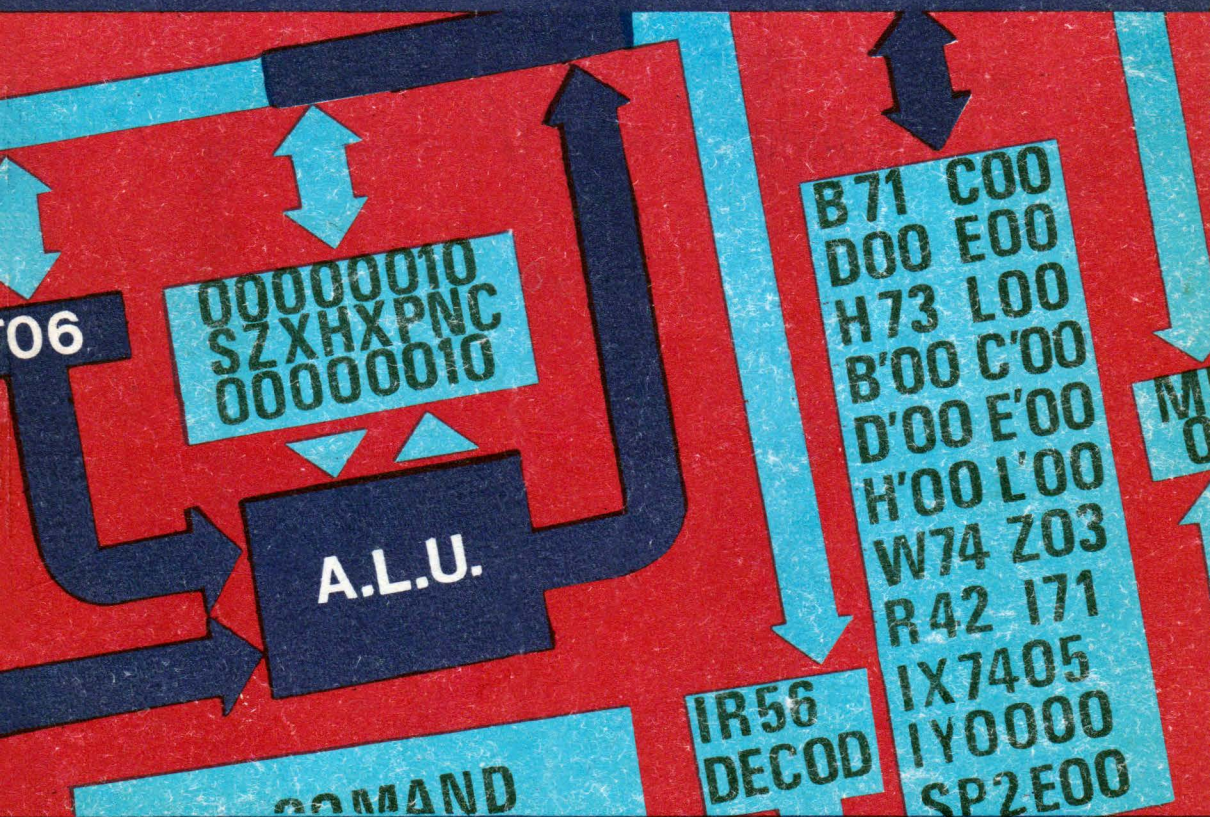


M. PATRUBÁNY

Volumul 2

TOTUL DESPRE... MICROPROCESORUL

Z80



EDITURA TEHNICA



Ciclu „Totul despre...”

1. Popduche. Automatizări aplicate, edita 1 și a II-a
2. Alexandru. Automatizări procesor tehnologic în industria chimică
V. H. Lăscărin. Proiectare tehnică-aplicată în ramurile industriale
G. Raymond. Tehnică relativă în color
T. Homan. Caracterizarea de producție în construcții de mașini
S. Radu. D. Biliu. Centrale telefonice automate. Sisteme de comunicații
R. Șteer. Proiectarea și construcția de aparatură de măsură
D. N. Șopru. Proiectarea radioreceptorilor
Y. Antonescu. M. Popovici. Ghid pentru controlul statistic al calității producției
N. Șteacuc. Tehnică imaginii în cinematograful și televiziune
P. Vezouan. Șt. Pătrușcu. Măsurarea timpului în construcții
I. Pătrușcu. V. Pătrușcu. Măsurarea presiunii în construcții
P. Popescu. P. Mihăilescu. Măsurarea debitului în construcții
P. Vezouan. Măsurarea nivelului în construcții
C. Hildeș. P. Băc (coordonator). Studiul muncii, vol. I—VIII
V. Bălcă. Șt. Căciulea. FELIX C-256. Structură și programare
R. J. Morris. Proiectarea cu circuite integrate TTL
K. Kawa. Kawa. Controlul de calitate pentru mașini și țevi de echipaj
A. M. Bărbulescu. Șt. Căciulea. Cursurile de probleme de programare
P. Constantin. Sisteme informatic, modele ale conductivității și sistemelor conduse
E. Ș. Băfă. Conductoare moderne a producției, vol. I și II
A. Văduțescu. Șt. Dispozitive semiconductoare. Manual de utilizare
A. Nădăla. Măsurarea volumului și cantității lichidelor în industrie
Ch. Jones. Design. Metode și aplicații
G. Păduș. Șt. Elaborarea și introducerea sistemelor informatic
C. Hildeș. Analiza și proiectarea circuitelor informatic în unitățile economice
A. Văduțescu. Șt. Circuite integrate liniare. Manual de utilizare vol. 1, 2, 3, 4
M. Șteacuc. Șt. Scheme de televiziune, magnetofon, vol. 1 și 2 ed. a II-a
D. W. Davies. Rețele de interconectare calculatoarelor
V. Pătrușcu. Șt. Fișiere, baze și baze de date
Ch. Bărbulescu. Șt. Comandă numerică a mașinilor-unelte
M. Șteacuc. Șt. Automatizări discrete în industrie. Cursuri de probleme
M. Pătrușcu. Șt. Cibernetică, automatizări, informatică în industria chimică
Ș. Căciulea. Optimizări în automatizări industriale
Ș. Mărcău. Sisteme numerice cu circuite integrate
I. Răduț. Șt. Manualul muncitorului electronic
M. Șteacuc. Proiectarea unității a circuitelor electronice
G. Ciucur. Tehnică măsurătorilor în telecomunicații
P. Nădăla. Caracterizarea instalațiilor de telecomunicații
R. Popescu. Șt. Circuite integrate analogice. Catalog
Șt. Lozancu. Șt. Caracterizare. Descriere. Funcționare
T. Răduțescu. Șt. Centrale telefonice automate
N. Josif. Șt. Tranzistori și modele de putere. Catalog
P. Postolnicu. Sisteme și linii de transmisii telefonice
M. Șteacuc. Șt. Receptoare TV în color
V. Bălcă. Șt. Sisteme interactive și limbaj conversational
Y. Bălcă. Șt. Calculatoare electronice, aplicații interactive, prelucrare imaginilor
T. Gărbu. I. Miu. Șt. Echipamente de calculare pentru sisteme în timp real
A. Davidoviciu. Șt. Bălcă. Limbaje de programare pentru sisteme de distribuție a datelor
M. Gărbu. Șt. Filip. Sisteme interactive în timp real cu prelucrare distribuită a datelor
În ciclul „Totul despre...” a apărut: A. Pătrușcu. Șt. „Totul despre... calculul personal și MIC

