bits

NOVEMBRE 1.982

1 El següent programa permet d'obtenir qualsevol dels 128 Tones. N'hi ha prou amb introduir el nº del Tone (del Ø al 127) i fer XEQ "TN". Com a característica addicional, el contingut del registre X és incrementat d'una unitat, de manera que successives pressions de la tecla [R/S] produeixen els Tones següents:

```
Ø1  LBL "TN"
Ø2  "▯"          (codi: F1 9F)
Ø3  XTOA
Ø4  "⊢▯▯"        (codi: F3 7F 91 7C)
Ø5  E6
Ø6  X<> b
Ø7  SIGN
Ø8  +
Ø9  END
```

Les línies Ø2 a Ø4 col·loquen al registre alfabètic els codis de les funcions TONE n, STO b, (essent n el contingut del registre X). Les línies Ø5 i Ø6 fan seguir l'execució del programa al registre M, tot salvant l'adreça actual a X, i així s'executa el Tone volgut i es torna a les línies Ø7 i Ø8, que incrementen el nº del Tone.

2 La funció 2↑X no existeix a la HP-41. Una curiosa manera d'obtenir-la bo i cumplint els requisits de tota funció monàdica (és a dir: salvant l'antic contingut de X a L, i conservant la resta de l'escala) és la següent:

```
Ø1  LBL "2↑X"
Ø2  FRC
Ø3  X<>F
Ø4  SF IND L
Ø5  X<>F
Ø6  END
```

Hi ha, però, una limitació. El programa sols treballa correctament quan els valors de X són enters positius inferiors a 8.

3 A vegades dins d'un programa hem d'executar un bucle exactament dos cops. En aquests casos, podem emprar el mètode clàssic amb comptador, o un de similar amb un flag.
El sistema més curt, que no requereix ni etiqueta ni funció de test, és el següent:

```
RCL b
(contingut del bucle)
X<> b
```

Així estalviem octets i l'execució és més ràpida. Sols cal prestar atenció al fet que el registre X no sigui modificat dins del bucle (o si ho és, a restablir-ne el valor).
Exemple: per executar dos cops la successió dels Tones 5, 6 i 4, faríem

```
RCL b
TONE 5
TONE 6
TONE 4
X<> b
```

4 En programes inter-actius, és corrent de trobar preguntes a les quals cal respondre afirmativament o negativa. Succeeix, però, que a vegades no recordem si per dir que SI cal escriure sols la inicial o tota la paraula (o potser només [R/S]).
A més, si volem adaptar-ho a unaltre idioma, ens cal modificar el programa.
La següent rutina resol aquests problemes, car interpreta correctament la resposta tant si escrivim la inicial com tota la paraula, i això en diferents idiomes, i fins i tot amb diferents expressions. Assageu, per exemple: (en català) SI, NO, CERT, FALS, AFIRMATIU, NEGATIU, BE, MALAMENT, (en francès) OUI, NON, (en anglès) YES, NO, (en basc) BAI, EZ, etc.
A més, interpreta la pulsació directa de [R/S] (sense escriure res) com a SI, per allò de què "qui calla, atorga":

```
Ø1  LBL "?"
Ø2  CF ØØ
Ø3  CF 23
Ø4  AON
Ø5  PROMPT
Ø6  AOFF
Ø7  FC?C 23
Ø8  SF ØØ
Ø9  ATOX
1Ø  5
11  −
12  8
13  MOD
14  2
15  X<=Y?
16  SF ØØ
17  END
```

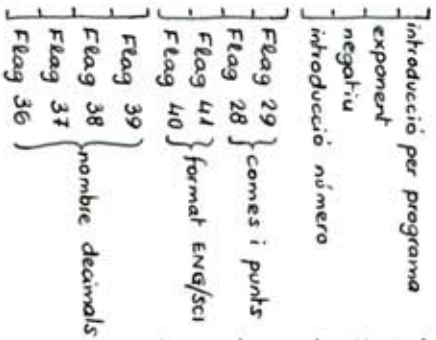
Per utilitzar-lo dins d'un programa, cal introduir el text de la pregunta a Alpha, i fer XEQ "?". Si la resposta és positiva, s'encendrà el flag ØØ, i en cas contrari, s'apagarà.

5 Les funcions XTOA i ATOX ens permeten fer servir el registre alfabètic com una pila FIFO que permet emmagatzemar fins a 24 nºs enters de 1 a 255.
Llavors ALENG ens indica el grau d'ompliment de la pila, i AROT alterar-ne l'ordre, al temps que POSA comprovar l'existència d'un element.

bits

NOVEMBRE 1.982

6 Quan introduïm un número, tant si ho fem manualment com dins d'un programa, es produeixen una sèrie d'alteracions en els registres P i Q. Més concretament, els vuit primers bits del registre P contenen l'estat d'una sèrie de flags relatius al mode de presentació en pantalla; i els quatre següents una sèrie d'indicacions accessòries segons el següent esquema:



Una possible aplicació d'això és la següent rutina que ens indica el nombre de decimals de la representació actual (format utilitzat):

```
Ø1  LBL "ND"
Ø2  1,Ø1
Ø3  STO N
Ø4  RCL P
Ø5  STO M
Ø6  "+A"
Ø7  RCL N
Ø8  LOG
Ø9  E1
1Ø  /
11  INT
12  END
```

N'hi ha prou amb fer XEQ "ND"
Cal observar que la funció LOG treballa també amb exponents no normalitzats. Els dígits superiors a 9 són interpretats com a zero.

7 Hi ha un caràcter que potser no haureu vist mai a la pantalla de la HP-41. Es tracta del símbol Π, dibuixat així:

No figura a la taula de caràcters del display ni tampoc se'n parla al Synthetic Programming.
Una forma de visualitzar-lo és la següent: amb un programa del tipus "KA", assignem els codis 64, Ø a una tecla qualsevol.
Després pulsem aquesta teda en mode USER, i observarem el següent:
S'omple la pantalla de símbols Π que tot seguit desapareixen, quedant la pantalla en blanc i la màquina bloquejada. Després d'uns segons, s'executa la funció suma i retorna el control del dauer.

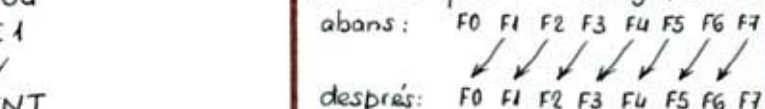
8 Encara que en el manual no queda clar, les funcions X<>F i XTOA admeten també valors negatius i fraccionaris. Així, per exemple:

```
-25,876  XTOA
```

és equivalent a:

```
25  XTOA
```

Basats en això, podem fer un senzill programa que desplaça un bit a l'esquerra els vuit primers flags, així:



El programa és:

```
Ø1  LBL "RE"
Ø2  CLX
Ø3  X<>F
Ø4  2
Ø5  /
Ø6  X<>F
Ø7  END
```

A l'inrevés, per desplaçar els flags cap a la dreta, podem emprar la següent rutina:

```
Ø1  LBL "RD"
Ø2  CLX
Ø3  CF Ø7
Ø4  X<>F
Ø5  2
Ø6  *
Ø7  X<>F
Ø8  END
```

9 La funció AROT treballa també amb números superiors a la llargada de la cadena alfabètica.
En aquests casos funciona com la funció MOD, és a dir, si fem 27 XEQ "AROT" i la llargada d'Alpha era 12, és com si féssim 3 XEQ "AROT" (car $3 \equiv 27 \bmod 12$).

10 I per acabar, un problema. Sabem que podem introduir un número de diferents maneres, per exemple, podem escriure:

```
365
3,65 E2
0,365 E3
,365 E3        (etc.)
```

Si després pulsem ENTER↑, el número apareix representat segons el format en vigor, independentment del mode d'introducció.
Ara bé, si deixeu la màquina a un company, i aquest introdueix un número i després fa ENTER↑ i us torna la máquina, ¿Com podeu saber de quina de les anteriors maneres ha realitzat l'introducció?
La solució al pròxim número.
(És possible fins i tot distingir si hem escrit 2,5 o bé 2,5Ø).

RAMON CERERols MACIÀ
c/. Provença, 513-515, 5è, 4a.
Barcelona - 25
☎ 241-43-07 (en hores d'oficina)
Soci nº 145

bits

DESEMBRE 1.982

11 Començaré donant la solució al problema plantejat al bits nº (10).

Consistia en saber com ha estat realitzada la introducció d'un número, un cop aquesta ha finalitzat per la pulsació de la tecla ENTER↑ o qualsevol altra funció.

Per a això, cal tenir assignats la funció RCL Q i un programa del tipus del "DECODE" (per exemple, el "DCX", número 78 de la biblioteca del Club.)

Fem RCL Q i executem el "DCX". Els 14 nybbles així decodificats contenen:

- el 1r, un dígit igual a 10 menys el nombre de dígits introduits abans de la coma,
- a partir del 2n, els dígits de la mantissa, completant la resta amb F, fins a l'11è.,
- el 12è, conté un 0 si el signe de l'exponent és positiu, i una D en cas contrari,
- el 13è i el 14è, els dígits de l'exponent, completant amb F els no utilitzats.

Per exemple, si haviem escrit 365,20 E-11 obtindríem:

73652 0 FFFFFD11

En canvi, si escrivim ,3652 E-8 que de fet és equivalent, obtenim:

A3652FFFFFFD8F

Cal remarcar que el signe de la mantissa no es conserva al registre Q, sino al segon bit del segon octet de P (vegeu el bits nº (6)).

12 La següent rutina, curta (25 octets) i ràpida (½ segon), permet de col·locar el contingut del registre M al registre absolut definit en decimal pel contingut del registre X.

```
01 LBL "SA"    (Store Absolut)
02 •
03 SF 25
04 X<> c
05 X<> M
06 STO IND Y
07 X<> M
08 X<> c
09 RDN
10 FC?C 25
11 SF 99
12 RTN
```

És fàcilment utilitzable com a sots-rutina, car conserva els registres X, Y, Z i L, i el contingut d'Alpha. (El registre T és substituit per zero).

Les línies 03, 10 i 11 serveixen per evitar que, en el cas que el registre apuntat no existeixi, es produeixi una condició d'error quan el registre c està modificat, la qual cosa provocaria un "crash". D'aquesta manera, si el registre no existeix, simplement apareix "NONEXISTENT".

Aquest programa equival, en certa manera, al "REG" del Synthetic Programming. A continuació en veurem algunes aplicacions.

13 Per una millor utilització del programa anterior, cal recordar l'organització de la memòria de la HP-41:

Cal parar atenció si modifiquem un LBL alfabètic, un END o un registre d'assignació, car això pot provocar un "crash" en fer el primer PACKING.

14 A vegades necessitem espai a la X-Memory per nous fitxers i ens interessa d'esborrar els antics. Llavors ens cal anar escrivint els noms dels fitxers i fent PURFL.

Això resulta lent, sobre tot si hi havia gaires fitxers.

En canvi, la següent rutina utilitza l'Store Absolut per esborrar ràpidament (en menys d'un segon) tots els fitxers existents:

```
01 LBL "PAF"   (Purge All Files)
02 "■■■■■■■"
03 191
04 GTO "SA"
```

La línia 02 es codifica així:

- en hexadecimal: F7,FF,FF,FF,FF,FF,FF,FF
- en decimal: 247,255,255,255,255,255,255,255

L'explicació del funcionament és que les X-Functions fan servir el registre FF,FF,FF,FF,FF,FF,FF per separar la zona de memòria utilitzada de la que no ho està.

El programa "PAF" simplement col·loca aquesta barrera al principi de la X-Memory, que és al registre absolut 191 (en decimal).

15 Si no ens interessa d'esborrar tots els fitxers, sino solament els n més antics, podem emprar la següent rutina:

```
01 LBL "PF"    (Purge n Files)
02 "■■X■■■■"
03 RCL M
04 LBL 00
05 X<> c
06 RCL M
07 STO 00
08 RDN
09 X<> c
10 PURFL
11 DSE Y
12 GTO 00
13 RTN
```

La línia 02 es codifica així:

- en hexa: F7,10,C0,01,69,0B,F0,C0
- decimal: 247,16,192,1,105,11,240,192

Per esborrar els n primers fitxers, només cal fer n XEQ "PF".

El temps d'execució depèn del nombre i tamany dels fitxers (per exemple, per 3 fitxers d'uns 40 registres cadascun, tarda uns 3 segons).

El programa és de fet una variant del "SA".

El que fa és canviar el nom del primer fitxer, per poder aplicar el PURFL, i repetir l'operació n cops.

Unaltra rutina equivalent és la següent:

```
01 LBL "PF"    (Purge n Files)
02 191
03 "ABCDEFG"
04 LBL 00
05 XEQ "SA"
06 PURFL
07 DSE Y
08 GTO 00
09 RTN
```

16 Si només queda un fitxer a la X-Memory i l'esborrem, mentre no creem cap altre fitxer, és possible encara de recuperar el fitxer antic, col·locant a Alpha el nom del fitxer (completant-lo amb es-

pais fins a obtenir un total de 7 caràcters) i a X el número 191 i fent XEQ "SA".
Si ara fem EMDIR, veurem que el fitxer torna a ser-hi, amb els valors que tenia abans.

17

Canviar el nom d'un fitxer és unaltra de les operacions complexes que esdevenen fàcils amb l'ajut del programa "SA".
Per a això, sols cal saber on estant situats els noms dels fitxers a la X-Memory.
El primer està sempre a l'adreça absoluta 191 (decimal).
Si aquest primer fitxer té n registres útils (ho podem comprovar fent EMDIR), com que n'hi ha dos més de capçalera, el següent estarà a l'adreça 191 - n - 2, i així successivament.
Així, per exemple, hindriem:

EMDIR:		Adreça del nom:
"A"	D01S	191
"B"	A02Ø	191-15-2 = 174
"C"	P0Ø9	174-20-2 = 152
"D"	D012	152-9-2 = 141

Si ara volem, per exemple, que el fitxer "C" passi a dir-se "FITXER", escriurem "FITXER " (amb un espai a la dreta, per completar els 7 caràcters) i farem:
152 XEQ "SA".
Amb un EMDIR comprovarem que s'ha produit el canvi de nom.

18

Per saber quants mòduls de memòria hi ha connectats en un determinat moment, podem fer servir la següent rutina (24 octets, i al voltant d'un segon de temps d'execució):

```
01 LBL "NM"   (Nombre Mòduls)
02 "▣---"
03 RCL M
04 X<> c
05 SIZE?
06 X<>Y
07 X<> c
08 CLX
09 64
10 /
11 RTN
```

La línia 02 es codifica:
- en hexa: F3,10,00,00
- en decimal: 243,16,0,0

Només cal fer XEQ "NM" i el nombre de mòduls apareix a X.
(evidentment, un QUADRAM comptarà com a 4 mòduls).

19

Quan executem les funcions ALENG o bé ATOX, el processador comença a buscar el primer caràcter no nul a partir de l'esquerra dels registres alfabètics (5è. octet del registre P).
Com a conseqüència, la funció és més ràpida si la cadena alfabètica és llarga.
A títol d'exemple, donaré els temps d'execució de la funció ALENG (en mil·lisegons) segons el nombre de caràcters a ALPHA:

caràcters:	0	1	2	3	4
temps(ms):	158	161	159	156	152

car.:	5	6	7	8	9	10
temps:	149	145	142	133	130	126

car:	11	12	13	14	15	16
temps:	123	120	116	113	104	100

c.:	17	18	19	20	21	22	23	24
t.:	98	94	91	87	84	76	72	69

(Observem unes discontinuïtats degudes als salts d'un registre a l'altre)
Per tant, quan haguem d'executar diversos ATOX sobre una mateixa tira alfabètica, convé desplaçar-la tant a l'esquerra com puguem, car guanyarem temps.
En veurem un exemple a continuació.

