytes to be loaded*

ZF->Reset

AF’->Save AF

Disable interrupts

Border white and MIC off

Stack->Push address **SA/LD-RET** (053F)

A->Read EAR state in bit 5

A->Set bit 1 (red border colour)

C->Save A (store initial state long-term)

ZF->Set

# 056B : LD-BREAK

**Return** if not zero flag (any time space is pressed)

# 056C : LD-START

Call **LD-EDGE-1** (CF set if success)

Loop to **LD-BREAK** in case of failure (CF not set)

HL->1045 (outer loop counter for 1 second delay)

# 0574 : LD-WAIT

Decrement B and loop to **LD-WAIT** until B=0 (256 times)

Decrement HL and loop to **LD-WAIT** until HL = 0 (outer loop counter)

Call **LD-EDGE-2** (sample 256 edges to check that we are in the middle of la lead-in section)

Loop to **LD-BREAK** in case of failure (CF not set)

# 0580 : LD-LEADER

B->156 (timing value)

Call **LD-EDGE-2**

Loop to **LD-BREAK** in case of failure (CF not set)

Compare B with 198

Loop to **LD-START** if B < 198 (edges too close together for a lead-in)

Increment H and loop to **LD-LEADER** while H !=0 (test 256 edge samples)

// we are in the middle of a 2 or 5 seconds lead-in, test every edge looking for the terminal sync signal

# 058F : LD-SYNC

C->201 (timing value)

Call LD-EDGE-1

Loop to **LD-BREAK** in case of failure (CF not set) (timeout)

Compare B with 212

Loop to **LD-SYNC** if B < 212 (gap too big for sync, normal lead-in edge gap)

Call LD-EDGE-1 (if sync pulse, another edge should appear before B rises to FF)

**Return** in case of failure (CF not set) (timeout)

C->C Xor 3 (cyan-> blue / red -> yellow)

H->0 (parity byte)

B->176 (timing)

Forward to **LD-MARKER** (with ZF reset to indicate first byte is discarded)

………………………

# 05C8 : LD-MARKER

L->00000001

……………………..

# 05E3 : LD-EDGE-2

Call **LD-EDGE-1**

**Return** if CF is not set (space pressed or timeout) (else continue to look for another adjacent edge)

# 05E7 : LD-EDGE-1

A<-22

# 05E9 : LD-DELAY

Decrement A and loop to **LD-DELAY** until A=0

CF->Reset

# 05ED : LD-SAMPLE

B->Increment (timeout counter) 🡺 when B was set ?

**Return** if B>255 (with failure)

A->Read EAR state in bit 5 and space key in CF (CF = 0 if key pressed)

**Return** if not CF (if space key pressed)

Compare bit 5 of A and bit 5 of C (long term value of EAR)

Loop to **LD-SAMPLE** if EAR dit not change

C->Invert all bits of C (invert long-term value)

A->Get 3 rightmost bits of C (border color bits, red for initial EAR value -> cyan for other EAR value)

Set border color (cyan/red) and MIC off

CF->Set (signaling edge found within time allowed)

**Return**