- M. K. Starr. Conducerea producției.
- T. Baron ș.a. Calitatea produselor Manual practic.
- A. Vlădescu ș.a. Radioreceptoare
- M. Mayer. Tiristoare în practică. Mutatoare cu comutație forțată
- G. Moltgen. Tiristoare în practică. Mutatoare cu comutație de la rețea
- L. Zamfirescu, I. Oprescu. Automatizarea cuptoarelor industriale
- I. Papadache. Automatica aplicată, ediția I și a II-a
- Șt. Alexandru. Automatizarea proceselor tehnologice în industria lemnului
- V. H. Lisicikin. Prognostica tehnico-științifică în ramurile industriei
- G. Raymond. Tehnica televiziunii în culori
- T. Homoș. Capacitatea de producție în construcții de mașini
- S. Radu, D. Filoti. Centrale telefonice automate. Sisteme de comutație.
- R. Stere ș.a. Tranzistoare cu efect de câmp
- D. N. Sapiro. Proiectarea radioreceptoarelor
- V. Antonescu, M. Popovici. Ghid pentru controlul statistic al calității producției
- N. Stanciu ș.a. Tehnica imaginii în cinematografie și televiziune
- P. Vezeanu, Șt. Pătrașcu. Măsurarea temperaturii în tehnică
- T. Penescu, V. Petrescu. Măsurarea presiunii în tehnică
- P. Popescu, P. Mihordea. Măsurarea debitului în tehnică
- P. Vezeanu. Măsurarea nivelului în tehnică
- C. Hidoș, P. Isac (coordonator). Studiul muncii, vol. I—VIII
- V. Baltac ș.a. Calculatorul FELIX C-256, Structură și programare
- R. L. Morris. Proiectarea cu circuite integrate TTL
- Ishikawa Kaoru. Controlul de calitate pentru maștri și șefi de echipe
- A. M. Buhtiarov ș.a. Culegere de probleme de programare
- P. Constantinescu. Sisteme informatice, modele ale conducerii și sistemelor conduse
- E. S. Buffa. Conducerea modernă a producției, vol. I și II
- A. Vătășescu ș.a. Dispozitive semiconductoare. Manual de utilizare
- A. Nadolo. Măsurarea volumului și cantității lichidelor în industrie
- Ch. Jones. Design. Metode și aplicații
- Gh. Pisău ș.a. Elaborarea și introducerea sistemelor informatice
- C. Hidoș. Analiza și proiectarea circuitelor informaționale în unitățile economice
- A. Vătășescu ș.a. Circuite integrate liniare. Manual de utilizare vol. 1, 2, 3, 4
- M. Silișteanu ș.a. Scheme de televizoare, magnetofone, vol. 1 și 2 ed. a II-a
- D. W. Davies. Rețele de interconectarea calculatoarelor
- V. Pescaru ș.a. Fișiere, baze și bănci de date
- Gh. Baștiurea ș.a. Comanda numerică a mașinilor-unelte
- N. Sprinceană ș.a. Automatizări discrete în industrie. Culegere de probleme
- M. Florescu ș.a. Cibernetică, automată, informatică în industria chimică
- S. Călin. Optimizări în automatizări industriale
- S. Maican. Sisteme numerice cu circuite integrate
- I. Ristea ș.a. Manualul muncitorului electronist
- M. Simionescu. Proiectare unitară a circuitelor electronice
- C. Cluceru. Tehnica măsurărilor în telecomunicații
- P. Nițulescu. Electroalimentarea instalațiilor de telecomunicații
- R. Rapeanu ș.a. Circuite integrate analogice. Catalog
- Șt. Lozneanu ș.a. Casetofone. Depanare. Funcționare
- T. Rădulescu ș.a. Centrale telefonice automate
- N. Iosif ș.a. Tiristoare și modele de putere. Catalog
- P. Postelnicu. Sisteme și linii de transmisii telefonice
- M. Silișteanu ș.a. Receptoare TV în culori
- V. Baltac ș.a. Sisteme interactive și limbaje conversaționale
- V. Baltac ș.a. Calculatoare electronice, grafica interactivă, prelucrarea imaginilor
- T. Geber, I. Miu ș.a. Echipamente periferice 1, 2, 3
- A. Davidoviciu, B. Bărbat. Limbaje de programare pentru sisteme în timp real
- M. Guran, Fl. Filip. Sisteme ierarhizate în timp real cu prelucrarea distribuită a datelor
- În ciclul Total despre... a apărut: A. Petrescu ș.a., Total despre... calculatorului personal a MIC

Patrubańy Miklós

Cuprins general

**TOTUL
DESPRE ...
MICROPROCESORUL**

Z80

Volumul 2

Volumul 2

**EDITURA TEHNICA
Bucureşti, 1989**



(Note: The following table is a reconstruction of the visible content from the image, which is largely mirrored and difficult to read. It lists chapters and page numbers.)

1. Structura internă	13
2. Semnalele externe	31
3. Secul de instrucţiuni	38
4. Cicliuri de bază	57
5. Instrucţiuni de transfer	73
6. Instrucţiuni aritmetice şi logice	125
7. Instrucţiuni diverse	174
8. Instrucţiuni de intrare/ieşire	181
9. Cicliuri speciale	181
10. Interconexiuni	224
11. Specificaţie tehnică	17
12. Structura constructivă	17
13. Definiţii	17

- Prefață:** ing. GH. CONSTANTINESCU
Recenzii: dr. ing. ADRIAN DAVIDOVICIU,
dr. ing. NICOLAE ȚĂPUȘ
dr. ing. LIVIU DUMITRAȘCU,
Redactor: ing. PAUL ZAMFIRESCU
- Coperti:** arh. SILVIA PINTEA,
arh. LIVIU DERVEȘTEANU (ilustrații)
Tehnoredactor: V. E. UNGUREANU
C.Z. : 681,3
Bun de tipar: 6 05 1989
Coli de tipar: 15
- ISBN 973-31-0009-9
ISBN 973-31-0007-2
- INTREPRINDEREA POLIGRAFICĂ CLUJ,**
Municipiul Cluj-Napoca, cd. nr. 425/1987



Cuprins general

Volumul 1

Prefață	V
Cuvânt înainte	VI
Despre carte și despre autor	X
Cuprins general	XI
Despre carte în limba engleză. Cuprins general în engleză	XII
Cuprins detaliat vol. 1	XIX

Partea I

M icroprocesorul Z80 : o prezentare inedită. Simularea funcționării lui Z80, cu și fără casetă magnetică. Aventuri cu Visible—Z80	XXIII
0. Praeludium. Calculatoare (retrospectivă) și microprocesoare.	1
1. Structura internă.	12
2. Semnalele externe	31
3. Setul de instrucțiuni	38
4. Cicluri de bază	57
5. Instrucțiuni de transfer	73
6. Instrucțiuni aritmetice și logice	120
7. Instrucțiuni diverse	144
8. Instrucțiuni de intrare/ieșire	176
9. Cicluri speciale	187