20

Sempre que un programa conté un bucle que s'executa moltes vegades, convé treure de l'interior del bucle les instruccions lentes.
Prenguem per exemple el programa "XH" de conversió hexadecimal a decimal, de la pàgina 60 del llibre d'en Doménech:
A l'interior del bucle hi ha tres instruccions lentes: les línies 04 ATOX, 09 16 i 16 55.
Podem evitar les introduccions numèriques posant-les abans del LBL ØØ i jugant amb l'stack.
Pel que fa a ATOX, com hem vist abans, podem precedir el bucle de RCL M, CLA, STO 0.
El resultat és la següent rutina (he afegit una petita modificació perquè el resultat quedi a X, que sembla més lògic):

```
01 LBL "HD"      16 CF 05
02 RCL M         17 X<>Y
03 CLA           18 X<>F
04 STO 0         19 R↑
05 55            20 X<Y?
06 16            21 ST- Y
07 FRC           22 RDN
08 LBL 01        23 ST+ Z
09 ATOX          24 R↑
10 X=0?          25 R↑
11 GTO 02        26 GTO 01
12 X<>F          27 LBL 02
13 LASTX         28 X<>Y
14 ST* Z         29 RTN
15 CF 04
```

(La línia 07 FRC té per missió col·locar 16 al registre L, bo i deixant 0 al registre X).

Els temps d'execució són:

nombre dígits:	1	3	5	7
programa "XH"	1,6	3,3	5,0	6,8 seg.
» "HD"	1,6	2,7	3,9	5,1 »

Ultra la major rapidesa, el programa "HD" conserva l'estat de tots els flags ("XH" altera els flags 0 a 8)
Cal també remarcar que al principi de "XH" hi manca un CF 08, car si aquest flag està accidentalment encès, i el primer dígit és superior a 9, el resultat és erroni.

21

El següent programa permet d'escriure amb tota facilitat programes sintètics, simplement introduint successivament els codis decimals dels octets, sense haver-nos de preocupar de càlcul d'adreces, ni partir de MEMORY LOST, inclús al mig d'un programa ja existent:

```
01 LBL "EP"   14 FS?C 08   27 DEC
02 RCL b      15 SF 06     28 LBL ØØ
03 STO M      16 FS?C 09   29 DSE X
04 -2         17 SF 07     30 7
05 AROT       18 FS?C 10   31 LBL 01
06 "└------"  19 SF 09     32 STOP
07 RCL M      20 FS?C 11   33 XTOA
08 X<> d      21 SF 10     34 RDN
09 CF 04      22 FS?C 12   35 DSE X
10 CF 05      23 SF 11     36 GTO 01
11 CF 06      24 X<> d     37 RDN
12 FS?C 07    25 E2        38 XEQ "SA"
13 SF 05      26 *         39 TONE 7
                           40 GTO ØØ
```

La línia 06 consisteix en "append 5 nuls".
El mode d'utilització és el següent:
Al lloc on vulguem introduir el programa, escrivim LBL "L", XEQ "EP", +, +, +,... (tants + com octets vulguem introduir, més un mínim de 12).
Llavors tornem al mode RUN, i fem XEQ "L". El programa s'aturarà perquè introduim el codi decimal del 1r octet i fem R/S.
Cada 7 octets sona un TONE que anuncia la creació d'un nou registre de programa, i el comptador que ens anuncia els octets que falten torna a 7.
Cal completar el darrer registre amb zeros.
Quan volguem escriure un programa amb aquesta rutina, pot ser pràctic fer-la servir primer per crear totes les línies sintàtiques seguides, i després, de forma normal, intercalar les línies normals.
(Al final, cal esborrar els signes + que sobren al principi i al final).

RAMON CERERols MACIÀ (145)
Cr. Provença, 513-515, 5è, 4ª
BARCELONA-25
☎ 241·43·07 (en hores d'oficina)
Tots els comentaris, crítiques, suggerències, etc., seran benvinguts.

bits

GENER 1.983

22 Lectura de les ROM-microcodi

La HP-41 conté dos tipus de memòries: ROM i RAM. Entre ambdues hi ha les següents diferències:

a) estructura: la ROM està dividida en 16 blocs de 4K-paraules de 10 bits; la RAM està organitzada en registres de 7 paraules de 8 bits.

b) contingut: la ROM pot contenir instruccions en codi màquina (microcodi) o programes en llenguatge usuari (RPN); la RAM conté registres de dades, programes de l'usuari, assignacions, registres d'estat i fitxers.

c) adreça: en tots dos casos l'adreça ve codificada en dos octets. En el cas de la ROM, el primer nybble indica el bloc, i els altres tres l'octet; pel que fa a la RAM, el primer nybble codifica l'octet, i els altres tres el registre.

d) progressió: les adreces de les successives instruccions d'un programa van en sentit creixent a la ROM, i decreixent a la RAM.

Cal remarcar que la mateixa adreça pot referir-se a la RAM o a la ROM. Internament, hi ha, doncs, un indicador que li permet al processador saber quin tipus de memòria ha de llegir.

Les úniques instruccions directament executables són les escrites en microcodi (codi màquina). Les instruccions en llenguatge usuari criden a rutines en microcodi contingudes a les ROM internes o dels mòduls d'ampliació.

La màquina bàsica conté tres blocs de ROM: els ϕ, 1 i 2. En aquests blocs hi ha el Sistema d'explotació, les instruccions bàsiques i els missatges.

Cada accessori sol incorporar un bloc suplementari de ROM: per exemple: la impressora, el bloc 6; el lector de targetes, el bloc E, etc. Aquestes ROM són escrites en microcodi.

En canvi, els mòduls d'aplicació (matemàtiques, estadística, jocs, etc) acostumen a ser escrits en llenguatge usuari.

Els programes en microcodi no poden ser llegits, ni copiats, pels mètodes normals.

A continuació presento un mètode per a poder llegir les ROM de microcodi, si be amb alguna limitació. (En efecte, cada paraula de microcodi conté 10 bits, dels quals sols decodificarem els 8 finals).

No obstant, veurem que és possible d'obtenir-ne alguna aplicació.

Procés a seguir:

a) Introduïu el programa de salt a la ROM:

```
φ1 LBL "ROM"     15 X<> M
φ2 XEQ "HD"      16 "┣ABCDE"
φ3 CLA           17 9
φ4 256           18 1/X
φ5 STO Z         19 STO N
φ6 /             2φ "┣ABC"
φ7 XTOA          21 CLX
φ8 FRC           22 STO O
φ9 *             23 X<>Y
1φ XTOA          24 ASTO b
11 SF φ5         25 LBL φφ
12 RCL b         26 STOP
13 FC?C φ5       27 RTN
14 GTO φφ        28 END
```

b) Introduïu un programa de decodificació del registre Alpha, per exemple, el següent (sols decodifica els 7 primers octets):

```
φ1 LBL "DCA"     φ5 AROT
φ2 ALENG         φ6 RCL M
φ3 7             φ7 XEQ "DX"
φ4 X<Y?          φ8 END
```

c) Per a una major comoditat, efectueu les següents assignacions:

- programa "ROM" a la tecla 11 ("A"),
- funció SST a la tecla 12 ("B"),
- funció sintètica STO b a la tecla 13 ("C"),
- "byte jumper" a la tecla 14 ("D"),
- programa "DCA" a la tecla 15 ("E").

També podeu substituir tot aquest procés d'assignacions, per l'execució del següent programa:

```
φ1 LBL "RAS"     15 "DCA"
φ2 CLKEYS        14 15
φ3 "ROM"         15 PASN
φ4 11            16 "■■■1X■■"
φ5 PASN          17 193
φ6 "%"           18 XEQ "JA"
φ7 12            19 "■---■A1"
φ8 PASN          2φ 192
φ9 13            21 XEQ "JA"
1φ PASN          22 SF 27
11 14            23 END
12 PASN
```

Nota: Aquests programes fan servir sots-rutines del "BUS" (Bloc Utilitari Sintètic).

El BUS és un conjunt de 14 rutines utilitàries sintètiques, inclòs a la Biblioteca del Club (és un dels últims programes), format a partir de rutines d'anteriors "bits", optimitzades, i altres de noves.

Pels que no disposin del mòdul "X-Functions", disposo també de versions dels programes de lectura de la ROM, que no el fan servir.

Les línies de text sintètiques del programa "RAS", tenen els següents codis:

Línia 16: 247, 240, 145, 124, 33, 4, 8, 17

Línia 19: 247, 240, 0, 0, 0, 241, 65, 49

d) Ara ja ho tenim tot a punt per començar el procés de lectura pròpiament dit.

Suposem que volem llegir els bytes a partir de l'adreça 1C5φ (en hexadecimal).

Farem el següent (en mode USER):

- Pulsem la tecla [A], apareix "_" en mode Alpha. Introduïm l'adreça 1C4F (un byte menys que la que volem llegir) i fem R/S. En quatre segons, el programa s'atura.
- Pulsem la tecla [B]. (Aparentment no passa res)
- Pulsem la tecla [C] (Observarem breument XROM φ5,6φ).
- Pulsem la tecla [D] (Observarem breument XROM φ5,φ1).
- Pulsem la tecla [E]. Després de sis segons, apareix: "φφ,φφ,1φ,φ1,φ3,φ8,φ9"

Com que els dos primers octets són nuls, vol dir que hem llegit 5 octets. (El nombre d'octets llegits depèn del contingut del darrer nybble de l'octet del qual hem partit.)

Així, doncs, ja sabem que:

Adreça	Contingut	
1C5φ	1φ	(hexa)
1C51	φ1	
1C52	φ3	
1C53	φ8	
1C54	φ9	

Per continuar la lectura, podem repetir el procés a partir de l'adreça 1C54.

Com que el segon nybble és un 9, llegirem nou octets, però el programa "DCA" només en decodifica 7.

Repetirem, doncs, la seqüència de tecles [A] [B] [C] [D] [E] i obtindrem, ara:

"φE, φ7, 14, 12, 19, 2φ, φ1"

que correspon a les adreces:

1C55 fins a 1C5B.

Si volem seguir, serà més convenient partir de l'adreça 1C59, car l'adreça 1C5B té el segon nybble 1, i sols llegiríem un octet.

Per comprendre el funcionament del programa, cal recordar com estan codificades les adreces de retorn dels sots-programes als registres a i b (Vegeu, per exemple, la secció 4F del "Synthetic Programming" de Wickes).
En efecte, sempre que el primer nybble de l'adreça de retorn és zero, el retorn és a RAM, en cas contrari a ROM.
(Això provoca una petita complicació en el nostre cas quan volem llegir la ROM interna del bloc Ø, però que està prevista i resolta).
El programa "ROM" el que fa és:
- aturar-se amb un codi de ROM a l'adreça de retorn del registre b.
- el contingut del registre X és el codi de l'adreça on volem anar.

En pulsar la tecla [B] que té assignada la Funció SST, s'executa la següent línia de programa, que és un RTN.
Això ens du a una adreça en ROM, quedant, doncs, encès l'indicador de ROM.
Quan pulsem [C], s'executa STO b, és a dir, col·loquem al registre b l'adreça volguda. Ja som on voliem.
Quan fem [D], el "byte jumper" carrega a Alpha els continguts dels octets següents a l'adreça que havíem indicat.
Finalment [E] ens decodifica aquests octets.

Advertències:
- Si especifiquem una adreça no existent, el programa ens portarà al principi de la RAM.
- Si quan estem a l'interior d'una ROM en microcodi, assagem de passar al mode PRGM, la màquina provarà d'interpretar aquell octet com una línia que formés part d'un programa. Pot ser que ho logri, però també que aparegui "PRIVATE", o fins i tot un "crash".

Aplicacions:
Amb els programes anteriors, podem "tafanejar" les ROM internes. Dues coses fàcils d'interpretar són: els missatges, i les taules d'adreces de funcions.

Missatges:
Els caràcters alfabètics no estan codificats en ASCII dins la ROM, sino en un codi especial. Així, la A és el codi Ø1, la B el codi Ø2, i així successivament, segons la taula que encapçala la següent columna:

2n nybble

1r nybble	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	⊢	a	b	c	d	e	−	⊤	⊥	X	X	X	μ	≤	Σ	◢

Si ara ens fixem en les adreces que havíem decodificat abans, i posem al costat l'equivalent de cada octet en aquesta taula, trobem:

Adreça	Contingut	Caràcter
1C5Ø	1Ø	P
1C51	Ø1	A
1C52	Ø3	C
1C53	Ø8	K
1C54	Ø9	I
1C55	ØE	N
1C56	Ø7	G
1C57	14	T
1C58	12	R
1C59	19	Y
1C5A	2Ø	(espai)
1C5B	Ø1	A

És a dir, llegim el missatge "PACKING", i després comença el "TRY AGAIN".
Hem trobat, doncs, la localització dels missatges. Si aneu provant, trobareu els altres missatges a les següents adreces:

Adreça 1r caràcter	Missatge
1CØF	ALPHA DATA
1C19	DATA ERROR
1C23	MEMORY LOST
1C2E	NONEXISTENT
1C39	NULL
1C3D	PRIVATE
1C44	OUT OF RANGE
1C5Ø	PACKING
1C57	TRY AGAIN
1C6Ø	YES
1C63	NO
1C65	RAM
1C68	ROM

A base de paciència podeu anar buscant els missatges del lector, la impressora, el mòdul X-Functions, etc.

Nota: Hewlett Packard introdueix a vegades modificacions en el SOFTWARE. Podria succeir que en alguna màquina les adreces no fossin exactament les que he indicat.

Taules de funcions:
Tot mòdul (lector, X-Functions, etc.) comença amb una taula de valors que ens indica on comença la rutina de microcodi corresponent a cada funció.
Prenguem, per exemple, el lector de targetes (bloc E). Llegim els continguts de les primeres adreces del bloc. El seu significat és:

Adreça	Contingut	Significat
EØØØ	1E	= 30 (decimal) XROM 3Ø,nn
EØØ1	25	= 37 (nombre de funcions)
EØØ2	ØØ	EØ59 (adreça XROM 30,00)
EØØ3	59	
EØØ4	Ø8	EB4A (adreça XROM 30,01)
EØØ5	4A	
EØØ6	Ø2	E2Ø4 (adreça XROM 30,02)
EØØ7	Ø4	
EØØ8	Ø2	
(...)		(...)
E04C	ØØ	ØØØØ (fi de la taula)
E04D	ØØ	
(...)		

És a dir, el primer octet és el número d'XROM. El segon octet el nombre de funcions (inclòs el nom del mòdul, que es considera la funció ØØ).
A continuació, cada dos octets, formen una adreça que cal sumar al codi del bloc per trobar l'adreça d'inici de la rutina de microcodi.
A partir d'aci podem suposar què passa quan en un programa d'usuari trobem, per exemple, la instrucció MRG.
En realitat, aquesta instrucció està gravada com a XROM 3Ø,Ø1.
Llavors, el processador busca el mòdul XROM 3Ø. Per això llegeix les capçaleres dels blocs connectats fins a trobar-ne un, el primer octet del qual sigui 3Ø (1E en hexadecimal).
El troba al bloc E. Llavors busca en aquest bloc la taula de funcions. La funció XROM 3Ø,Ø1 estarà a l'adreça que li marqui els octets EØØ4 i EØØ5, és a dir EB4A.
Finalment, executa la rutina que comença a EB4A, i que correspon a la funció MRG.

El nom de cada funció està escrit a l'inrevés, a partir de l'adreça anterior a aquella en la qual comença la corresponent rutina de microcodi.
El codi emprat és el descrit abans, però el darrer caràcter del nom de la funció ve marcat perquè se li suma 8Ø (en hexadecimal).
Així, per exemple, si fem una lectura a partir de l'adreça EB45, obtindrem els següents codis:

Adreça	Contingut		
EB46	EØ		
EB47	87	"G"	"MRG" a l'inrevés nom de la funció que comença a EB4A
EB48	12	"R"	
EB49	ØD	"M"	
EB4A	CC	← principi de XROM 3Ø,Ø1 (segons la taula d'adreces que hem vist abans)	
EB4B	2F		
(...)			

Aprofito per fer una observació sobre el programa de lectura. Observareu que no he partit de l'adreça EB46,

perquè el segon nybble del seu contingut és zero, i no llegeix cap octet. Llavors el programa "DCA" decodifica el contingut d'Alpha abans del salt, obtenint un resultat erroni. També es pot donar el cas que els primers octets llegits siguin zero, i els passem per alt.
Per això cal fer sempre diverses lectures de zones que es vagin solapant, per comprovar que no incorrim en aquests errors.

De la manera descrita anteriorment, podem trobar els noms de totes les funcions d'un mòdul. Primer cal buscar-ne l'adreça a la taula de funcions del principi del bloc, i llavors llegir els octets anteriors a aquesta adreça a l'inrevés, fins a trobar-ne un superior a 8Φ (en hexadecimal).

Cal tenir en compte que l'adreça pot variar segons el connector on collaquem el mòdul.
Així, el X-Functions pot ser el bloc 8, A, C, E segons a quin port el connectem.

23 Calendari

I ara, per desintoxicar-nos un xic de "sintètica", alguns programes relatius al calendari que no fan servir ni programació sintètica, ni el mòdul X-Functions.
El primer és un programa que calcula la data del Diumenge de Pasqua per qualsevol any entre el 1900 i el 2099:

```
Φ1 LBL"PASQUA"      29 31
Φ2 FIX Φ            3Φ +
Φ3 CF 29            31 X<>Y
Φ4 CF Φ5            32 4
Φ5 19ΦΦ             33 /
Φ6 -                34 INT
Φ7 STO Y            35 +
Φ8 19               36 7
Φ9 MOD              37 MOD
1Φ RCL X            38 +
11 7                39 25
12 *                4Φ -
13 1                41 CHS
14 +                42 31
15 19               43 X<>Y
16 /                44 X<=Φ?
17 INT              45 SF Φ5
18 CHS              46 X<=Φ?
19 X<>Y             47 +
2Φ 11               48 CLA
21 *                49 ARCL X
22 +                5Φ FS? Φ5
23 4                51 "+ MARC"
24 +                52 FC? Φ5
25 29               53 "+ ABRIL"
26 MOD              54 AVIEW
27 STO T            55 END
28 -                (99 bytes)
```

Per a fer-lo servir, tan sols cal introduir el número de l'any i fer XEQ "PASQUA".
Per d'altres anys, posar el número de l'any i [R/S].
Exemples:

```
1983 XEQ "PASQUA" ⇒ "3 ABRIL"
2000      R/S     ⇒ "23 ABRIL"
2027      R/S     ⇒ "28 MARC"
```

L'algoritme ha estat tret del "Scientific American", de febrer 1.981, pàgina 14. Podeu trobar un procediment similar al "Science & Vie", d'octubre 1.982, pàgina 130.

La següent rutina transforma una data (introduïda en el format: DIA [ENTER↑] MES [ENTER↑] ANY) en un número, corresponent al nombre de dies transcorreguts des d'una data determinada (després en veurem dues aplicacions):

```
Φ1 LBL "ND"      14 LBL ΦΦ
Φ2 CF Φ5         15 3Φ,6
Φ3 X<>Y          16 *
Φ4 3             17 INT
Φ5 X>Y?          18 R↑
Φ6 SF Φ5         19 +
Φ7 SIGN          2Φ X<>Y
Φ8 +             21 365,25
Φ9 FC? Φ5        22 *
1Φ GTO ΦΦ        23 INT
11 12            24 +
12 +             25 END
13 DSE Y         (46 bytes)
```

Una aplicació d'aquesta rutina és el següent programa, que calcula el nombre de dies entre dues dates:

```
Φ1 LBL "DD"      Φ7 PROMPT
Φ2 "DATA 1?"     Φ8 XEQ "ND"
Φ3 PROMPT        Φ9 RCL ΦΦ
Φ4 XEQ "ND"      1Φ -
Φ5 STO ΦΦ        11 END
Φ6 "DATA 2?"     (38 bytes)
```

Exemple: Si volem calcular els dies transcorreguts des del 14 d'abril de 1.951 fins el 2 de gener de 1983, farem:

```
          XEQ "DD"  ⇒ "DATA 1?"
14↑4↑1951   R/S     ⇒ "DATA 2?"
2↑1↑1983    R/S     ⇒  11.586
```

La solució és, doncs, 11.586 dies.

Cal dir que el programa "ND" és vàlid des del primer de març de 1.900 fins el 28 de febrer de 2.100.

Unaltre aplicació és el següent programa que calcula el dia de la setmana corresponent a una data qualsevol (entre els límits indicats):

```
Φ1 LBL "DS"      Φ7 LBL ΦΦ
Φ2 XEQ "ND"      Φ8 "DIVENDRES"
Φ3 7             Φ9 RTN
Φ4 MOD           1Φ LBL Φ1
Φ5 XEQ IND X     11 "DISSABTE"
Φ6 PROMPT        12 RTN
13 LBL Φ2        21 RTN
14 "DIUMENGE"    22 LBL Φ5
15 RTN           23 "DIMECRES"
16 LBL Φ3        24 RTN
17 "DILLUNS"     25 LBL Φ6
18 RTN           26 "DIJOUS"
19 LBL Φ4        27 END
2Φ "DIMARTS"     (91 bytes)
```

Exemple: per saber quin dia de la setmana és el 10 d'abril de 1983:
1Φ↑4↑1983 XEQ "DS" ⇒ "DIUMENGE"

24 Inicialitzacions

En alguns programes de jocs m'he trobat amb el problema d'haver d'inicialitzar un elevat nombre de registres, posem per cas, 50 registres.
Els valors a introduir acostumaven a ser números enters no més grans de 200.
Davant això, les possibles solucions que podem adoptar són:

a) La més usual és incloure dins la rutina d'inicialització, una instrucció RDTA. Això ens ocupa 4 pistes adicionals per les dades, i ens obliga a repetir l'operació de lectura per cada inicialització (En un joc, cada cop que comencem una nova partida).

b) Unaltra possibilitat és incloure a la inicialització tantes instruccions del tipus mm STO nn, com faci falta. (Per exemple: 37, STO 21, 51, STO 22, 189, STO 23, etc.) Aquest mètode, però, presenta l'inconvenient d'augmentar la ocupació de memòria del programa (en el nostre cas, en uns 200 a 250 bytes.)

c) Podem reduir aquest augment d'espai memòria a solament uns 80 bytes amb el següent sistema:
- Introduïm línies de text formades per caràcters, els codis dels quals siguin els valors que cal introduir als successius registres.
- Afegim una sots-rutina del tipus de la següent:

```
LBL ΦΦ
ATOX
X=Φ?
RTN
STO IND ΦΦ
DSE ΦΦ
GTO ΦΦ      (11 bytes)
```

El conjunt pot quedar de la següent manera:

```
5ϕ
STO ϕϕ
"■■■■■■■■■■■■■■■■"
"⊢■■■■■■■■■■"
XEQ ϕϕ
"■■■■■■■■■■■■■■■■"
"⊢■■■■■■■■■■"
XEQ ϕϕ
"■■"
XEQ ϕϕ      (69 bytes)
```

Entre la sots-rutina i les línies d'ací damunt sols ocupen 80 bytes, és a dir, al voltant de la tercera part del mètode b.
A més, el llistat queda més compacte (17 línies contra 100 del mètode b).

Amb algunes variants, es pot adaptar al cas de números fraccionaris o negatius. En general, sempre que el nombre de registres que calgui inicialitzar sigui gran, valdrà la pena emprar aquest sistema.

25 Identificació programes

Quan estem optimitzant un programa, n'anem fent diverses versions que solem conservar escrites en targetes magnètiques.
Llavors pot passar que confonguem les diverses versions d'un mateix programa, i que no trobem quina és la última que havíem redactat.

Les dues rutines que dono a continuació, solucionen aquest problema. La primera d'elles serveix per comprimir la data (dia-mes-any) i el número de versió (1 a 15) a una línia de text de dos caràcters (total: 3 bytes).
Codificació de Versió i Data:

ϕ1	LBL "CVD"	19	PROMPT
ϕ2	"ANY?"	2ϕ	+
ϕ3	PROMPT	21	ENTER↑
ϕ4	198ϕ	22	FIX ϕ
ϕ5	−	23	CF 29
ϕ6	13	24	"242-"
ϕ7	*	25	256
ϕ8	"MES?"	26	MOD
ϕ9	PROMPT	27	ST- Y
1ϕ	+	28	X<>Y
11	32	29	LASTX
12	*	3ϕ	/
13	"DIA?"	31	ARCL X
14	PROMPT	32	"⊢-"
15	+	33	ARCL Y
16	15	34	AVIEW
17	*	35	END
18	"VERSIO?"	(81 bytes)	

Sols és vàlida pels anys 1980 al 1989.
Fem XEQ "CVD" i introduim les dades a mida que el programa les demana.

Al final obtenim un resultat de la forma "242-mmm-nnn".
Això representa els codis dels octets d'una línia de text de dos caràcters (codi 242).
Per qualsevol mètode, introduïm aquesta línia de text com a primera línia del programa que volem identificar, davant inclús de l'etiqueta global que hi pugui haver a la capçalera.
Ara ja podem enregistrar el programa, que restarà identificat, i sols haurem augmentat l'ocupació en tres octets.

Quan volguem identificar el programa, llegim la primera línia (fent GTO "nom del programa", GTO.ϕϕ1, SST en mode RUN). Llavors executem la següent rutina, i obtindrem en clar el nº de versió i la data:
Decodificació de Versió i Data:

ϕ1	LBL "DVD"	16	"⊢-"
ϕ2	RCL M	17	198ϕ
ϕ3	XEQ "BD"	18	+
ϕ4	ENTER↑	19	ARCL X
ϕ5	FIX ϕ	2ϕ	"⊢)"
ϕ6	CF 29	21	PROMPT
ϕ7	"VERSIO "	22	LBL ϕϕ
ϕ8	15	23	MOD
ϕ9	XEQ ϕϕ	24	ARCL X
1ϕ	"⊢("	25	X<> L
11	32	26	/
12	XEQ ϕϕ	27	INT
13	"⊢-"	28	ENTER↑
14	13	29	END
15	XEQ ϕϕ	(74 bytes)	

Es poden utilitzar versions d'aquest programa adaptades a diferents usos. Per exemple, per numerar els programes, o per indicar el número de soci de l'autor, o els accessoris necessaris, etc.

26 Efectes sintètics sense "sintètica"

La instrucció RCLFLAG crea un registre amb la següent estructura:

El coneixement d'aquesta estructura, unit al fet del Bug 7 que tenen totes les HP-41, ens permet de fer una rutina que encèn un flag o bloc de flags qualsevol, entre el Fϕϕ i el F43, sense fer servir cap instrucció sintètica (Normalment, els flags 3ϕ a 43 no poden ser alterats pels mètodes standard).

El programa és el següent:

ϕ1	LBL "FL"	ϕ7	DSE Y
ϕ2	7	ϕ8	GTO ϕϕ
ϕ3	CLA	ϕ9	RCL Z
ϕ4	255	1ϕ	ASTO Y
ϕ5	LBL ϕϕ	11	STOFLAG
ϕ6	XTOA	12	END

Per encendre (posar a set) el flag n, cal fer: n, XEQ "FL".
Per encendre el bloc de flags que comença a mm i acaba a nn, cal fer: mm,nn XEQ "FL".

Podem obtenir dos efectes "sintètics":
a) Catàlegs especials: S'obtenen fent 3ϕ XEQ "FL" i en acabar el programa, R/S.
Varien segons el format en curs (p.ex.: FIX ϕ, FIX 9, etc.)
Aquests catàlegs es poden aturar, fer-los anar pas a pas, etc. com els normals.
b) FIX 1ϕ: Aquest format provoca representacions especials dels números.
Per obtenir-lo, feu: FIX 8, 38, XEQ "FL". Després escriviu el nº 1,23456789ϕ E13, feu ENTER↑ i veureu:
1"#$%&'()
Després feu 2,* , 3,* , i aneu veient què s'obté.
(En realitat els números a X estan normalment, és solament el display que els fa aparèixer diferent).

27 Visualització del codi 255

El codi 255 (hexa: FF) apareix a Alpha com ■, però si aquest codi figura dins un registre que conté una tira alfabètica, ni el codi 255 ni els que el segueixen, són visualitzats.
Exemple: Fem CLA, 64, XTOA, 65, XTOA, 255, XTOA, 66, XTOA, 67, XTOA, ara passem a mode ALPHA i veurem ⁿA■BC. Si fem ASTO X, i sortim del mode ALPHA, veiem només ⁿA. Però en canvi, el registre X conté els cinc codis. En efecte, si esborrem ALPHA amb CLA i fem ARCL X, repuperem tots cinc caràcters.

bit-s
Gener del 1983 - pàgina 4

Ramon Cererols Macià (145)

Provença, 513-515, 5è., 4a.
Barcelona-25
☎ 241-43-07 (hores d'oficina)

bits

FEBRER 1.983

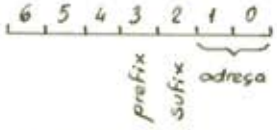
(28) Assignacions i registres interns

El company Jordi Domènech (2) fou el primer en adonar-se que quan realitzem una assignació (mitjançant ASN o PASN), la posició de memòria 2009 (és a dir, el cinquè byte del registre Q) conté el codi de la funció assignada, segons la taula de bytes.
Continuant l'estudi d'aquest fet, he trobat que les posicions 1009 i 0009 (bytes 6è. i 7è. del mateix registre Q) contenen l'adreça on es troba el microcodi de la funció a la ROM.
Podem comprovar-ho llegint els bytes de la ROM anteriors, i hi trobarem el nom de la funció escrit a l'inrevès. (Recordeu la descripció feta al bits (22)).
Així, doncs, podem localitzar l'adreça de qualsevol funció només assignant-la a una tecla qualsevol i llegint el registre Q.
Per exemple, podeu comprovar els següents exemples:

funció:	adreça:	
ASIN	1098	
HMS	1199	
MEAN	11B9	
AON	133C	
MRG	EB4A	
RDTA	E204	
AROT	A26E	(*)
RCLFLAG	A366	(*)
TIME	50EE	
T+X	59FB	

(*) Nota: L'adreça real de les funcions del mòdul X-Functions depèn del port on estigui connectat. (Varia solament el primer dígit).

En resum, l'estructura del registre Q després d'haver executat una assignació, és la següent:



L'octet #4 conté 00 quan es tracta d'una funció bàsica (de les que apareixen al CAT 3), i conté 80 (hexa) quan és una funció externa (de les del CAT 2).
Pel que fa als octets #6 i #5, no he pogut esbrinar què signifiquen.

Aprofitant aquestes propietats de Q, he modificat el meu programa "FB" (Funció a Byte):

```
01 LBL "FB"      15 STO M
02 FIX 0         16 "+="
03 SF 29         17 CLX
04 84            18 4
05 PASN          19 X≠Y?
06 X<> Q         20 ARCL Y
07 X<> M         21 CF 29
08 ,             22 ARCL Z
09 "+ABC"        23 AVIEW
10 STO N         24 CLA
11 CLX           25 84
12 ATOX          26 PASN
13 ATOX          27 END
14 X<> Z                (56 bytes)
```

Per fer-lo servir, només cal introduir a Alpha el nom de la funció de la qual volem conèixer el codi, i fer XEQ "FB".
Per successives funcions, només cal passar el nom i [R/S]:
Exemples:

"LOG" XEQ "FB" ⇒ "LOG=86"
"SIN" [R/S] ⇒ "SIN=89"
"ANUM" [R/S] ⇒ "ANUM=166,66"
"WSTS" [R/S] ⇒ "WSTS=167,138"

Seguint amb l'examen dels registres interns després d'una assignació, el darrer octet del registre + (posició de memòria 000A), conté el codi de la tecla.
Recordem que aquest codi s'obté amb la fórmula:

$$F + 16 \times (C-1) + 8 \times S$$

on F és el número de la fila, C el número de la columna, i S el signe (0 si és positiu, i 1 si és negatiu, és a dir, amb SHIFT).