Partea a II-a

Complemente privind microprocesorul Z80 și simulatorul său	231
A. Manualul de utilizare a casetei de simulare Visible—Z80	233
B. Procedură de testare folosită pentru validarea programului Visible—Z80	250
C. Comparație semantică a limbajelor de asamblare ale microprocesorului 18080 și Z80	257
D. Date de catalog ale microprocesorului Z80	263
E. Analiza setului de instrucțiuni ale microprocesului Z80	271
F. Lista celor 696 instrucțiuni declarate ale microprocesorului Z80	395
G. Cele 458 instrucțiuni ascunse ale microprocesorului Z80	420

Volumul 2

Cuprins general	V
Cuprins detaliat vol. 2	VI
<i>Partea a III-a</i>	
Programare microprocesorului Z80.	
Un proiect : o casă de marcat electronică	1
10. Interregnum. Ghid de proiectare structurată	3
11. Specificația tehnică	17
12. Structura constructivă (Hardware)	25
13. Definirea structurilor de date	60
14. Implementarea programului (Software)	78

15. Module software dedicate pentru testare	122
16. Inițializare și supervizare	134
17. Lista comentată a programului	141
18. Memento-ul proiectului	217
19. În loc de probleme și exerciții „Învierea casei” de marcat	224
Bibliografie	232
Caseta cu produsele-program pentru calculatoarele personale PRAE și a MIC (pentru cărțile cu casetă)	VII

Cuprins detaliat

Volumul 2

Partea a III-a

Programarea microprocesorului Z80 : Un proiect : o casă de marcat electronică

Cap. 10. Interregnum. Ghid de proiectare structurată	3
10.1. Software-ul : artă sau meserie ?	3
Noțiuni de programare structurată	
10.2. Instrumente de lucru	9
10.2.1. Organigrama	9
10.2.2. Limbajul de nivel înalt ("pseudo" PASCAL)	11
10.2.3. Asamblorul	14
Cap. 11. Specificația tehnică	17
11.1. Specificația constructivă	17
11.2. Specificație funcțională	18
11.3. Funcții de bază	20
11.4. Funcții speciale	22
11.5. Formatul bonurilor	23
Cap. 12. Structura constructivă (Hardware)	25
12.1. Subansamble funcționale	25
12.1.1. Tastatura	25
12.1.2. Dispozitivul de afișaj	34
12.1.3. Imprimanta	39
12.1.4. Generatorul de sunet	43
12.2. Structura hardware a echipamentului	56
Cap. 13. Definirea structurilor de date	60
13.1. Structuri de bază	60
13.1.1. Codurile interne	61
13.1.2. Generatorul de caractere (segmente) pentru dispozitivul de afișaj	62
: LEDGEN	
13.1.3. Generatorul de caractere (puncte) pentru imprimantă	63
: PRTGEN	
13.1.4. Mesaje în EPROM	64
13.2. Zone de manevră	65
13.2.1. Buffer de intrare date de la tastatură : KEYBUF	66
13.2.2. Buffer de ieșire pentru dispozitivul de afișaj : LEDBUF	66
13.2.3. Buffer de ieșire pentru imprimantă : PRTBUF	67
13.2.4. Buffer de editare pentru imprimare : EDBUF	69
13.2.5. Tabela de distribuire a parametrilor de stare : SETUPTAB	71

13.3.	Regiștri RAM		72
13.3.1.	Registrul datelor în format BCD extins : EBCD		73
13.3.2.	Regiștri de aritmetică BCD		74
13.3.3.	Regiștri BCD de uz general, în memorie		74
13.4.	Farametri și variabile		75
13.5.	Fluxul de date în casa de marcat (o primă aproximație)		77
Cap. 14. Implementarea programului (Software)			78
14.1.	Bucła principală	: MAINLOOP	79
14.1.1.	Bonul client normal	: BUYTICK	81
14.1.2.	Procesarea tastei FUNC	: PROCFUNC	83
14.1.3.	Programarea sesiunii de lucru	: SETUP	86
14.1.4.	Bonul client anulat	: ERATICK	89
14.1.5.	Totalul vânzărilor pe sortimente	: TOTSORT	91
14.1.6.	Totalul vânzărilor pe zi	: TOTDAY	93
14.1.7.	Sinteza vânzărilor	: SYNTH	94
14.2.	Programe de prelucrare		96
14.2.1.	Citirea unui preț	: INPRICE	96
14.2.2.	Prelucrarea unui preț	: PRPRICE	98
14.2.3.	Anularea unui preț	: ERPRICE	99
14.2.4.	Pregătirea următorului preț	: NXPRICE	100
14.2.5.	Citirea codului de sortiment	: INCODE	100
14.2.6.	Generarea implicită a codului de sortiment	: IMPLCODE	101
14.2.7.	Numărător pentru tastări succesive FUNC	: CNTFUNC	102
14.2.8.	Operații pentru client.	: OPFORBUY	103
14.2.9.	Emiterea bonului client normal	: EMITBUYTICK	104
14.2.10.	Emiterea bonului client anulat	: EMITERATICK	104
14.2.11.	Imprimarea din EDBUF	: PRTTICK	104
14.2.12.	Localizarea registrului de sortiment	: SORTADR	107
14.2.13.	Actualizarea totalului de sortiment (+/-)	: ADDSORT	108
		: SUBSORT	
14.3.	Aritmetică BCD în virgulă fixă		108
14.3.1.	Adunarea a două numere	: ADDBCD	108
14.3.2.	Scăderea a două numere	: SUBBCD	110
14.3.3.	Înmulțirea a două numere	: MULTBCD	110
14.3.4.	Incrementarea unui număr în format BCD extins : EBCDINC		111
14.4.	Analiză sintactică și conversii de coduri		113
14.4.1.	Analiza și conversia numerelor introduse de la tastatură	: SYNTAN	113
14.4.2.	Împachetarea numerelor EBCD în format BCD	: EBCDBCD	114
14.4.3.	Conversia inversă a numerelor	: BCDCI	116
14.5.	Vehiculare de date		117
14.5.1.	Depunerea unui caracter în KEYBUF	: STORENUM	117
14.5.2.	Depunerea unui caracter în LEDBUF	: DISPNUM	118
14.5.3.	Depunerea unui caracter în KEYBUF și LEDBUF	: STORDISP	118
14.5.4.	Normalizarea numerelor introduse	: STOREINT	118
14.5.5.	Afișarea unui rezultat din PRTBUF	: DISPSUM	118
14.5.6.	Stergerea bufferelor KEYBUF și LEDBUF	: CLBUFDP	119
14.5.7.	Umplerea unor zone RAM cu constante	: FILLBYTS	119
14.5.8.	Accesul la TMPBCD	: MOVTMP	
		: TMPMOV	119
14.5.9.	Distribuirea parametrilor de stare	: TRANSPAR	119
14.5.10.	Transferul mesajelor din EPROM în RAM	: EXPAND	120

Cap. 15. Module software dedicate pentru testare	122
15.1. Test de EPROM	122
15.2. Test de RAM	125
15.2.1. Test de RAM nedistructiv (de conținut)	125
15.2.2. Test de RAM distructiv (de adresare)	127
15.3. Test pentru dispozitivul de afișaj	130
15.4. Test pentru imprimantă	130
Cap. 16. Inițializare și supervizare	134
16.1. Pornirea rece	: CSTART 135
16.2. Pornirea caldă	: WSTART 136
16.3. Protecția la căderea accidentală a tensiunii de alimentare	: DROPOUT 138
Cap. 17. Lista comentată a programului	141
Cap. 18. Memento-ul proiectului	217
18.1. Harta memoriei RAM	217
18.2. Fluxul de date în casa de marcat (aproximația finală)	219
18.3. Ierarhia modulelor de software elaborate	219
18.4. Lista rutinelor încorporate în produs	221
Cap. 19. În loc de probleme și exerciții. „Învierea” casei de marcat	224
19.1. Enunțul temei de lucru	224
19.2. Ghid de lucru.	225
19.2.1. Exerciții impuse	225
19.2.2. Figuri „liber alese”	227
19.3. Tabela codurilor ASCII	228
19.4. Lista rutinelor uzuale ale calculatoarelor personale PRAE și aMIC	229
Bibliografie	232

INTERREGNUM

Partea a III-a

Dați zidarii să construiască casele în aceeași manieră în care programatorii să creeze programele, atunci primul ciocănitoare ar distruge civilizația

M. WEINBERG

PROGRAMAREA MICROPROCESORULUI Z80 UN PROIECT: O CASĂ DE MARCAT ELECTRONICĂ

Cei care au aproape totul despre structura microprocesorului și despre setul său de instrucțiuni. Pentru el putea utiliza un limbaj de programare care să descrie în mod clar și precis hardware-ul, ceea ce va permite dezvoltarea hardware-ului fiind să se banalizeze, concomitent cu creșterea vertiginoasă a investițiilor de efort și a cheltuielilor necesare pentru elaborarea pachetelor de software dedicate. Păstrând măsura proporțiilor, putem afirma că una și aceeași placă de unitate centrală, echipată cu microprocesor, memorie (RAM și EPROM) și câteva dispozitive de intrare/ieșire, poate constitui nucleul oricărui echipament. Începând cu o mașină de spălat automată, trecând pe la calculatoarele personale până la roboții industriali. Elementul distinctiv va fi în toate aceste cazuri software-ul.

Dedicăm partea a III-a a cărții prezentării, pe baza unui studiu de caz detaliat a tehnicii de programare a microprocesorului Z80 în limbaj de asamblare, limbaj care permite exploatarea la maximum a resurselor oricărei unități centrale de calculator.

Pentru descrierea mai clară a algoritmilor, în întregul volum vom folosi și un limbaj de nivel înalt, un "pseudo" PASCAL.

Devansăm studiul de caz prin acest capitol intermediar, în care vom sintetiza câteva principii de bază, nutrirând speranța că pe această cale să concușăm la înlăturarea afirmației malițioase citată mai sus.

10.1. Software-ul: artă sau meserie? Noțiuni de programare structurată

În scurta istorie de aproximativ 40 de ani a calculatoarelor electronice, odată cu suportul hardware au evoluat spectaculos și limbajele de programare, numărul și diversitatea lor fiind astăzi foarte mare.

Elementul care a făcut posibile afirmații de genul celui din motto-ul acestui capitol, este cristalizarea relativ târzie a unor tehnologii pentru elaborarea pachetelor de software. Recomandările cu caracter general, universal acceptate, s-au impus abia la mijlocul deceniului trecut, adică după aproape 30 de ani de existență.

Cap. 15. Metode software dedicate pentru teste	122
15.1. Test de EPROM	122
15.2. Test de RAM	125
15.2.1. Test de RAM nedistructiv (de continut)	125
15.2.2. Test de RAM distructiv (de adresare)	127
15.3. Test pentru dispozitivul de afisaj	130
15.4. Test pentru imprimanta	130
Cap. 16. Initalizare si supervizare	134
16.1. Pornirea rece	135
16.2. Pornirea calda	136
16.3. Protectia la caderea de tensiune	138
Cap. 17. Lista programelor	141
Cap. 18. Metode de proiectare	217
18.1. Harta marcată	217
18.2. Fluxul de lucru	219
18.3. Metode de proiectare	219
18.4. Liste de date	221
Cap. 19. Lucrul la calculator	224
19.1. Enunțuri de lucru	224
19.2. Ghid de lucru	225
19.2.1. Execuții impuse	225
19.2.2. Figuri „liber alese”	227
19.3. Tabela codurilor ASCII	228
19.4. Lista rutinelor uzuale ale calculatoarelor personale PRAE și AMIC	229
Bibliografie	232

INTERREGNUM

Dacă zidarii ar construi casele în aceeași manieră în care programatorii își creează programele, atunci prima ciocănitoare ar distruge civilizația"

G. M. WEINBERG

Ghid de proiectare structurată.

Cei care au parcurs prima parte a acestei cărți știu totul, sau aproape totul despre structura microprocesorului și despre setul lui de instrucțiuni. Pentru al putea utiliza eficient este necesar să se cunoască modul de programare al microprocesorului, căci așa cum demonstrează tendințele de dezvoltare, elaborarea hardware-ului tinde să se banalizeze, concomitent cu creșterea vertiginosă a investițiilor de efort și a cheltuielilor necesare pentru elaborarea pachetelor de software dedicate. Păstrând măsura proporțiilor, putem afirma că una și aceeași placă de unitate centrală, echipată cu microprocesor, memorie (RAM și EPROM) și câteva dispozitive de intrare/ieșire, poate constitui nucleul oricărui echipament, începînd cu o mașină de spălat automată, trecînd pe la calculatoarele personale pînă la roboții industriali. Elementul distinctiv va fi în toate aceste cazuri software-ul.

Dedicăm partea a doua a cărții prezentării, pe baza unui studiu de caz detaliat a tehnicii de programare a microprocesorului Z80 în limbaj de asamblare, limbaj care permite exploatarea la maximum a resurselor oricărei unități centrale de calculator.

Pentru descrierea mai clară a algoritmilor, în întregul volum vom folosi și un limbaj de nivel înalt, un "pseudo" PASCAL.

Devansăm studiul de caz prin acest capitol intermediar, în care vom sintetiza cîteva principii de bază, nutrirînd speranța ca pe această cale să concurăm la înfrimarea afirmației malițioase citată mai sus.

10.1. Software-ul : artă sau meserie ? Noțiuni de programare structurată

În scurta istorie de aproximativ 40 de ani a calculatoarelor electronice, odată cu suportul hardware au evoluat spectaculos și limbajele de programare, numărul și diversitatea lor fiind astăzi foarte mare.