Tornant al registre Q, quan efectuem l'assignació d'un programa de l'usuari (en lloc d'una funció), els bytes #1 i #0, contenen també l'adreça en RAM del primer byte del LBL assignat.
Amb una modificació, però: el primer nybble (el que conté el nº de byte) apareix multiplicat per 2.

Això ens ofereix la possibilitat de trobar, per programa, l'adreça de qualsevol LBL global, a base de decodificar els darrers bytes de Q, després d'assignar el programa.

Aprofito l'ocasió per aclarir un punt. L'adreça obtinguda per aquest sistema és la real del primer octet del LBL.
En canvi, si fem GTO "alpha" RCL b XEQ "DX", l'adreça que obtenim és la del byte immediatament anterior.
En efecte, si el punter de programa apunta a un determinat byte, la instrucció que s'executarà és la que comença al byte següent.

Un exemple de programa que ens permet de conèixer l'adreça d'una etiqueta global només col·locant el nom a Alpha i fent XEQ "Ad", és el següent:

```
01 LBL "Ad"      12 /
02 84            13 20
03 PASN          14 /
04 RCL Q         15 +
05 XEQ "BD"      16 FIX 1
06 STO Y         17 84
07 40096         18 CLA
08 MOD           19 PASN
09 ST- Y         20 X<>Y
10 X<>Y          21 END
11 LASTX               (43 bytes)
```

L'adreça és obtinguda en el format rrr,b on rrr és el número de registre (en valor decimal absolut) i b el número de byte.
El programa fa servir la sots-rutina "BD" del "BUS".

Queden encara bytes de Q dels quals desconec el significat. Si algú descobreix alguna altra cosa, li agrairé que m'ho faci saber.

(29) Byte-jumper per teclat

Hi ha una assignació sintètica que em sembla que no és massa coneguda, i en canvi ofereix unes grans possibilitats.
Per exemple, permet de modificar diversos octets d'un programa empaquetat sense desfer les compil·lacions ni canviar res més.

En efecte, el byte-jumper clàssic requereix introduir un byte anterior que determina la llargada del salt. Aquest byte suplementari ha de ser després eliminat i esborrem les compil·lacions.
També el byte-grabber crea una línia de text supèrflua que caldrà eliminar, produint-se el mateix problema.

L'assignació de què parlo és la que s'obté (amb un programa d'assignació sintètica, com el "KA", o el "MKX") amb els codis 4,178.

Quan en mode RUN executem la funció així obtinguda, apareix al display: → _ _

La màquina espera ara la introducció de dos dígits. Segons el número que introduïm, el punter de programa saltarà un cert nombre de bytes.

Si volem saltar n bytes, el número que cal introduir serà 16·n.
Si volem saltar m registres i n bytes, el número que cal introduir serà 16·n + m.
Si volem tornar cap enrera, cal precedir el número amb la pulsació de la tecla SHIFT, i apareix ➤ IND __

Si el número introduit és ΦΦ, el programa buscarà un LBL Φ1.

Com funciona aquesta assignació? Si mirem la taula de bytes, veurem que correspon al primer octet de la instrucció GTO Φ1. De la mateixa manera que quan executem RCL, la màquina ens demana RCL __ i el núm. que introduim és utilitzat com a segon octet de la instrucció; igual passa amb ➤ __, solament que en aquest cas, el segón octet és justament el que marca la distància de salt d'un GTO compil·lat.
És per això que si posem ΦΦ, la màquina interpreta que no és compil·lat, i busca el LBL Φ1 pel procediment normal.

A continuació donaré una sèrie d'exemples d'aplicació d'aquesta instrucció sintètica.

1') Creació d'instruccions del tipus: X<> M

	display:
[PRGM]	ΦΦ REG nn
XEQ "X<>" 99	Φ1 X<> 99
[RUN]	(registre X)
➤16, RCL b,	
RTN, STO b	
[PRGM]	ΦΦ REG nn
DEL ΦΦ1, RDN	Φ1 RDN
GTO.ΦΦ1	Φ1 X<> M

(El fet de no introduir octets superflus, ens evita els llargs i repetits PACKINGs).

2') Línies de text sintètiques

per exemple, suposem que volem crear la línia: "ABμDE"

	display:
[PRGM]	ΦΦ REG nn
"ABCDE"	Φ1 "ABCDE
[RUN] ➤48	
RCL b, RTN,	
STO b, [PRGM]	ΦΦ REG nn
DEL ΦΦ1, LBL 11	Φ1 LBL 11
GTO.ΦΦ1	Φ1 "ABμDE

3') Privatització d'un programa

Suposem que volem privatitzar un programa que conté l'etiqueta LBL "alpha".

GTO "alpha"	
[PRGM]	Φ1 LBL "alpha"
GTO .999	nn END
[RUN] ➤32	
RCL b, RTN,	
STO b [PRGM]	ΦΦ REG nn
DEL ΦΦ1, HMS+	Φ1 HMS+
[RUN]	
GTO "alpha"	
[PRGM]	PRIVATE

4t) Desprivatització d'un programa

Suposem que volem desprivatitzar un programa que conté l'etiqueta LBL "alpha".
Amb un CAT 1 ens situem al programa immediatament posterior i llavors fem:

[RUN] GTO.ΦΦ1	
➤ IND 16	
RCL b, RTN,	
STO b, [PRGM]	ΦΦ REG nn
DEL ΦΦ1, LBL Φ8	Φ1 LBL Φ8
[RUN]	
GTO "Alpha"	
[PRGM]	Φ1 LBL "alpha

5è) Canvis en un programa compil·lat sense descompil·lar-lo

Per a poder aplicar el mètode següent, cal que la vostra 41 tingui el Bug 8.
Per a saber-ho, feu la prova següent:

[PRGM] GTO..	ΦΦ REG nn
GTO ΦΦ, BEEP, LBL ΦΦ [RUN]	
(si ara feu R/S, no sonarà el BEEP) GTO.ΦΦ1 R/S	
[PRGM] GTO.ΦΦ1	Φ1 GTO ΦΦ
X<>Y	Φ2 X<>Y
[OFF], [ON]	
GTO.ΦΦ1 R/S	

Si sona el BEEP, teniu el Bug 8, en cas contrari, no.
(Cal haver seguit exactament les instruccions).

Cas que tingueu efectivament el BUG 8, podeu efectuar canvis en determinats octets d'un programa compil·lat, de la següent manera:

Primer de tot, passeu al mode [PRGM] i feu tots els canvis que vulgueu, amb dues precaucions:

a) Quan hagueu de sortir del mode [PRGM], en lloc de pulsar aquesta tecla, feu [OFF], [ON]

b) Feu els canvis fent servir la instrucció ➤__ per no desplaçar els octets.

Un cop haguem acabat tots els canvis, fem: (estem en PRGM):
GTO .999, [OFF], [ON], ➤32, RCL b, RTN, STO b, [PRGM], DEL ΦΦ1, LBL Φ8, [RUN], GTO "alpha".
(De fet, si el programa estava ja empaquetat, tot aquest darrer pas pot ser substituit per un senzill [OFF], [ON]).

Aquesta assignació (➤__) funciona també dins la ROM, però crec que la longitut de salt d'un GTO compil·lat en ROM està codificada de manera diferent, i no he tingut encara temps de trobar la fórmula. (Algú la coneix?).

30 FIX, SCI, ENG sintètics

Alguns haureu observat l'extrany comportament d'algunes instruccions FIX, SCI, ENG, quan llur segon octet no correspon a un digit del Φ al 9.
A què es degut això?
Sembla ser que el funcionament real d'aquestes instruccions és el següent:

FIX, SCI, ENG } 0abc defg

1') els bits del 2n nybble (defg) passen a ocupar el lloc dels flags 36 al 39 (els que determinen el nombre de digits).
2') els flags 40 i 41 són posats a zero.
3') el primer nybble (0abc) executa un OR lògic bit a bit amb el nybble característic del tipus d'instrucció (FIX = 1000, SCI = 0000, ENG = 0100), i el nybble resultant és sumat al format pels flags 40 al 43.
4t) el carry, si n'hi ha, s'arrossega als flags 39 i anteriors (fins al ΦΦ, si cal)

Si repasseu aquest procés per als codis standard, veureu que s'obtenen els resultats habituals. Però veiem ara què podem fer emprant codis no standard.

# digits					FIX ENG	G	RAD	
36	37	38	39	40	41	42	43

1') Podem crear una funció equivalent a ENG n, fent servir una instrucció SCI.
En efecte, n'hi ha prou amb fer SCI 4n (en hexadecimal).

El primer nybble és 0100, i el nybble característic de SCI és 0000; el resultat de l'OR lògic és 0100, i en sumar-lo als flags 40-43, que valen 00XX, obtindrem 01XX. Ha quedat, doncs, encès el flag 41, que determina el format ENG.

2n) Si estem en mode DEG, podem substituir les dues instruccions RAD, ENG n per una sola instrucció SCI 5n (en hexa). (Resseguiu el mateix procés anterior, però ara el primer nybble és 0101). Observeu que si la màquina no estava en mode DEG els resultats canvien.

3r) Podem encendre el flag 3ø amb el següent programa:

```
RCL b
RAD
FIX e
FC? 3ø
STO b
```

Aquest programa funciona així: RAD deixa els flags 36-43 així: XXXX XX01 FIX e comença col·locant 1111 als flags 36-39, deixant el bloc 36-43 així:

1111 XX01

el següent pas:

1111 0001

ara el primer nybble del FIX (0111) fa el OR amb el nybble característic de FIX (1000), quedant 1111 que es suma amb 0001. El resultat és 1111 amb carry=1 que es va arrossegant cap endarrera. El procés es repeteix fins que el carry arriba al flag 3ø (o el que volguem).

4t) La següència:

```
RCL b
RAD
FIX e
STO b
```

va incrementant el número binari format pels flags 00 al 35 (Veureu com l'indicador USER va encenent-se i apagant-se). El cicle es repeteix unes 20 vegades cada segon, és a dir, que partint d'un registre d prèviament posat a zero, el flag 30 s'encén al cap d'un segon i mig. El flag 27 al cap d'uns 13 segons (indicador USER). I així, aniríem seguint, fins que perquè s'encengués el flag øø haurien de passar més de cent anys!

31 Temps d'execució

L'arribada del mòdul de temps permet de medir amb precisió i comoditat els temps d'execució de les diferents instruccions. Això ens permet de comparar següències a fi d'optimitzar-ne la velocitat. Amb tal fi, podeu fer servir el següent programa:

```
ø1 LBL "TF
ø2 CF 21
ø3 4
ø4 X<> c
ø5 CLΣ
ø6 X<> c
ø7 37
ø8 "TF
ø9 CRFLAS
1ø "■■"■■
11 APPCHR
12 "SEQ?
13 AVIEW
14 CLA
15 CF 22
16 LBL ø1
17 STOP
18 FC?C 22
19 GTO ø2
2ø XTOA
21 GTO ø1
22 LBL ø2
23 245
24 ALENG
25 /
26 INT
27 5ø
28 X>Y?
29 X<>Y
3ø STO ø1
31 LBL ø3
32 APPCHR
33 DSE X
34 GTO ø3
35 "■■≠■R
36 APPCHR
37 STOPSW
38 CLX
39 SETSW
4ø ""■
41 ASTO øø
42 "INIC?
43 PROMPT
44 X<> øø
45 STO b
46 LBL "R
47 RCLSW
48 E7
49 *
5ø 25
51 -
52 RCL ø1
53 /
54 FIX 1
55 END
```

(116 bytes)

línia 1ø: 244, 206, ø, 166, 148
línia 35: 245, 166, 153, 29, 241, 82
línia 4ø: 242, 96, 189

El programa medeix el temps d'execució en mil·lisegons d'una següència d'instruccions introduida mitjançant els codis dels octets.

Mode d'utilització: Feu XEQ "TF" Apareix "SEQ?". Introduïu el codi del primer octet i [R/S] Després els successius octets, fins que quan no n'hi ha més, feu solament [R/S]. Després d'uns segons apareix "INIC?", per què podeu introduir els valors inicials de l'stack, d'alpha i dels registres de dades. Quan estigui tot preparat, [R/S] i apareix el temps d'execució en mil·lisegons.
Notes: la següència no pot tenir més de 24 octets. No pot fer servir els registres Røø i Rø1 (són emprats pel programa). El registre X és normalitzat abans de començar la següència.

Funcionament: La rutina crea un fitxer en X-Memory que conté la següència repetida 50 cops (o menys, si és llarga), i que conté un RUNSW al principi i un STOPSW al final.

El programa resulta més exacte i còmode d'utilitzar que el que va aparèixer al Key-Notes. Cal recordar, de totes maneres, que els temps sols són vàlids a títol comparatiu, car varien d'una màquina a l'altra, i segons les condicions de treball (temperatura, voltatge).

Per exemple, a continuació presento alguns dels resultats trobats:

Funcions amb l'stack:

X<>Y	9,5 mseg		R↑	11,1 mseg	
RDN	15,5	"	CLX	9,1	"
ENTER↑	10,7	"	CLST	9,5	"

Moviment de registres:

(Quan hi ha dos diferents temps per la mateixa funció, el més curt és quan no puja l'stack, i el més llarg si ha de pujar).

LASTX	10,2	12,1
RCL intern	16,9	18,7
RCL curt	20,3	22,3
RCL llarg	21,8	23,7
RCL ind intern	29,2	31,3
RCL ind reg.	33,6	35,5
X<> intern	18,1	
X<> registre	22,9	
STO intern	15,5	
STO curt	19,1	
STO llarg	20,3	
STO ind intern	27,9	
STO ind reg.	32,3	

"NOP"s diversos

LBL curt	9,9	mseg
LBL llarg	11,9	"
text ø	11,3	"
CLA	8,9	"

Com veiem, quan no importi esborrar alpha, la funció CLA pot ser el NOP més ràpid per posar després d'un ISG o DSEque es faci servir per incrementar un registre.

Cada nul que hagi quedat en un programa sense empaquetar, tarda 5,3 mseg en ser saltat.

Introduccions numèriques:

El temps és 23,8 + 31,4·n on n és el nombre de digits, amb una excepció: el zero tot sol dura 64,4 segons (contra 55,2 de qualsevol altra xifra).
Per això és preferible el punt al zero, car el punt tarda 57,2 mseg (tot sol).
L'exponent tot sol (E) tarda 49,6 mseg.

Increment d'un registre:

1 +	E +	ISG X LBL ØØ	ISG X LBL ØØ	ISG X ""	ISG X " "	ISG X
81,3	75,7	69,1 (*)	71,7 (#)	69,1 (*)	73,3 (#)	61,9 (#)

(*) Si es produeix el salt
(#) Si no es produeix el salt

Duplicació del registre X:

Seqüència 2 * 89,3 mseg
» ST+ X 31,9 »

Similarment podem comparar les diferents seqüències.

Nota: Per medir, per exemple, la durada d'un RCL nn sense pujar l'escala, medim la seqüència CLX, RCL nn i després en treiem el temps del CLX.