Elementul care a făcut posibile afirmații de genul celui din motto-ul acestui capitol, este cristalizarea relativ tîrzie a unor tehnologii pentru elaborarea pachetelor de software. Recomandările cu caracter general, universal acceptate, s-au impus abia la mijlocul deceniului trecut, adică după aproape 30 de ani de exis-

tență a calculatoarelor electronice. Fenomenul este explicabil, și se poate regăsi în orice domeniu de activitate umană: validarea unor metode de elaborare, se poate face doar după câțiva ani buni, răstimp în care se pot consemna anomalii, erori și imperfecțiuni, imprevizibile în etapa de definire și de lansare.

În perioada de început, acceptarea sau refuzul unui pachet de software se făcea pe baza funcționalității. S-a considerat a fi program bun, acel program care rezolva corect problemele formulate în specificația sa. Odată cu trecerea anilor s-a demonstrat că este aproape exclus ca un pachet software mai voluminos să fie complet lipsit de erori. Ieșind la iveală, unele mai devreme, altele mai târziu-cîteodată chiar după ani de exploatare ireproșabilă, aceste erori au impus apariți, unei noi noțiuni: întreținerea (service-ul) software-ului. În aceeași direcție a acționat și necesitatea crescîndă de a adapta unele programe la condiții noi, cona cretizate prin modificarea unor elemente din specificația funcțională sau din cea a suportului hardware.

Astfel s-a ajuns la situația aparent stupefiantă ca aproximativ 80% din activitatea de software în lume să se refere la adaptarea, punerea la punct și service-ul unor programe existente, și nu la elaborarea altora noi.

În aceste condiții criteriile de calificare a programelor s-au diversificat, pe lângă cerințele de corectă funcționare și cele de performanță (viteză de execuție, capacitate), apărînd cele legate de ușurința cu care programul considerat poate fi înțeles de alții și cu ușurința cu care el va putea fi modificat pentru o nouă implementare.

La urma urmei aceste cerințe de calitate au impus respectarea unor canoane (reguli) de elaborare, care luate în ansamblu formează tehnologia programării.

Un set din aceste recomandări se constituie în a forma metodologia programării structurate, una din tehnicile cele mai eficiente.

Programarea structurată își propune să elaboreze produse software în care să se distingă clar structurile principale ale programului (aidoma structurilor de rezistență a clădirilor), structuri care vor fi proiectate, programate și testate înainte de a aborda orice problemă de detaliu.

Stilul acesta de abordare a problemelor, începînd cu ansamblul și coborînd treptat la detalii (top-down) caracterizează fiecare etapă de lucru pe parcursul elaborării unui produs program structurat.

În lucrarea lui, S. Williams [14] sintetizează principalele recomandări de programare formulate în lucrările [12]—[18], [20], [23].

Vom adopta aceste recomandări pentru microprocesorul Z80, completîndu-le cu altele rezultate pe baza experienței noastre.

Recomandări generale

a) *Proiectați programul înainte de a începe să-l scrieți.* Programe bine proiectate din punct de vedere logic se scriu și se pun la punct mult mai ușor decît cele a căror scriere s-a demarat fără rumegarea consistentă a problemei. Cu cît se consumă mai mult timp pentru proiectarea unui program, cu atît mai ușor se va realiza acel program.

b) *Proiectați programe structurate.* Programele structurate se pun mult mai ușor la punct și se pot modifica mai ieftin (adapta la noi cerințe) decît cele nestructurate (de exemplu: programele „cîrnaț”).

c) *Incepeți totdeauna proiectarea cu nivelul ierarhic superior, abordând problema din punctul de vedere al utilizatorului.*

d) *Impuneți-vă convenții de programare, și folosiți-le cu maximă perseverență în întregul program.*

e) *Includeți testarea modulelor în procesul de elaborare, avansând spre abordarea detaliilor cu nivele ierarhic superioare puse la punct.*

f) *Comentați cât mai bine programele, astfel încât ele să poată fi înțelese și de alții.*

g) *Căutați un partener care să revizuiască tot ceea ce ați făcut.*

h) *Acordați nume cât mai sugestive modulelor create. Insistând asupra acestei activități, un „naș” bun poate scuti pe el însuși și pe colegi de multe linii de comentariu explicativ.*

Vom parcurge în continuare principalele etape de lucru : *proiectarea, elaborarea, testarea, finalizarea.*

Proiectarea programelor (de sus în jos, de la ansamblu la detaliu)

Metoda sugerată este contrară instinctului. Se cere ca să proiectăm module software care nu fac aproape nimic, neștiind încă cum se va rezolva oricare problemă concretă. Totuși aceasta este calea cea bună.

a) *Puneți pe hîrtie descrierea a ceea ce urmărește programul. În cazul pachetelor mai complexe, scrieți documentația de utilizare a aceluia program înainte de a fi scris nici măcar o linie din program. Numai astfel, încercînd a scrie, vă puteți pune în situația utilizatorului. Pe parcursul elaborării manualelor de utilizare va trebui să clarificați multe aspecte care v-au scăpat la specificarea produsului sau pe parcursul elaborării proiectului logic.*

b) *Proiectați structurile de date înainte de a scrie nici măcar o linie din program. Structurile de date constituie un element esențial al proiectului logic, ele putînd juca un rol determinant în tehnicile de programare pe care va trebui să le folosiți.*

c) *Împărțiți problema în module funcționale și elaborați descrierea lor într-un limbaj descriptiv sau pe organigrame.*

d) *Proiectați programele de interfață dintre modulele definite, specificînd clar fluxul informațional intermodule.*

e) *Specificați modul în care veți testa fiecare modul elaborat. Dacă din start nu puteți întrezări o metodă de testare cât mai eficientă, restructurați de pe acum modulele.*

Implementarea și testarea modulelor

Și această activitate se va desfășura de sus în jos, în paralel cu elaborarea modulelor program.

a) *După ce ați elaborat un modul program testați-l și puneți-l la punct. Veți întreba cum anume se poate pune la punct un program care apelează rutine care încă nici nu sînt concepute ? Aceste rutine le veți înlocui cu rutine „oarbe”, care nu fac nimic (RET) sau, returnează o valoare constantă, sau vă imprimă un mesaj de genul : „ai fost la mine : x”, unde x este numele rutinei „oarbe”.*

apelate. (În literatura de limbă engleză aceste module se numesc destul de spirital și „*stubroutine*”, „*praf*” (în ochi ?), în loc de „*subroutine*”. Testînd astfel toate ramurile modului elaborat, puteți avansa în munca de elaborare a programului, substituind treptat rutinele „oarbe” cu cele reale.

b) Fiți pregătiți să modificați (eventual să reprojecțați) unele module ierarhic superioare, în măsura în care avansați la cele inferioare, dacă rutinele reale impun acest lucru. Veți scăpa astfel de munca destul de anevoioasă de rescriere a programelor de interfață dintre module, rescriere a cărei necesitate apare de obicei la detectarea unor anomalii pe parcursul testării finale.

c) După terminarea unor module mai mari, rugați pe cineva să vă revizuiască programele. Această etapă de lucru este deosebit de importantă, din mai multe motive :

- Există probabilitatea reală ca cel care a elaborat un program să cadă mereu în aceeași „capcană logică”, scăpîndu-i astfel de repetate ori o eroare. Modul de gîndire, iminent diferit, al unei alte persoane va permite detectarea facilă a unor astfel de sincope.

- Pe parcursul acestei colaborări ambii parteneri pot învăța unii de la alții.

- Pe această cale primiți un „*feedback*” despre inteligibilitatea programului dvs.

- Încercînd să explicați lectorului unele secvențe din program s-ar putea să descoperiți chiar dvs. unele erori care pînă în acel moment v-au scăpat.

Urmarea acestor recomandări este desigur dificilă pentru începătorul absolut. El va trebui să se familiarizeze mai întîi cu limbajul pe care intenționează să-l folosească, să-și formeze anumite deprinderi, studiînd și modificînd programe existente, iar doar după aceea să-și propună elaborarea unui program nou, caz în care îi recomandăm să urmeze recomandările formulate mai sus.

În continuare vom înșira cîteva intimități de programare.

Recomandarea unor tehnici de detaliu

- Nu folosiți niciodată coduri de instrucțiune ca date. Nu modificați coduri de instrucțiune prin program. Urmărirea, punerea la punct și înțelegerea unor astfel de programe este deosebit de anevoioasă.

- Încercați să cadrați fiecare modul program pe o pagină. Dacă el se va dovedi a fi mai lung, atunci transformați părți din el în subrutine, astfel încît să realizați prezenta cerință.

- Încercați să dirijați instrucțiunile de salt astfel încît destinația lor să fie mai jos pe pagină. Astfel asigurați citirea programului după modul obișnuit de citire al unui text. (Recomandarea nu este valabilă pentru salturile de la sfîrșitul buclelor, care se vor efectua obligatoriu în sus).

- La folosirea unor instrucțiuni de salt condiționat, încercați să le aranjați astfel încît lista programului să continue cu condiția falsă. Condiția adevărată ar trebui să fie (pe cît posibil) locată într-un modul separat. Astfel permitem utilizatorului să identifice mai ușor bucla principală a modului respectiv. În caz contrar s-ar putea să-l obligați pe urmaș să răsfoiască pagini întregi de listing pentru a identifica cele 10 instrucțiuni ale buclei principale.

Exemplu : Dacă vă propuneți ca într-un program să se lanseze activități diferite pentru cazul în care tastați literele A, B, C, D, bucla principală ar trebui să se constituie după cum urmează :

```
LOOP : CALL INKEY      ; se citește tasta
CP      "A"
JP      Z,JOBA
CP      "B"
JP      Z,JOBB
CP      "C"
JP      Z,JOBC
CP      "D"
JP      Z,JOBD
JP      LOOP
```

■ *Incercați să elaborați module cu un singur punct de intrare și un singur punct de ieșire. Pe cât posibil, intrarea să fie prima instrucțiune de pe pagină, iar ieșirea ultima.*

■ *Evitați salturile care trec de limita paginii respective. În mod normal recomandarea nu se referă la salturile care se efectuează la module separate și din care nu se mai revine în pagina curentă. (de exemplu : rutină comună de tratare a unor erori.)*

■ *Folosiți cât mai multe nume simbolice pentru datele care la o nouă implementare s-ar putea modifica.*

■ *Folosiți nume de etichete și de simboluri care să fie cât mai sugestive, permițând identificarea funcției lor deja pe baza numelui simbolic.*

■ *Folosiți totdeauna etichete pentru adresele de destinație a unor salturi. Evitați salturile referite printr-un deplasament față de valoarea curentă a contorului program al asamblorului. În acest din urmă caz, inserarea sau eliminarea oricărei instrucțiuni din corpul programului va da peste cap adresa de destinație a saltului.*

■ *La intrarea într-o subrutină verificați încadrarea parametrilor în limitele de valori pe care le admiteți. În acest caz nu se vor întâmpla evenimente necontrolate la apariția unor valori de apel neprevăzute. Cea mai bună soluție este cea de a genera în aceste cazuri, un mesaj de eroare, care să identifice locul anomaliilor și valoarea neprevăzută primită.*

■ *Limitați pe cât posibil comunicația între subrutine la un registru, mereu același. Nu vă așteptați niciodată ca la revenirea din subrutină valoarea aceluia registru să fie nemodificată.*

■ *Salvați și restaurați în subrutine toți regisrii, cu excepția celui desemnat pentru comunicație. Vă scutiți astfel de multă bătaie de cap la implementarea programului, când (nerespectînd această recomandare) o întrebare plină de nedumerire „Cine mi-a stricat registrul B?” va fi destul de frecventă.*

■ *Evitați să intercalați instrucțiuni de salt condiționat între două operații de stivă, PUSH și POP. Dacă pe o ramură veți uita să reechilibrați stiva, prin efectuarea aceluiași număr de POP-uri și PUSH-urile parcurse în amonte, instrucțiunea de revenire din subrutină (RET) nu va încărca valoarea adresei de revenire în PC, ci o dată oarecare, caz în care programul „o va lua în bălării”.*

■ *Mai ales în faza de început a carierei de programator, evitați pe cât posibil folosirea instrucțiunilor EX (SP),HL ; EX (SP),IX sau EX (SP),IY.*

● *Nu folosiți instrucțiunea EXX pentru salvări curente de regiștri.* Urmărind listingul vă va fi greu să „știți” în fiecare moment, care din setul de regiștri este cel activ. Recomandăm să utilizați regiștri secundari pentru programarea unor variabile globale, și/sau ca set alternativ de lucru în rutinele de tratare a întreprerilor. Veți putea folosi în schimb această instrucțiune (EXX), ori de câte ori cerințele de viteză nu permit salvări pe stivă sau în memoria de lucru.

Dacă veți fi reușit să puneți la punct programul astfel încât el să funcționeze aparent fără erori, să nu credeți că treaba a fost terminată. Greul de-abia începe : trebuie să puneți la punct documentația de implementare și să o actualizați pe cea de utilizare.

Documentarea programelor

Documentația de utilizare a unui program este de obicei un material distinct, destinat utilizatorilor produsului finit, pentru acei, pe care îi interesează modul de operare al programului și nu modul în care el rezolvă problemele. Când definiți această documentație, e bine să încercați să vă transpuneți în „pielea” utilizatorului, pentru a-i putea fi realmente de folos.

Documentația de implementare este la fel de importantă ca și cea de utilizare. Calitatea documentației de implementare poate juca un rol determinant în viața unui produs software. Dacă ea nu este inteligibilă sau dacă are lipsuri, atunci efortul de asimilare va fi probabil prea mare, fapt care va determina pe noul implementator să ia totul de la început, rescriind întregul program.