32 Per escriure números

El següent programa transforma un número de Ø a 999 a la seva forma literal correcta.
Cal introduir el número i fer XEQ "NUM" (si la forma literal sobrepassa els 24 caracters, sols veiem els 24 darrers).

```
Ø1 LBL "NUM
Ø2 "CAL ESCRIURE
Ø3 AVIEW
Ø4 SF Ø5
Ø5 CLA
Ø6 INT
Ø7 ABS
Ø8 1 E3
Ø9 MOD
1Ø STO ØØ
11 1ØØ
12 X>Y?
13 GTO 92
14 CF Ø5
15 MOD
16 X<> ØØ
17 LASTX
18 /
19 INT
2Ø SIGN
21 LASTX
22 X=Y?
23 GTO 91
24 XEQ IND X
25 "┣-CENTS
26 GTO 92
27 LBL 91
28 "CENT
29 LBL 92
3Ø 2Ø
31 RCL ØØ
32 X<=Y?
33 GTO 94
34 1Ø
35 MOD
36 STO Ø2
37 RCL ØØ
38 X<>Y
39 -
4Ø STO Ø1
41 XEQ IND Ø1
42 RCL Ø2
43 X=Ø?
44 PROMPT
45 "┣-
46 2Ø
47 RCL Ø1
48 X≠Y?
49 GTO 93
5Ø "┣I-
51 LBL 93
52 XEQ IND Ø2
53 PROMPT
54 LBL 94
55 XEQ IND ØØ
56 PROMPT
57 LBL ØØ
58 FS? Ø5
59 "┣ZERO
6Ø RTN
61 LBL Ø1
62 "┣UN
63 RTN
64 LBL Ø2
65 "┣DOS
66 RTN
67 LBL Ø3
68 "┣TRES
69 RTN
7Ø LBL Ø4
71 "┣QUATRE
72 RTN
73 LBL Ø5
74 "┣CINC
75 RTN
76 LBL Ø6
77 "┣SIS
78 RTN
79 LBL Ø7
8Ø "┣SET
81 RTN
82 LBL Ø8
83 "┣VUIT
84 RTN
85 LBL Ø9
86 "┣NOU
87 RTN
88 LBL 1Ø
89 "┣DEU
9Ø RTN
91 LBL 11
92 "┣ONZE
93 RTN
94 LBL 12
95 "┣DOTZE
96 RTN
97 LBL 13
98 "┣TRETZE
99 RTN
1ØØ LBL 14
1Ø1 "┣CATORZE
1Ø2 RTN
1Ø3 LBL 15
1Ø4 "┣QUINZE
1Ø5 RTN
1Ø6 LBL 16
1Ø7 "┣SETZE
1Ø8 RTN
1Ø9 LBL 17
11Ø "┣DISSET
111 RTN
112 LBL 18
113 "┣DIVUIT
114 RTN
115 LBL 19
116 "┣DINOU
117 RTN
118 LBL 2Ø
119 "┣VINT
12Ø RTN
121 LBL 3Ø
122 "┣TRENTA
123 RTN
124 LBL 4Ø
125 "┣QUARANTA
126 RTN
127 LBL 5Ø
128 "┣CINQUANTA
129 RTN
13Ø LBL 6Ø
131 "┣SEIXANTA
132 RTN
133 LBL 7Ø
134 "┣SETANTA
135 RTN
136 LBL 8Ø
137 "┣VUITANTA
138 RTN
139 LBL 9Ø
14Ø "┣NORANTA
141 END
(395 bytes)
```

Les línies 25 i 28 duen un espai al final. No hi ha cap línia sintètica.
El SIZE ha de ser ≥ ØØ3. El programa funciona en una 41 bàsica, sense cap mòdul auxiliar.

Evidentment, per un altra idioma no n'hi ha prou amb traduir les linies de text. Cal canviar tota l'estructura del programa.

Si algú hi estigués interessat, disposo d'una versió francesa (revisada per un natiu).

33 Intercal·lació d'un registre

Quan calgui intercal·lar un registre en una llista (per exemple, en un programa d'ordenació), podem emprar el següent sistema:

Suposem que tenim els registres de RØØ a R14 i volem col·locar el registre R15 entre el RØ3 i el RØ4 (de fet, el col·loquem a RØ4 i desplacem els antics RØ4 a R14 un lloc a la dreta).
Cal fer:

4,ØØ5Ø11 REGSWAP

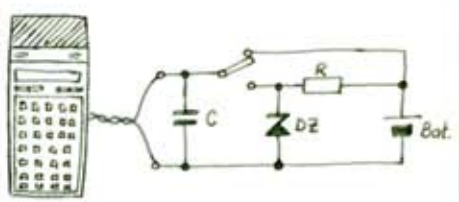
i queda

0, 1, 2, 3, 15, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

34 Senyal extern

Sóc força reaci a posar un soldador dins la meva 41 (l'únic muntatge que hi he fet és una alimentació amb piles externes).

És, però, interessant poguer controlar dispositius externs (rebent senyals de fora, i enviant-los-hi), i he pensat que un mètode de captar un senyal extern, sense cap soldadura interna, podria ser:



El diode zener ha de ser d'un voltatge que permeti el funcionament de la màquina, però encengui el flag 49. (D'aquesta manera, l'estat d'aquest flag ens indica el de l'interruptor extern).
Una advertència: no ho he provat.

Disposo d'una fotocòpia (directa de l'original, i al mateix tamany) del llibre "SYNTHETIC PROGRAMMING ON THE HP-41C".
Si algú hi està interessat, li puc vendre (a preu de fotocòpia).

RAMON CERERÓLS MACIÀ
(soci nº 145)
Provença, 513-515, 5è. 4a.
Barcelona - 25
☎ 241·43·07 (hores d'oficina)

SUPLEMENT ALS bits DE FEBRER '83:

Continuant amb l'estudi de l'assignació sintètica → __ , aquesta permet en mode PRGM la construcció fàcil de GTO's precompilats.
El següent procediment sols és vàlid per màquines que presentin el Bug 8.

A títol d'exemple, crearem un petit (molt petit) programa que conté un GTO precompilat. (El programa no té altra utilitat que la demostració del funcionament d'aquesta instrucció).

Feu el següent:

	display
GTO ..	
[PRGM]	ØØ REG nn
LBL "Td"	Ø1 LBL "Td
→ IND 8Ø	Ø2 GTO Ø1
[OFF]	
[ON] RTN	

Ara el programa està a punt per a ser executat. Fixeu-vos que si el GTO no estigués compilat, en fer [R/S] apareixeria NONEXISTENT, car no hi ha cap LBL Ø1.

Però fixem-nos què passa en realitat, segons el número que hi hagi al registre X:

2	[R/S]	-2
-3	[R/S]	-3
5	[R/S]	-5

És a dir, el programa executa l'invers de la funció ABS, car dóna sempre el valor d'entrada, però amb signe negatiu (el Ø queda invariat).

Com funciona el programa? En fer [R/S] (o bé XEQ "Td"), la primera instrucció és el GTO compilat que ocasiona un salt cap enrera, introduint-se dins el text del LBL, on troba el caràcter "T", que correspon al codi CHS.#1 Després troba la "d", que interpreta com la funció X>Ø?#2.
Si el número és positiu, executa la següent funció, que torna a ser el GTO, tornant-se a repetir el procés fins que X≤Ø, en quin cas se salta el GTO i va a l'END.
D'aquesta manera, en un màxim de dos cicles, s'obté el valor negatiu (o nul).

I ara, un problema: Si assigneu aquest programa a la tecla 12 ([1/x]) i repetiu les proves, obtindreu sempre -1, independentment del valor inicial. Per què? (La solució, als pròxims bits).

Si entreu en mode PRGM, en sortir-ne destruïreu la compilació (Cal fer [OFF], [ON]).

Les següents assignacions funcionen igual que → __ , però amb un display diferent:

codis	display	GTO		codis	display	GTO	
4,178	→ __	Ø1	no admeten alpha	4,186	e± Ī __	Ø9	admeten alpha
4,183	IΣ+ __	Ø6		4,187	Ø≤9 __	1Ø	
4,188	2 __	11		4,191	Q __	14	

Les que admeten entrades alfabètiques produeixen XEQ α, LBL α i LBL α, respectivament.

Una curiositat: El bucle més curt seria el produït per un GTO precompilat produït per → IND 32, que envia a si mateix. (La màquina queda en un bucle infinit).

.END.REG ØØ

Notes:
#1: Mireu la taula de bytes. Byte 54(hexa) = 84(decimal)
#2: Idem. Byte 64(hexa) = 100(decimal)

bits

MARÇ 1.983

35 Números de línia

Fent proves amb l'assignació sintètica →__ (Vegeu el "bits" número (29)) vaig observar un fenòmen que en principi em va semblar anòmal.
Per a reconstruir-lo, feu el següent:

(pulsacions)	(display)
GTO.. [PRGM]	ΦΦ REG nnn
LBL "PROVA"	Φ1 LBL "PROVA
RCL 19	Φ2 RCL 19
[RUN] →16 [PRGM]	Φ2 RCL 19

Però això no era el que jo esperava. En efecte, la instrucció RCL 19 té per codi 144,19. Per tant, en fer el →16 (salt d'un octet cap endavant) ens hauríem de situar a l'octet que conté el codi 19, corresponent a la introducció numèrica 3, i semblava que hauria d'aparèixer Φ2 3 en lloc de Φ2 RCL 19.

Què havia passat? La primera suposició fou que la funció →__ no havia funcionat, però no era així. En efecte, si feu:
[RUN] GTO.ΦΦ2 →16 [R/S]
obtenim com a resultat efectivament 3. (En canvi, GTO.ΦΦ2 [R/S] ens executa RCL 19).

Però fixeu-vos en una cosa: si substituïu el [R/S] per [SST], és a dir, si feu:
GTO.ΦΦ2 →16 [SST], ja no obtenim el 3, si no altra vegada el RCL 19.

Això em va donar la solució. El que passava era el següent:
La instrucció →16 funcionava en tots els casos, però quan després executàvem una instrucció que obligava a calcular el número de línia (com [SST], que el mostra a la pantalla mentre mantenim la tecla pulsada), el microprocessador de la HP-41 comença a buscar des del principi del programa en curs fins arribar al principi de la línia que conté l'adreça actual. No es pararà, doncs, al mig d'una línia.
En canvi, quan fem [R/S], no necessita calcular el número de línia, i, per tant, l'execució comença des de l'adreça actual, sense saber si està al mig d'una línia o no.

Ens queda encara, però, una incògnita a resoldre. Si en lloc de la instrucció →__, fem servir el BJ (byte jumper), el comportament és diferent. En efecte, feu:
GTO.ΦΦ2 [RUN] BJ [SST]
i obtindreu un 3, en lloc del RCL 19. ¿Com és que ara no passa com abans? ¿És que ara no ha de calcular el nº de línia?
La resposta és que no: no ha de calcular el nº de línia, perquè quan hem fet GTO.ΦΦ2, el nº de línia s'ha posat a 2, i com que el byte jumper és simplement una línia de texte, no el canvia, i per tant, en fer [SST], com que el processador ja sap que el nº de línia és 2, no necessita tornar-lo a calcular, i executa el programa des de l'adreça actual.

En canvi, amb la instrucció →__ les coses van diferent, perquè recordem que aquesta assignació sintètica equival en realitat a un GTO.
Ara bé, quan la màquina executa un GTO sap que el número de línia canvia, i per tant, abans d'executar la instrucció amb [SST], el recalcularà, modificant d'aquesta manera l'adreça.

Un cop aclarit, doncs, el fenòmen, cal concretar-lo un xic per poder-ne treure aplicacions pràctiques.

El número de línia es conserva als tres darrers nybbles del registre e (adreces hexa: 1ΦΦF i ΦΦΦF). Segons el contingut d'aquests nybbles, el comportament del processador quan li cal mostrar un número de línia, és el següent:

a) FFF: aquest codi indica que el número de línia s'ha perdut i cal recalcular-lo.

b) ΦΦΦ: aquest codi indica que ens trobem al principi d'un fitxer de programa. El display mostrarà ΦΦ REG nnn, i si introduïm una instrucció, aquesta es gravarà directament a l'adreça actual.

c) qualsevol altre valor: indica que aquest és el número de línia (no cal calcular-lo) i si introduïm una instrucció, aquesta es gravarà després de la instrucció actual.

Quan llegim o executem un programa pas a pas, el nº de línia s'incrementa de 1 cada vegada, però si trobem un GTO o un XEQ o un RTN, el nº de línia és posat a FFF (Comprovarem que el següent [SST], quan executem el programa pas a pas, tardarà més en mostrar-nos la següent instrucció. No quan només el llegim).

Quan executem un programa normalment:
a) si l'execució acaba per una instrucció STOP, RTN, PROMPT, VIEW, AVIEW, OFF, o per una condició d'error, o per interrupció manual mitjançant [R/S] o OFF; el número de línia es posa a FFF.

b) si l'execució acaba per un END, el número de línia es posa a ΦΦΦ.

El número de línia es posa també a ΦΦΦ quan fem manualment GTO.ΦΦΦ o bé RTN, o quan fem un catàleg 1 i el deixem acabar (si no el deixem acabar, es posa a FFF, encara que ens haguem parat a un end).

Fins ara hem vist com funciona normalment el nº de línia, però lògicament, també podem alterar-lo, posant el valor adequat al registre e.

La manera més senzilla de col·locar un determinat nº de línia (per exemple nn) és de fer: [EEX] nn STO e.
Cal, però, tenir en compte dues coses:
a) nn es traduirà d'hexadecimal a decimal.
b) el registre e conté informació de les assignacions de les tecles amb SHIFT. Caldrà doncs, recuperar aquesta informació.

Per no alterar la resta del registre e, podem emprar X<>e que ens salva el contingut de e a X per després poder-lo recuperar amb un nou X<>e, o bé podem valer-nos del mètode de l'exemple següent:

Suposem que després del programa "PROVA", introduïm el programa "SEGUENT", així:

GTO.. [PRGM]	ΦΦ REG nnn
LBL "SEGUENT"	Φ1 LBL "SEGUENT
LBL Φ2	Φ2 LBL Φ2
LBL Φ3	Φ3 LBL Φ3
LBL Φ4	Φ4 LBL Φ4
[RUN]	

Ara, en mode RUN, fem RCL e i després:

[PRGM]	Φ4 LBL Φ4
SST SST	Φ1 LBL "SEGUENT
SST	Φ2 LBL Φ2
[RUN] STO e [PRGM]	Φ4 LBL Φ2

És a dir, tenim en realitat la mateixa línia de programa però amb unaltre número de línia, i amb una curiosa propietat:

BST	Φ3 LBL "SEGUENT
BST	Φ2 END
BST	Φ1 RCL 19

Hem anat a parar al programa anterior (el "prova"), saltant l'END de separació de programes

Això vol dir que el BST considera els END com un LBL α, i que és unicament el número de línia = ΦΦ1 que li indica que ha arribat al principi d'un fitxer de programa.

Una aplicació d'això és la lectura d'un programa privat sense necessitat de desprivatitzar-lo. Suposem que "PROVA" és un programa privat (podem privatitzar-lo pel mètode descrit al bits (29)) i que el segueix el programa "SEGUENT".
Només farem servir l'assignació sintètica X<> e:

GTO "SEGUENT"	
EEX 2, X<>e, [PRGM]	Φ2 LBL "SEGUENT
BST	Φ1 END
SST	Φ1 LBL "PROVA

Ara, mentres anem fent SST, podrem llegir les successives línies del programa "PROVA". Però en quan fem un BST, apareixerà el PRIVATE (perquè la màquina, per saber on comença la línia anterior, segueix fins al següent END, que li dóna l'adreça de l'anterior i al mateix temps li indica que el programa és privat).

Unaltra aplicació del canvi del número de línia amb X<>e és un mètode per crear instruccions del tipus RCL M més senzill que el descrit al bits (29).
El mètode és:

GTO.. [PRGM]	ΦΦ REG nnn
RCL 99	Φ1 RCL 99
[RUN] → 16 CLX	Φ,ΦΦΦΦ
X<>e [PRGM]	ΦΦ REG nnn
RDN	Φ1 RDN
GTO. ΦΦ1	Φ1 RCL M

Evidentment, la línia no cal que sigui la primera, i si ja estem dins d'un programa, no cal el primer pas (GTO.. [PRGM]).

36 Manipulació de Flags

Al bits (26) presentava un programa de manipulació de flags que sols funcionava a les màquines que tenien el Bug 7. A continuació presento una versió millorada que serveix per totes les màquines.

El programa encén o apaga un Flag o bloc de flags qualsevol del FΦΦ al F43.

Per encendre el Flag nn, cal fer nn XEQ "FL".
Per apagar-lo: -nn XEQ "FL"
Per encendre els Flags del mm al nn, cal fer mm,nn XEQ "FL".
Per apagar-los -mm,nn XEQ "FL".

Φ1	LBL "FL	198,0,243,0,70,76
Φ2	"■■■■■■■	247,31,255,255,255,255,255
Φ3	X<Φ?	102
Φ4	"■■-----	247,31,240,0,0,0,0,0
Φ5	RCL M	144,117
Φ6	X<>Y	113
Φ7	STOFLAG	166,109
Φ8	END	192,4,9 (31 bytes)

El programa no pot apagar el Flag ΦΦ tot sol, car -Φ = Φ, però normalment no el farem servir per cancelar solament el FlagΦΦ.

Per les aplicacions, us remeto al bits (26). Solament resumiré:
a) Encesa del Flag 3Φ per obtenció de CATàlegs especials.
b) Format FIX/ENG.
c) Format FIX 1Φ i superiors.
d) Accés als Flags dels perifèrics. Per exemple, el Flag 31, que controla el format de presentació de la data amb el Mòdul de Temps.

37 Solució al problema de febrer

Al suplement dels bits de febrer plantejava un problema. La resposta és la següent:
La primera instrucció: LBL "Td està formada per sis octets.
El primer i el segon contenen el codi del LBL i la distància a l'anterior LBL α o END. El tercer octet és 240+n+1 on n és el nombre de caràcters del nom del LBL. El quart octet (i això és l'important en el nostre cas) conté el codi de la tecla a la qual està assignat el LBL (o bé zero, si no està assignat).
Segueixen després els codis dels caràcters.

En el cas del problema el GTO compilat saltava 5 octets endarrera, és a dir, a l'octet que conté la tecla assignada. Mentre el LBL no estava assignat, aquest octet era nul, i el programa se'l saltava.
En assignar el LBL a la tecla 12 (1/x), aquest octet és ocupat pel codi 17 (decimal) que correspon a la instrucció numèrica 1.
El número 1 ocupa, doncs, el registre X, i el programa produeix el valor -1, independentment del valor inicial.

38 Sortida de senyals

La HP-41 és una màquina aïllada del món exterior per totes bandes menys per una, que és la persona que la manipula.

Mentre aquesta persona és allà, no hi ha problema, però quan falta, la màquina no té possibilitat d'enterar-se de què passa a fora, ni d'enviar-hi cap resposta.

Hi ha ocasions que interessaria que la màquina pogués comunicar-se amb l'exterior directament (alarmes, il·luminació d'escaparates, control de processos, etc).

En aquests casos, podem recórrer a algun bricolatge. De totes maneres, som molts els que no volem (o gosem) posar un soldador dins la màquina.
Per a aquests, en el bits (34) vaig presentar un sistema d'entrada de senyal que no requeria cap soldadura interna, sino que aprofitava l'alimentació exterior.

Faltava un sistema similar per sortida de senyal. Llavors en Farrando em va donar la idea d'aprofitar la diferència de consum en els diferents estats de

la màquina.

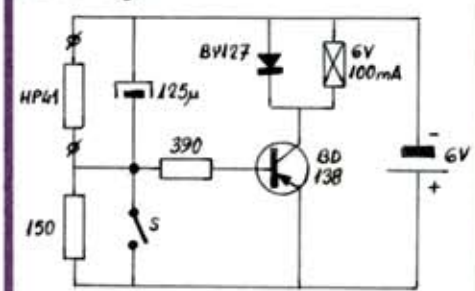
Vaig medir els consums, i vaig obtenir els següents valors:

a) Durant l'execució d'un programa, un catàleg, EMDIR, ALMCAT, SW, i durant la pulsació d'una tecla o l'execució d'una funció o quan, en mode CLOCK, canvia el display: consum de 6 a 10 mA. Aquest consum és pràcticament constant, i sols varia amb el voltatge d'alimentació (a 6V uns 7 mA, a 4,5V uns 10 mA).

b) Amb la màquina ON, però sense executar cap programa, ni executar cap instrucció, al voltant de 0,5 mA (amb el CLOCK al display, es produeix un impuls de 6-10mA cada segon).

c) Amb la màquina OFF, consum de l'ordre dels μA.

El primer muntatge que vaig fer (i que funciona correctament) és el següent:



El circuit pot quedar permanentment connectat, car el seu consum en buit és inferior a 1 μA. Quan no el volem utilitzar, tanquem l'interruptor S.

Funcionament: Quan l'interruptor S està obert, tot el corrent que consumeix la HP-41 ha de passar per la resistència de 150 Ω. Mentre la màquina no executa cap programa ni cap funció, el consum està per sota de 1 mA, i per tant, la caiguda de potencial als extrems de la resistència:

$V = IR = 0{,}001 \cdot 150 = 0{,}15\ V$

insuficient per superar la unió base-emisor del transistor (al voltant de 0,7 V). Per tant, aquest no condueix.

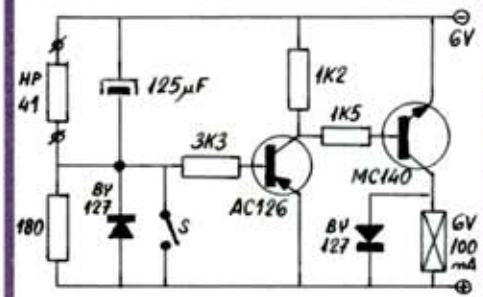
En el moment que un programa comença a executar-se, el consum puja a uns 8-9 mA i la caiguda de potencial esdevé:

$V = IR = 0{,}0085 \cdot 150 = 1{,}3\ V$

Això fa circular un corrent de quasi 2 mA per la base del BD-138, que, multiplicada pel factor d'amplificació del transistor, és suficient per activar el relé. (El diode és per absorbir el corrent de ruptura de la bobina del relé).

Aquest senzill circuit, per bé que funciona, presenta l'inconvenient de produir una important caiguda de voltatge a la resistència de 150 Ω, i per tant, si les piles no estan ben noves, el display indicarà BAT.

Per solucionar això, podem substituir el BD-138 per un transistor de germani (voltatge base-emisor aproximadament igual a 0,25 V) i, a més, shuntar la resistència amb un diode de silici que limiti la caiguda de voltatge a uns 0,7 V, encara que fem servir el lector (que no podria funcionar amb el sistema anterior).
Afegint un transistor, obtenim un circuit més fiable (que encara no he dut a la pràctica):



Potser caldrà reajustar la resistència de 180 Ω a fi de lograr:
a) que no es dispari mai mentre no s'executi cap programa,
b) que es dispari sempre que executi un programa.

Amb la resistència ben ajustada, si feu CLOCK, el relé es dispararà un cop cada segon (permetent de sincronitzar un rellotge extern).

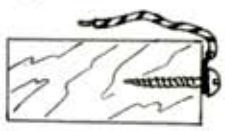
Si voleu activar un circuit a una hora determinada, poseu una alarma que dispari un programa continu a aquella hora, i deixeu la màquina apagada (mentre manipuleu la màquina, tanqueu l'interruptor S, a fi d'evitar dispars del relé).

Un cop l'alarma preparada i la màquina tancada, obriu S. A l'hora pre-determinada, el relé s'activarà, i romandrà així mentre vagi el programa.

Podeu combinar alarmes que engaguin i apaguin el relé, i amb el CLOCK podeu produir trens d'impulsos per activar comptadors externs.

Evidentment, aquest sistema no té tantes possibilitats com el que consisteix en treure un senyal del beeper, però a canvi, no toquem res de la màquina.

Nota: Per qui no tingués muntada l'alimentació externa, un sistema senzill d'obtenir-la, és:
Agafeu una barra rodona de fusta d'uns 11,5 mm ϕ que podeu trobar a les botigues de bricolatge.
Talleu-ne dos troços d'uns 26 mm de llarg. A un dels extrems de cada tros, poseu-hi un petit cargol de cap rodó que fixi un fil elèctric prim, així:



Ara traieu les piles del seu lloc, i en lloc de les dues extremes, col·loqueu-hi les de fusta que heu preparat.

Els dos fils els podeu treure per la petita tapa del lateral (al costat de la tecla SST) i dur-los a un bloc de bateries de 6V, de qualsevol tipus (per exemple, 4 bateries R-20, que permeten quasi un mes de funcionament continu en mode PRGM, o molts mesos en utilització normal d'unes hores al dia).

A més de durar molt més (unes 10 vegades) les piles R-20 són més barates que les que du la màquina.

39 Criptografia

Mentre que els sistemes clàssics de criptografia resulten fàcilment desxifrables, els moderns sistemes de "clau pública" no solen ser fàcilment programables en una màquina petita.

El programa que a continuació llisto és suficientment fiable per a aplicacions normals i alhora curt i senzill.
El seu mode d'utilització és:
a) Amb SIZE ≥ ΦΦS, feu XEQ "CRP"
b) Quan la màquina us demana la clau, introduïu la que prè-

viament hagueu convingut amb el vostre corresponsal, i que ha de ser un número amb 6 xifres decimals $\phi \leq x < 1$ (per exemple, una data o número de telèfon que recordeu fàcilment)

c) Quan apareix: "TEXTE?" introduïu el que voleu codificar (us recomano fer-ho en troços de 12 caràcters, no oblidant els espais). Els caràcters vàlids són de la A a la Z i l'espai.

d) Després de fer R/S, en pocs segons apareix el texte codificat. Si no heu acabat, torneu al punt c).

Per decodificar el texte, quan demana la clau, introduïu-la en negatiu.
Donat el sistema de xifratge que es fa servir, cal introduir el texte sense cap alteració (en el mateix ordre i sense deixar cap lletra ni espai)

```
Φ1  LBL "CRP
Φ2  9821
Φ3  STO Φ2
Φ4  ,211327
Φ5  STO Φ3
Φ6  27
Φ7  STO Φ4
Φ8  "CLAU?
Φ9  PROMPT
1Φ  CF ΦΦ
11  X<Φ?
12  SF ΦΦ
13  ABS
14  STO ΦΦ
15  AON
16  "TEXTE=?
17  LBL Φ1
18  STOP
19  ALENG
2Φ  STO Φ1
21  LBL Φ2
22  ATOX
23  G4
24  X>Y?
25  ENTER↑
26  -
27  RCL ΦΦ
28  RCL Φ2
29  *
3Φ  RCL Φ3
31  +
32  FRC
33  STO ΦΦ
34  RCL Φ4
35  *
36  INT
37  FS? ΦΦ
38  CHS
39  +
4Φ  RCL Φ4
41  MOD
42  32
43  +
44  LASTX
45  X≠Y?
46  +
47  XTOA
48  DSE Φ1
49  GTO Φ2
5Φ  GTO Φ1
51  END
    (92 bytes)
```

La idea del programa és la següent: a partir de la clau introduïda, generem una sèrie pseudo aleatòria de números, de 0 a 26. (27 caràcters autoritzats).

Llavors, desplacem cada caràcter que cal traduir tants llocs dins l'alfabet com el número obtingut. D'aquesta manera la freqüència d'aparició de cada lletra dins el texte xifrat és sempre la mateixa, la qual cosa dificulta la seva decodificació.

Una mateixa lletra, doncs, té diferents traduccions, segons el lloc on és situada.

Perquè pogueu comprovar la correcta introducció del programa, intenteu decodificar el següent texte, que ha estat xifrat amb la clau 0,141593:

FNQQTEWRGHMS
MJIFDDYUDBEF
DUKBVSLQ

40 Flag 54 (Pause)

El Flag 54 normalment està Set durant una Pausa, i Cancel la resta del temps, per això sempre que fem FS? 54 la resposta és NO.
Ara bé, el Flag 54 pot ser també activat amb instruccions sintètiques, per exemple: E2 STO d.

Quan la màquina no està executant un programa i s'encén el Flag 54, el processador interpreta que s'ha iniciat una Pausa. Llavors, espera el temps de la PSE i emprèn l'execució del programa des de l'adreça actual.

Així, doncs, podem executar el programa en el que estem situats, sense pulsar R/S ni fer XEQ "α", simplement fent:

E2 STO d

D'altra banda, si a l'interior d'un programa introduïm la seqüència: E2 STO d, el programa serà executat normalment, però quan acabi, en lloc de parar-se, farà una PSE i continuarà l'execució. (Si haviem acabat a l'END, el programa continua a la línia Φ1).

Cal recordar que E2 STO d esborra la resta de flags.

Com a curiositat, proveu el següent programa:

```
Φ1  LBL "PROVA
Φ2  E2
Φ3  STO d
Φ4  END
```

El programa s'executa, para un moment, torna a executar-se, i així indefinidament (hem obtingut un bucle sense GTO, ni LBL, ni manipulació del registre b).

Ara, modifiquem el programa:

```
Φ1  LBL "PROVA
Φ2  E2
Φ3  STO d
Φ4  RTN
Φ5  "FI
Φ6  AVIEW
Φ7  END
```

Quan l'executem, comprovarem que el RTN es transforma en una PSE (no vull dir que la instrucció canviï, sinó que es comporta com una PSE).
En arribar a l'END, el programa s'atura (perquè cada cop que es reprèn l'execució després d'una PSE, el Flag 54 és posat a zero.

41 Seqüències musicals

```
Φ1  LBL "TNS    198,0,246,0,84,78,83
Φ2  4           20
Φ3  X<> c       206,125
Φ4  CLΣ         112
Φ5  X<> c       206,125
Φ6  "T          241,84
Φ7  37          19,23
Φ8  CRFLAS      166,74
Φ9  CF 22       169,22
1Φ  LBL Φ1      2
11  "TONE?      245,84,79,78,69,63
12  PROMPT      142
13  FC?C 22     171,22
14  GTO Φ3      180,18
15  LBL Φ2      3
16  "■          241,159
17  XTOA        166,111
18  X<Φ?        102
19  "■■P■       244,17,16,80,117
2Φ  APPCHR      166,67
21  GTO Φ1      178,227
22  LBL Φ3      4
23  "■■         242,145,124
24  APPCHR      166,67
25  LBL Φ4      5
26  ""■         242,96,189
27  RCL M       144,117
28  X<> b       206,124
29  "REP?       244,82,69,80,63
3Φ  PROMPT      142
31  FC? 22      173,22
32  GTO Φ4      181,194
33  "T          241,84
34  CLFL        166,72
35  GTO Φ2      179,182
36  END         194,11,9      (81 bytes)
```

Aquest programa permet d'executar una seqüència de fins a 126 tones sintètics, seguits o amb pauses intercalades (cada pausa ocupa l'espai de 2 tones) i pot servir doncs d'ajut per a la preparació de combinacions de tones per programes de jocs, per exemple.

El mode d'utilització és:

a) XEQ "TNS", apareix "TONE?"

b) Introduïu el número del TONE (de 0 a 127), o bé, si voleu una pausa, introduïu -1 (cada pausa dura 3 dècimes de segon), i pulseu R/S

c) Torna a aparèixer "TONE?" Si no heu acabat d'introduir la seqüència de Tones, torneu al punt b). Si heu acabat, només pulseu R/S, i s'executarà la seqüència.

d) Al final, apareix "REP?". Si voleu tornar a sentir la seqüència, feu R/S. Si voleu començar la introducció d'una nova seqüència, poseu el primer Tone i R/S i continueu al punt c).

Com a exemples, proveu les següents seqüències:

1) Toccata de Bach:
1, 0, 33, -1, 0, 15, 94, 109, 124, 13.