Pentru a nu se putea pierde, recomandăm includerea documentației de implementare sub forma unor comentarii în însăși corpul programului.

lata sugestiile noastre.

a) *Includeți la începutul programului o descriere a funcției principale a programului.* Treceți în revistă principalele module ale programului, precum și joncțiunile lor. Amintiți principalele convenții folosite, tot aici.

b) *Includeți și instrucțiunile (comenzile) necesare pentru o nouă generare a codului din programul sursă.*

Aceasta este destinația originală a fișierelor apelate prin comanda **SUBMIT**, (CP/M, ISIS II, SFDX), care vor include toate comenzile necesare pentru constituirea codului obiect, care poate proveni din mai multe fișiere sursă. Într-un caz ideal, fiecărui program sursă ar trebui să i se atașeze și un fișier de generare cod, apelabil prin comanda **SUBMIT**.

c) *Includeți o descriere clară a fiecărei structuri de date importante, la începutul blocului de date sau a modulelor care vehiculează aceste date.*

d) *Prevedeți la începutul fiecărei rutine o descriere textuală a funcției rutinei respective, a regiștrilor afectați și a celor folosiți pentru transferul informațional.*

e) *Specificați în clar funcția fiecărei secvențe de cod.* Dacă undeva folosiți artificii sau manevre mai greu inteligibile, descrieți detaliat tehnica folosită.

f) *Structurați programul prin folosirea spațiilor și a liniilor goale, astfel încât entitățile distincte să „sară în ochi”.*

g) *Comentați pe cât se poate fiecare linie a programului sursă, mai ales secvențele greu inteligibile.*

Dacă ați parcurs cu atenție recomandările făcute în prezentul paragraf, veți putea da singuri răspunsul la întrebarea formulată în titlu.

Este incontestabil faptul că activitatea de programare este una pur intelectuală, abstractă. Este normal ca pe parcursul acestei activități să cădem în ispita unor aventuri spirituale, complexe. Spiritul inovator este un alt imbold care ne împinge spre adoptarea unor soluții nemaîntinse. Iată de ce programarea poate fi considerată artă.

Dar odată cu răspîndirea pe scară largă a calculatoarelor și în contextul cerințelor impuse modulelor software, enunțate la începutul paragrafului, va trebui adesea să acceptăm compromisul simplității.

Nu se poate concepe o industrie de software, fără respectarea regulilor de „conviețuire” amintite. Iată de ce programarea coboară treptat din sferele înalte, devenind pe zi ce trece tot mai mult o meserie, cu reguli care trebuie respectate, o meserie a intelectului.

10.2. Instrumente de lucru

Vom prezenta principalele mijloace care se vor folosi pe parcursul elaborării programelor în limbaj de asamblare.

10.2.1. Organigrama

Pentru a elabora un program care să rezolve o problemă dată, este necesar ca înainte de toate să extragem esența problemei, spargînd soluția în pași individuali de efectuat. Secvența activităților astfel obținute se numește **algoritm**. Ca să ilustrăm acest procedeu vom algoritmiza procedura de realizare a unei legături telefonice.

1. Ridicăm receptorul și așteptăm tonul.

2. Dacă tonul nu sosește în câteva secunde, închidem receptorul și reluăm activitatea începînd cu punctul 1.

3. Dacă tonul a sosit, formăm numărul dorit și așteptăm formarea apelului.

4. Dacă sosește semnalul de „ocupat”, atunci închidem receptorul și reluăm activitatea începînd cu punctul 1.

5. Dacă se formează apelul, așteptăm ca apelatul să răspundă.

6. Dacă apelatul nu răspunde timp de 30–40 sec., atunci abandonăm activitatea; STOP.

7. Dacă apelatul ridică receptorul, legătura telefonică este stabilită; STOP.

Această procedură-cunoscută de toți — are toate elementele unui program de-calculator. Astfel distingem o secvență de inițializare (1.), trei bucle (2.→1.; 4.→1. și 5.), o ieșire anormală (6.) și o rezolvare dorită a problemei (7.).

Reprezentarea grafică a unui algoritm se numește **organigramă**.

În fig 10.1. redăm organigrama algoritmului prezentat.

Organigrama se constituie dintr-o serie de căsuțe interconectate prin segmente direcționate, care indică căile de derulare a algoritmului.

Organigramele se scriu în limbaj natural și/sau folosind expresii matematice/logice

Pentru constituirea organigramei se folosesc trei simboluri principale :

- căsuța dreptunghiulară, care specifică întreprinderea unei activități ;
- căsuța romboidală, care specifică luarea unei decizii ;
- săgețile care unesc cele două tipuri de căsuțe.

Recomandăm ca **organigrama** să fie cuprinsă pe o singură pagină. Dacă dimensiunile ei depășesc formatul paginii, atunci organigrama se defalcă pe mai multe pagini, continuitatea lor fiind asigurată prin căsuțe numerotate.

Organigramele au o importanță deosebită în elaborarea programelor de calculator. Elaborarea unor **organigrame** corecte, înainte de a începe programarea efectivă a problemei, poate juca un rol hotărâtor în ceea ce privește timpul și efortul necesar pentru implementarea unui program. Se estimează că doar 10% din programatorii lumii știu să programeze și fără organigramă. Nenorocirea este că și ceilalți 90% consideră că fac parte din cei 10%. Drept urmare majoritatea programelor elaborate necesită eforturi mari pentru a fi puse la punct.

Cert este că odată cu creșterea experienței de programare a unui individ, organigramele se pot constitui „în cap”, trecându-se direct la programare. Și în acest caz, lipsa unor organigrame se va resimți în procesul de service al programului, în momentul în care cineva va trebui să înțeleagă acel program.

Iată de ce vă recomandăm să folosiți totdeauna organigrama, ori de câte ori programul dvs. depășește 10-15 linii.

În partea a doua a cărții se găsesc numeroase exemple în acest sens.

Menționăm că organigramele nu prezintă unicul mijloc de fixare a unui algoritm. Datorită simplității lor, odată cu evoluția limbajelor și a conceptelor de programare, organigramele au pierdut teren pe alocuri. Au apărut limbajele

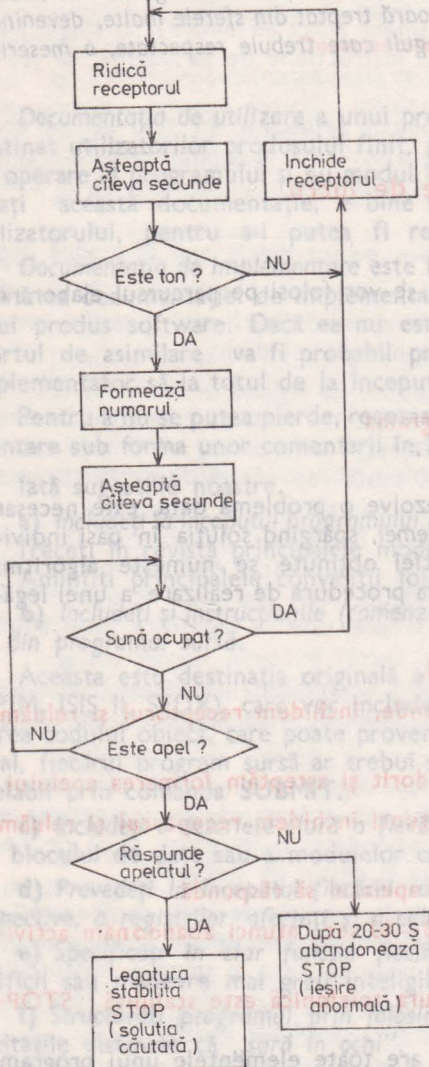


Fig. 10.1. Organigrama unui apel telefonic

de nivel înalt *descriptive*, care se mulează adesea mai bine pe unele tipuri de algoritmi.