2) Alarma:
57, 72, 57, 72, 57, 72, 57, 72, -1, 57, 72, 57, 72, 57, 72, 57, 72, -1, 57, 72, 57, 72, 57, 72, 57, 72.

3) Mozart:
96, -1, 109, 96, -1, 109, 0, 109, 0, 2, 3, -1, 66, 80, 80, 66, 83.

4) Grills:
57, 89, 57, 89, 57, 89, 57, 89, -1, -1, -1, 57, 89, 57, 89, 57, 89, 57, 89, -1, -1, -1, 57, 89, 57, 89, 57, 89, 57, 89.

(Han estat tretes del "Calculator Tips & Rutines")

42 Accés a la ROM

Per col·locar la màquina dins la memòria morta, si disposem d'un mòdul d'aplicació, només cal fer GTO "α", on "α" sigui el nom d'un LBL global qualsevol del mòdul d'aplicació.

Si no disposem de cap mòdul d'aplicació, el mètode que se sol emprar és posar un codi adequat a l'adreça de retorn (4t. i 5è. bytes del registre b). Aquest codi és tal que el primer nybble sigui 1 o 2 (si és 0, el retorn és a RAM).

Un cop fet això, s'executa un RTN, i es produeix el salt a la ROM (aquest mètode és l'emprat al bits (22)).

Ara bé, aquest RTN ha de ser executat manualment amb SST, perquè sinó, l'execució continuaria dins la ROM, amb resultats imprevisibles.

Hi ha una manera d'evitar la necessitat d'aquesta pulsació manual: és fer el salt a una adreça de la ROM que contingui un codi corresponent a una instrucció que aturi l'execució (p. ex.: codi 132 = STOP).

El següent programa aprofita aquesta possibilitat, i la seva execució ens du directament a la ROM.

01	LBL "ROM	198,0,244,0,82,79,77
02	"■■■P↗	245,133,17,21,80,5
03	ASTO b	154,124
04	END	194,2,9 (18 bytes)

Podrem seguir-ne millor el funcionament si escrivim en hexadecimal el registre b tal com queda en executar ASTO b:

10, 00, 85, 11, 15, 50, 05

Després de la línia 03 ASTO b, el contingut de b és, doncs, l'indicat, per tant, l'execució continuarà després del byte 5 del registre 005 (és a dir, al byte 4 del registre M).

Ara bé, la línia 02 ha introduit en aquest byte el codi 85 (hexa) que correspon a un RTN. Això fa que l'execució es traslladi a l'adreça de retorn (després del byte 1115 de la ROM) és a dir, al byte 1116, que conté precisament el codi 84 (hexa) que correspon a la instrucció STOP, que atura el programa, amb el punter dins la ROM.

Si prèviament havíem col·locat al registre X l'adreça d'on volem anar, com que el programa ROM no altera l'stack, fent un STO b anem directament a aquella adreça.

Això permet de simplificar el programa d'accés a la ROM del bits (22).

Un bug del mòdul X-Functions, encara no explorat, és el que presenten les funcions GETP i SAVEP quan el punter està a la ROM.

Si després d'executar XEQ "ROM" intentem salvar un programa amb SAVEP, obtenim a vegades NO ROOM, malgrat haver-hi espai sobrat. A vegades salvem uns codis que no tenen res a veure amb el programa que volíem salvar.

El GETP en aquestes condicions a vegades va, a vegades produeix un Crash, a vegades surt PACKING, TRY AGAIN.

(Potser els codis traspassats corresponen a la ROM).

43 ↗__ a la ROM

Al bits (29) em preguntava com funcionava la funció sintètica ↗__ dins la ROM.

La resposta és senzilla: el número que introduïm després de ↗__ és directament el nombre de bytes que saltem.

Així, si partim de l'adreça E1G2 i fem ↗01 anem a E1G1, i si ara fèssim ↗IND 01 tornem al mateix lloc.

Per tant, els GTO compilats curts dins els mòduls d'aplicació, admeten salts fins a 127 bytes, en lloc de 112 com en RAM.

També cal observar que mentre a la RAM un byte jumper i un ↗nn van en el mateix sentit, en la ROM van en sentit contrari.

44 Programes de jocs

Hi ha jocs per als quals existeix un algoritme que permet a un dels dos jugadors guanyar amb tota seguretat. En aquests casos, aquest algoritme és el que ha de seguir el programa.

Normalment, però, aquest algoritme, o no existeix, o el joc és tant complexe que encara no s'ha trobat.

Llavors, hi ha diferents consideracions tàctiques i estratègiques que ens permeten avaluar l'estat del joc en cada moment.

Aquestes consideracions, fruit normalment de l'experiència, les traduim en una funció d'avaluació, que dóna, per cada possible jugada vàlida, un index del nostre avantatge.

Ara bé, si resulta relativament fàcil trobar qualitativament quins factors han d'intervenir en el càlcul d'aquesta funció, no ho resulta tant valorar quantitativament quin pes ha de tenir cada un d'ells en el conjunt de la funció.

Llavors ens veiem obligats a provar el programa moltes vegades, i, a la vista dels resultats, afinar els paràmetres de la funció.

Aquesta tasca resulta lenta, i pot ser realitzada més còmodament i amb més eficàcia, així:

Fem un programa "mestre" que vagi fent enfrontar entre elles diverses versions del programa (canviant en cada versió els valors dels paràmetres de la funció d'avaluació, de forma ordenada, o, fins i tot aleatòria).

Després de cada grup de partides, seleccionar (el mateix programa "mestre") aquell programa que hagi guanyat, i seguir "l'evolució" a partir d'ell.

Així podem deixar que el nostre programa vagi millorant-se per ell mateix. (vagi "aprenent").

El procés té una certa similitud amb el que segueix l'evolució en els sers vius, i podem arribar a pensar en programes que, establertes simplement les regles d'un joc, vagin aprenent a jugar-hi per ells sols.

45 Decimal a hexadecimal

Normalment els números decimals que hem de passar a hexadecimal són enters inferiors a 65.536.
Aquest programa realitza la conversió. Per servir-nos-en, només cal introduir el nº en decimal i fer XEQ "DH". En uns 3 segons i mig apareixen quatre dígits hexadecimals que són la traducció del número decimal (els primers poden ser zeros).

Si volem que el programa vagi més ràpid, fem SF 10. Llavors l'execució és en un segon i mig, però el resultat s'obté en la notació "natural", és a dir:

A	B	C	D	E	F
:	7	∠	=	⊃	?

01	LBL "DH	198,0,243,0,68,72
02	ENTER↑	131
03	"A	241,65
04	256	18,21,22
05	/	67
06	XTOA	166,111
07	X<> L	206,116
08	MOD	75
09	XTOA	166,111
10	"├--	243,127,0,0
11	FIX 9	156,9
12	ARCL M	155,117
13	E	27
14	CHS	84
15	AROT	166,70
16	ASHF	136
17	ASHF	136
18	FS? 10	172,10
19	PROMPT	142
20	"├ ₹	242,127,1
21	58	21,24
22	LBL 01	2
23	SIGN	122
24	ATOX	166,71
25	X=Y?	120
26	PROMPT	142
27	LASTX	118
28	X<>Y	113
29	X<Y?	68
30	GTO 00	177,32
31	7	23
32	+	64
33	LBL 00	1
34	XTOA	166,111
35	X<>Y	113
36	GTO 01	178,210
37	END	202,8,9 (64 bytes)

46 Funcions trigonomètriques

Les funcions trigonomètriques son relativament lentes. Per això, en programes que les facin servir molt (astronomia, geometria, etc.) convé mirar de treure'n el profit més gran possible.

Una consideració a tenir en compte és que funcionen més ràpidament en mode RAD que en mode DEG.

Unaltre fet és que si d'un mateix valor ens fa falta al mateix temps el SIN i el COS, és millor fer 1 P-R que ens dóna tots dos valors alhora (als registres X i Y).

Per comprovar l'eficàcia d'aquestes consideracions, compararem els temps d'execució per calcular: (SIN X + COS X).

Per a això, podem emprar 2 programes:

A) SIN, LASTX, COS, +

B) E, P-R, +

(Observem de pas que el segon programa és un octet més curt).

Els temps d'execució són variables. Per això els he medit amb el TIMER repetint cada versió 50 vegades amb diferents valors. Els resultats mitjos, en segons, són:

		DEG	RAD
programa	A)	0,97	0,81
"	B)	0,60	0,52

47 Visualització d'indicadors

Si en el curs d'un programa, utilitzem alguna de les funcions: SF, CF, FS?C, FC?C, DEG, RAD, GRAD, AON, AOFF, per canviar l'estat d'algun dels Flags que apareixen al display, aquesta modificació és instantàniament traslladada al display.

Però en canvi, si algun d'aquests mateixos Flags és canviat per modificació directa del registre d (p.ex.: STO d, X<> d), llavors el display no és actualitzat fins que no es produeix alguna funció que l'actualitzi.

Per a comprovar-ho, escriviu el programa següent:

01	LBL "FLAGS	09	DSE X
02	SF 00	10	GTO 00
03	SF 27	11	SF 01
04	GRAD	12	100
05	CLX	13	LBL 01
06	X<> d	14	DSE X
07	100	15	GTO 01
08	LBL 00	16	END

Les línies 02, 03 i 04 encenen una sèrie d'indicadors a la pantalla. Les línies 05 i 06 apaguen els Flags, però no els indicadors. Les línies 07 a 10 són una pausa per veure-ho. En produir-se SF 01, el display és actualitzat per complet.

bits

MAIG 1983

51 Joc del Master - Mind

Començarem amb un programa dels que podríem anomenar, emprant un terme actual, "opció zero", és a dir, executable en una HP-41C standard, sense cap mòdul ni accessori adicional, i amb solament les instruccions de base (sense programació sintètica).

El programa permet de jugar al "Master-mind", quan no disposem de la plaqueta del joc i un company que faci el paper de "codificador".

En efecte, la màquina tria a l'atzar una combinació, i calcula per cada assaig el nombre de peces ben i mal col·locades.

Els diferents colors han estat substituits pels números de 1 a 6. Es tracta, doncs, d'encertar un número de quatre xifres de 1 a 6 (poden ser repetides).

Per cada assaig, introduim un número de quatre xifres i obtenim el número d'ordre de l'assaig, el número assajat, i el nombre de peces encertades ben i mal col·locades.

El programa és el següent:

```
01 LBL "MIND
02 FIX 0
03 CF 29
04 CLX
05 STO 05
06 1,0004
07 STO 12
08 LBL 01
09 RCL 00
10 9821
11 *
12 ,211327
13 +
14 FRC
15 STO 00
16 6
17 *
18 1
19 +
20 INT
21 STO IND 12
22 ISG 12
23 GTO 01
24 "ASSAIG?
25 LBL 00
26 PROMPT
27 STO 15
28 1 E4
29 /
30 STO 14
31 6,0009
32 STO 13
33 LBL 02
34 RCL 14
35 10
36 *
37 INT
38 LASTX
39 FRC
40 STO 14
41 RDN
42 STO IND 13
43 ISG 13
44 GTO 02
45 0
46 STO 10
47 STO 11
48 1
49 ST+ 05
50 1,0004
51 STO 12
52 LBL 03
53 RCL IND 12
54 ABS
55 STO IND 12
56 ISG 12
57 GTO 03
58 1,0004
59 STO 12
60 LBL 04
61 RCL 12
62 5
63 +
64 STO 13
65 RCL IND 12
66 RCL IND 13
67 X≠Y?
68 GTO 05
69 -1
70 ST- 10
71 ST* IND 12
72 ST* IND 13
73 LBL 05
74 ISG 12
75 GTO 04
76 1,0004
77 STO 12
78 LBL 06
79 RCL IND 12
80 X<0?
81 GTO 09
82 6,0009
83 STO 13
84 LBL 07
85 RCL IND 12
86 RCL IND 13
87 X≠Y?
88 GTO 08
89 -1
90 ST- 11
91 ST* IND 12
92 ST* IND 13
93 GTO 09
94 LBL 08
95 ISG 13
96 GTO 07
97 LBL 09
98 ISG 12
99 GTO 06
100 CLA
101 ARCL 05
102 "⊢:
103 ARCL 15
104 "⊢
105 ARCL 10
106 "⊢B-
107 ARCL 11
108 "⊢M
109 GTO 00
110 END
```

El programa ocupa 213 bytes i requereix un SIZE mínim de 016 registres.

Abans de començar la primera partida, cal introduir una llavor aleatòria (un nº qualsevol $0 \le x < 1$, amb unes sis xifres decimals) al registre R00. Per exemple, si volem reproduir exactament els resultats que es donen a continuació, fem: 0,140451 STO 00.

Llavors fem XEQ "MIND". Apareix "ASSAIG?". Provem, per exemple:

1345 [R/S] "1: 1345 0B-2M"

La qual cosa significa: primer assaig: 1345. Zero números correctament col·locats, i dos encertats però mal col·locats.

Podem seguir provant:

2456 [R/S] "2: 2456 1B-2M"

El joc s'acaba quan encertem les quatre peces:

4252 [R/S] "3: 4252 4B-0M"

Per començar una nova partida, fem XEQ "MIND" (no cal canviar R00. El programa se n'encarrega).

52 Sincronització del timer.

La manera standard de sincronitzar el rellotge del Mòdul de Temps és incòmoda i força inexacte.

En efecte, l'execució de les funcions CORRECT i SETIME té lloc en deixar anar la tecla, no en apretar-la. A més, a partir d'aquest moment, li cal buscar als diferents catàlegs fins a trobar la funció. (Si la funció és assignada, busca dins els registres d'assignació).

A fi d'eliminar aquests inconvenients, i per tant millorar en gran mida la precisió en la sincronització, hom pot emprar el següent programa:

```
01 LBL "SINCR
02 SIZE?
03 E
04 X>Y?
05 PSIZE
06 STOPSW
07 CLX
08 SETSW
09 TIME
10 RUNSW
11 SW
12 RCL 00
13 HMS+
14 E-6
15 HMS+
16 FIX 6
17 END
   (37 bytes)
```

Exemple d'utilització: suposem que volem sincronitzar el Timer amb els senyals horaris de Ràdio Nacional d'Espanya de les 14:00.

Un minut abans aproximadament (el temps exacte no té importància), fem XEQ "SINCR". La màquina es posarà en mode cronòmetre.

En sentir el senyal horari, pulsem la tecla [ENTER↑] (tant exacte com puguem). Llavors, quan valguem, fem SHIFT [←]. Ens apareix l'hora que marcava el timer en pulsar la tecla [ENTER↑], és a dir, a les 14:00. Ara, serà fàcil corregir la diferència amb la funció T+X (en signe contrari de la que hi hagi).

Aquest mètode permet d'augmentar la precisió gràcies a diverses característiques:

a) la sincronització es produeix en prémer la tecla, no en deixar-la,
b) en estar el teclat redissenyat per la funció SW, la màquina no ha de perdre temps buscant caràlegs o registres d'assignació,
c) es pot encara reduir l'error humà amb tres o quatre sincronitzacions successives i efectuant la correcció de l'error mitjà, segons l'exemple següent:

Suposem que fem tres proves:

A les 14h. marca 14,0000044
" " 15h " 15,0000051
" " 16h " 16,0000048

L'error mig és $\frac{44+51+48}{3} : 47{,}7$

Llavors fem:
-0,0000048 XEQ "T+X"
i el rellotge haurà quedat sincronitzat. (Recordem però, que el factor d'ajust no queda afectat i en cas necessari, ens caldrà ajustar-lo amb SETAF).

Les línies 14 i 15 corregeixen el temps entre la 09 i la 10.

53 Missatges llargs

Si passem en mode Alpha quan aquest conté un missatge de més de 12 caràcters, la resta va desfilant, un a un.

Si el missatge és llarg, la desfilada pot resultar lenta (fins a 7 segons per a 24 caràcters).

En aquest cas, mentre el missatge desfila, podem pulsar qualsevol tecla (fins i tot ON i ←) i apareixeran de cop els 12 caràcters finals, sense haver d'esperar.

54 Factorial de nº grans

La funció FACT no podem utilitzar-la per valors més grans de 69. El següent programa no té aquest límit, encara que és a costa del temps d'execució.

```
01 LBL "FAC
02 CLA
03 09
04 STO N
05 10↑X
06 X<>Y
07 1
08 X>Y?
09 GTO 02
10 X<>Y
11 LBL 00
12 ST* Y
13 X<> Z
14 X<=Y?
15 XEQ 01
16 X<> Z
17 DSE X
18 GTO 00
19 X<> M
20 X<>Y
21 ENTER↑
22 LOG
23 INT
24 ST+ Z
25 10↑X
26 /
27 LBL 02
28 RCL Y
29 X≠0?
30 LOG
31 INT
32 9
33 −
34 FIX IND X
35 RDN
36 CLA
37 ARCL X
38 "."
39 FIX 0
40 CF 29
41 ARCL Y
42 AVIEW
43 FIX 9
44 BEEP
45 RTN
46 LBL 01
47 ST/ Y
48 RCL N
49 ST+ M
50 RDN
51 RTN
52 END
```

84 bytes SIZE 000

Per als reacis a la programació sintètica, podeu canviar les línies següents:

```
02 CLRG
04 STO 02
19 X<> 01
48 RCL 02
49 ST+ 01
```

} llavors cal un SIZE 003

El resultat és obtingut en notació científica, però amb exponents de nombre de dígits variable segons les necessitats. El nombre de decimals de la mantissa s'ajusta per obtenir un conjunt de 12 caràcters. (La mantissa resta a X i pot ser vista fent ←).

Exemples d'utilització:

6 XEQ "FAC" ⇒ 7,200000000 2
(en ~3 segons)

85 XEQ "FAC" ⇒ 2,81710141 128
(en ~22 segons)

217 XEQ "FAC" ⇒ 2,1744341 414
(en ~51 segons)
fent FIX 9 ⇒ 2,174434126 (que és la mantissa amb 10 xifres significatives).

55 Othel·lo millorat

Arrel de la publicació del programa d'Othel·lo ràpid, alguns m'han demanat per la versió millorada (tant en qualitat de joc com en comoditat d'ús). El programa és el següent:

```
01 LBL "OTG
02 CF 21
03 4
04 X<> c
05 CLΣ
06 X<> c
07 "OT
08 82
09 CRFLD
10 1008
11 PSIZE
12 CLRG
13 99
14 STO 00
15 "87'26885?/:48%
16 "-88.$8#+8
17 XEQ 17
18 "8*"839)(82Bt|!
19 "Ø8
20 XEQ 17
21 50
22 STO 00
23 "d8OL
24 XEQ 17
25 38
26 STO 00
27 "cb]ZVSNM
28 XEQ 17
29 8
30 STO 00
31 "X88888888
32 LBL 17
33 ATOX
34 X=0?
35 RTN
36 9
37 −
38 STO IND 00
39 DSE 00
40 GTO 17
41 ,3
42 "CFXC
43 XEQ 18
44 ,32
45 "L?BGW\_b
46 XEQ 18
47 ,36
48 "DEJMQTYZ
49 XEQ 18
50 ,4
51 "=>INPU"a
52 XEQ 18
53 ,45
54 ",Pfc
55 XEQ 18
56 51
57 X<>F
58 SF 29
59 FIX 0
60 "C4: P-J?
61 AON
62 STOP
63 AOFF
64 ATOX
65 80
66 X=Y?
67 GTO 16
68 RCL 01
69 STO 83
70 STO 75
71 CHS
72 STO 82
73 STO 76
74 LBL 16
75 CLA
76 LBL 00
77 RCL 01
78 STO 00
79 CF 08
80 "TU?
81 -349
82 CF 23
83 AON
84 PROMPT
85 AOFF
86 FC? 23
87 GTO 14
88 ATOX
89 +
90 ATOX
91 RCL 05
92 *
93 +
94 XEQ 07
95 CLX
96 X<>F
97 RCL 01
98 −
99 X<>F
100 LBL 14
101 CLX
102 SEEKPT
103 19,1
104 SAVERX
105 19,05
106 STO 11
107 CLX
108 STO 13
109 STO 16
110 LBL 01
111 RCL IND 11
112 X≠0?
113 XEQ 02
114 ISG 11
115 GTO 01
116 RCL 16
117 X=0?
118 GTO 15
119 "JO
120 3
121 −
122 STO Y
123 RCL 05
124 MOD
125 65
126 +
```

```
127 XTOA
128 X<>Y
129 RCL Ø5
13Ø /
131 INT
132 RCL Ø5
133 -
134 ARCL X
135 AVIEW
136 BEEP
137 CLX
138 X<>F
139 RCL Ø1
14Ø -
141 X<>F
142 RCL Ø2
143 STO ØØ
144 CF Ø8
145 RCL 16
146 XEQ Ø7
147 GTO ØØ
148 LBL 15
149 "PASSO,
15Ø AVIEW
151 BEEP
152 GTO ØØ
153 LBL Ø2
154 STO 18
155 RCL 13
156 RCL IND Y
157 FRC
158 E2
159 *
16Ø STO 15
161 X<=Y?
162 GTO Ø5
163 STO 14
164 RCL Ø2
165 STO ØØ
166 CF Ø8
167 R↑
168 XEQ Ø7
169 FC? Ø9
17Ø RTN
171 RCL 15
172 FC? Ø5
173 STO 14
174 RCL 13
175 RCL 14
176 X<=Y?
177 GTO Ø6
178 RCL 11
179 INT
18Ø RCL Ø1
181 -
182 E3
183 /
184 19
185 +
186 STO 17
187 RCL Ø1
188 STO ØØ
189 SF Ø8
19Ø LBL Ø3
191 RCL IND 17
192 X=Ø?
193 GTO Ø4
194 XEQ Ø7
195 FC? Ø9
196 GTO Ø4
197 RCL IND Ø9
198 FRC
199 E2
2ØØ *
2Ø1 RCL 15
2Ø2 -
2Ø3 ST- 14
2Ø4 STO 17
2Ø5 LBL Ø4
2Ø6 ISG 17
2Ø7 GTO Ø3
2Ø8 RCL 13
2Ø9 RCL 14
21Ø X<=Y?
211 GTO Ø6
212 STO 13
213 RCL 18
214 STO 16
215 LBL Ø6
216 CLX
217 SEEKPT
218 19,1
219 GETRX
22Ø RTN
221 LBL Ø5
222 STO 11
223 RTN
224 LBL Ø7
225 CF Ø9
226 STO Ø9
227 RCL Ø7
228 STO 12
229 LBL Ø8
23Ø RCL Ø9
231 RCL IND 12
232 +
233 STO 1Ø
234 RCL IND X
235 RCL ØØ
236 X≠Y?
237 GTO 11
238 LBL Ø9
239 LASTX
24Ø ST+ 1Ø
241 RCL IND 1Ø
242 RCL Z
243 X=Y?
244 GTO Ø9
245 CHS
246 X≠Y?
247 GTO 11
248 SF Ø9
249 FS? Ø8
25Ø RTN
251 RCL IND 12
252 ST- 1Ø
253 LBL 1Ø
254 RCL Ø2
255 ST* IND 1Ø
256 ST+ 14
257 RCL IND 12
258 ST- 1Ø
259 RCL 1Ø
26Ø RCL Ø9
261 X≠Y?
262 GTO 1Ø
263 LBL 11
264 DSE 12
265 GTO Ø8
266 FC? Ø9
267 RTN
268 RCL ØØ
269 CHS
27Ø X<> IND Ø9
271 ,
272 STO IND Y
273 RCL Ø7
274 STO 12
275 LBL 12
276 RCL Ø9
277 RCL IND 12
278 +
279 RCL IND X
28Ø X=Ø?
281 GTO 13
282 SIGN
283 LASTX
284 X=Y?
285 GTO 13
286 RCL Z
287 STO IND Y
288 LBL 13
289 DSE 12
29Ø GTO 12
291 RTN
292 LBL 18
293 ATOX
294 X=Ø?
295 RTN
296 X<>Y
297 ST+ IND Y
298 GTO 18
299 END
(596 bytes)
```

Les línies sintètiques tenen els següents codis:

Línia 15:
255,31,55,39,38,54,3Ø,9,53,59,
47,46,58,52,9,37

Línia 16:
25Ø,127,45,8,1Ø,44,36,9,35,
43,1Ø

Línia 18:
255,8,42,34,9,51,57,41,4Ø,
56,5Ø,9,29,49,33,32

Línia 19:
243,127,48,28

Línia 23:
244,1ØØ,7,79,76

Línia 27:
248,99,98,93,9Ø,86,83,78,77

Línia 31:
248,1,17,2,16,3,15,8,1Ø

Línia 42:
244,67,7Ø,88,91

Línia 45:
248,6Ø,63,66,71,87,92,95,98

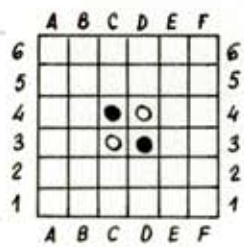
Línia 48:
248,68,69,74,77,81,84,89,9Ø

Línia 51:
248,61,62,73,78,8Ø,85,96,97

Línia 54:
244,59,64,94,99

Per poder executar el programa, cal un mínim de 3 mòduls de memòria i el mòdul X-Functions.

Com l'anterior, aquest programa juga sobre un tauler 6×6:



Utilització:

a) feu XEQ "OTG"

b) després de la fase d'inicialització apareix: [C4: P-J?] en mode ALPHA.
Cal contestar P o J segons que volguem que la casella C4 estigui inicialment ocupada per una peça del programa (P) o del jugador (J).

c) llavors demana [TU?]. Si volem que la primera jugada la faci la màquina, simplement pulsem [R/S].
Si volem començar nosaltres, escrivim la jugada (estem en mode ALPHA) amb la lletra de la columna i el número de la fila, i [R/S].

d) a continuació la màquina actualitza el tauler i rumia la seva jugada. Passat el temps de reflexió, sona un BEEP i anuncia la seva jugada, p. ex.: [JO D2,].
Triga encara uns segons durant els quals actualitza el tauler, i finalment s'atura demanant la jugada humana: [JO D2, TU?]

e) escrivim la resposta i [R/S].
Si ens veiem obligats a passar, només fem [R/S], i tornem al pas d).

El programa complet cap en 3 targes magnètiques. No calen targes de dades. La inicialització completa està inclosa al programa, segons una variant del mètode descrit al "bits" (24) de gener '83.

Tots els fitxers que puguessin existir a la X-Memory queden esborrats.

El temps mig per jugada en una màquina sense accelerar és al voltant d'un minut, però pot variar força, segons la complicació de la posició en curs.

El programa no fa intervenir l'atzar, és a dir, davant una determinada posició, la resposta és sempre la mateixa.
Si es vol, però, resulta senzill d'afegir-hi l'atzar conservant el nivell de joc i la rapidesa.

Per a això, només cal afegir

durant la fase d'inicialització una rutina que sumi aleatòriament quantitats de Ø,Ø1 als valors posicionals.

La funció d'avaluació és:

avaluació = valor posició − nombre de peces canviades − diferència contra-jugada.

amb les següents observacions:

a) quan queden menys de 12 caselles buides, el terme del nombre de peces canviades és eliminat.
b) la diferència contra-jugada és igual a la diferència entre els valors posicionals de la jugada i la millor contra-jugada; tret del cas que la jugada és millor que la millor contra-jugada, en quin cas la diferència contra-jugada és nul·la.

Aquesta funció és un compromís entre efectivitat i rapidesa.

Com que la taula de jugades està ordenada per ordre decreixent de valors posicionals, i com que els altres termes són negatius, des del moment que trobem una jugada el valor posicional de la qual ja sigui inferior a l'avaluació màxima trobada, ja no cal seguir l'estudi.

Codis interns de les caselles:

100	101	102	103	104	105	106	107
93	94	95	96	97	98	99	100
86	87	88	89	90	91	92	93
79	80	81	82	83	84	85	86
72	73	74	75	76	77	78	79
65	66	67	68	69	70	71	72
58	59	60	61	62	63	64	65
51	52	53	54	55	56	57	58

Valors posicionals:

45	32	40	40	32	45
32	30	36	36	30	32
40	36			36	40
40	36			36	40
32	30	36	36	30	32
45	32	40	40	32	45

El significat dels registres és:

RØØ: codi jugador contrari al que està en estudi.

RØ1 a RØ8: taula dels vectors de direcció (fixa durant tota l'execució).

RØ9 i R1Ø: caselles en consideració.

R11 i R12: comptadors de les taules.

R13 i R14: avaluacions màxima i actual.

R15: valor posicional.

R16: codi casella triada.

R17: comptador contra-jugades

R18: casella actual.

R19 a R5Ø: taula de les jugades (conté els codis de les caselles per ordre decreixent de valor posicional. Mentre la casella no és adjacent a cap casella ocupada, el contingut és nul, per evitar càlculs innecessaris).

RS1 a R1Ø7: taula de les caselles. Conté:

- +1 si la casella és ocupada per una peça del programa.
- −1 si és ocupada per una peça del jugador.
- Ø si és exterior al tauler.
- XX,YY si és interior i lliure (XX és la localització a la taula de jugades; YY és el valor posicional).

56 ➤ØØ en mode RUN

Encara unaltra propietat de la funció ➤__.

Si el punter de programa és dins un programa que conté un LBL Ø1, i estem en mode RUN (és a dir, no en mode PRGM), si fem ➤ØØ, això modificarà els octets del programa en la posició des de la qual executem ➤ØØ.

És a dir, sense entrar en mode PRGM, podem modificar la memòria de programa.

Això és degut a què ➤ØØ equival a un GTO no compil·lat, l'efecte del qual és de grabar la longitud del salt als octets que haurien de cumplir aquesta missió en un GTO normal.

Com a exemple, feu el següent:

Introduïu el programa:

```
Ø1  LBL "PROVA
Ø2  LBL Ø1
Ø3  "ABC
Ø4  END
```

Ara passeu al mode RUN (sortiu de PRGM) i col·loqueu-vos a la línia Ø3:

```
GTO "PROVA"
GTO .ØØ3
```

Ara feu ➤ØØ i torneu al mode PRGM. El programa s'haurà convertit en el següent:

```
Ø1  LBL "PROVA
Ø2  RCL IND X
Ø3  −
Ø4  *
Ø5  /
Ø6  END
```

Nota: Per qui no conegui la funció ➤__, repasseu els bits (29) (febrer '83), (35) (març '83) i (43) (març '83).

57 Codi màquina 6502

```
LBL "BRANCH
AON
"Ad. BRANCH?
XEQ Ø1
X<>Y
STO ØØ
"Ad. DESTI?
XEQ Ø1
AOFF
CLX
RCL ØØ
−
2
−
-128
X>Y?
GTO Ø3
CHS
X<=Y?
GTO Ø3
ST+ X
X<>Y
X<Ø?
+
STO ØØ
"BRANCH:
16
/
INT
XEQ Ø4
RCL ØØ
16
MOD
XEQ Ø4
PROMPT
LBL Ø1
55
PROMPT
16
FRC
LBL Ø2
ATOX
X=Ø?
RTN
X<>F
LASTX
ST* Z
CF Ø4
CF Ø5
X<>Y
X<>F
R↑
X<Y?
ST- Y
RDN
ST+ Z
R↑
R↑
GTO Ø2
LBL Ø3
"MASSA LLARG
PROMPT
LBL Ø4
9
X<>Y
X>Y?
16
+
39
+
XTOA
END
```

Aquest programa calcula l'adreça relativa que cal incloure a la instrucció BRANCH quan programem en codi màquina el micro 6502.

Evitem així haver de dur a terme càlculs pesats i propensos a errors.