Ambele metode se vor putea folosi, ele *completându-se* în multe cazuri cu succes.

10.2.2. Limbajul de nivel înalt ("pseudo" PASCAL)

Limbajele de nivel înalt au apărut în anii '50, ele constituind un *salt calitativ* în dezvoltarea limbajelor de programare. Dintre numeroasele limbaje de nivel înalt create în aproximativ 30 de ani pot fi amintite: **FORTRAN, COBOL, BASIC, PL/1, PASCAL, C, ADA**, etc. Alegerea unuia dintre ele *depinde de problema* de rezolvat (de exemplu, **FORTRAN** se utilizează mai ales în aplicații științifice, **COBOL** în probleme cu caracter economic, **C** pentru programe de sistem, etc.). Utilizarea acestor limbaje a redus considerabil timpul de elaborare al programelor.

Pe lângă *avantajele* oferite, limbajele de nivel înalt au și anumite *dezavantaje*. În primul rând, *codul obținut* este de obicei mult mai *voluminos* decât cel obținut prin utilizarea unui limbaj de asamblare. *Avantajul portabilității* este diminuat de faptul că ele nu exploatează anumite posibilități ale hardware-ului.

Limbajele de nivel înalt (mai exact, anumite "pseudo" limbaje de nivel înalt) pot fi folosite însă și în cazul în care programul se scrie în limbaj de asamblare. În acest caz, *limbajele de nivel înalt* pot fi utilizate pentru descrierea mai clară a *algoritmilor*, în faza de proiectare, ele impunându-se mai ales în cazul metodei *top-down*.

Să luăm ca exemplu apelul telefonic din paragraful anterior. La o primă aproximare algoritmul arată astfel :

"se sună persoana dorită"

"dacă apelatul se prezintă, începe conversația"

Folosind un pseudo-Pascal :

```
procedure apel  
begin
```

 „se sună persoana dorită”

 „dacă apelatul se prezintă, începe conversația”

```
end
```

Să dezvoltăm mai departe cele două acțiuni. Cea de a doua este mai simplă :

dacă „apelatul răspunde în 30—40 sec ”

atunci : „legătura stabilită”

altfel : „se abandonează”

În Pascal :

```
if „apelatul răspunde în 30—40 sec.”
```

```
then „legătura stabilită”
```

```
else „se abandonează”
```

Prima acțiune: „se sună persoana dorită”, se dezvoltă mai departe în felul următor:

repetă
 G. „obține tonul”
 „formează numărul”
 „așteaptă semnalul de răspuns”
 până când „sună soneria”

Mergînd tot așa mai departe, în final se obține următoarea procedură în pseudo — Pascal:

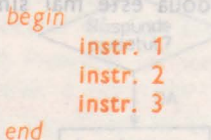
```

procedură apel
  begin
    repeat
      begin
        repeat
          if „receptorul ridicat”
            then „închide receptorul”
          else
            „ridică receptorul”
            „așteaptă citeva secunde”
          end
        until „este ton”
        „formează numărul”
        „așteaptă semnalul de răspuns”
      end
    until „sună soneria”
  if „apelatul răspunde în 30—40 sec.”
    then „legătura stabilită”
    else „se abandonează”
  end apel
  
```

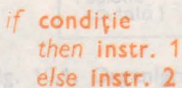
În continuare vom prezenta pe scurt toate instrucțiunile limbajului utilizat. Prin instr. vom înțelege orice instrucțiune a limbajului: atribuire, apel de rutină, etc.

■ **Instrucțiunea compusă**

Un grup de instrucțiuni așezate într-un corp **begin--end**, formează o **instrucțiune compusă**, echivalentă cu o singură instrucțiune



■ **Instrucțiunea if**



Dacă condiția este adevărată se execută instr. 1, altfel se execută instr. 2. Organigrama corespunzătoare în fig. 10.2.

Ramura else poate și să lipsească.

if condiție
then instr.

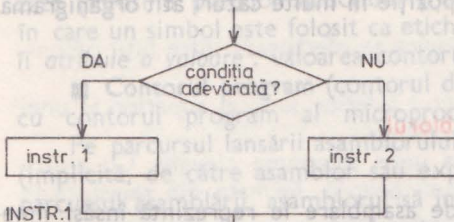


Fig. 10.2.

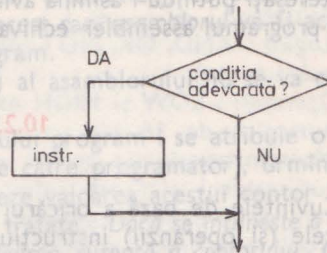


Fig. 10.3.

■ Instrucțiunea **while**

while condiție **do** instr

Dacă condiția este adevărată, se execută instr și se revine la testarea condiției. Dacă la un moment dat condiția devine falsă, se trece la executarea instrucțiunii ce urmează după **while**. Organigrama fig. 10.4.

■ Instrucțiunea **repeat**

repeat instr. **until** condiție

instr. se execută o dată sau de mai multe ori, pînă cînd condiția devine adevărată. Atunci se trece la instrucțiunea ce urmează după **repeat**. (Fig. 10.5)

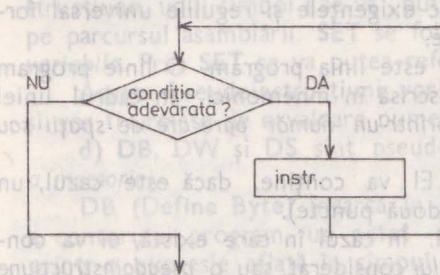


Fig. 10.4.

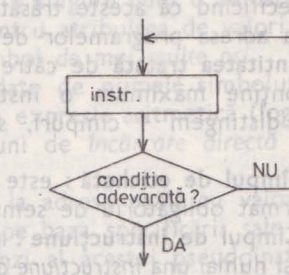


Fig. 10.5.

■ Instrucțiunea **case**

case e of

e1 : instr. 1

e2 : instr. 2

e3 : instr. 3

en : instr. n

Dacă e = e1, se execută instr. 1, dacă e = e2, se execută instr. 2

Limbajul de nivel înalt pe care vi-l propunem se aseamănă cu limbajul Pascal, instrucțiunile folosite reprezentând un subset al acestuia.

Remarcați desfășurarea pe orizontală a buclelor interioare, tehnică care ușurează identificarea părților funcționale distincte ale programului, înlesnind reconstituirea algoritmului.

Pe parcursul părții a doua a acestei cărți vom folosi frecvent acest limbaj; cei interesați putându-l asimila avînd la dispoziție în multe cazuri atît organigrama, cît și programul assembler echivalent.

10.2.3. Asamblorul

Cuvintele de bază a oricărui limbaj de asamblare le reprezintă însăși mnemonicele (și operanzii) instrucțiunilor microprocesorului considerat. Rolul asamblorului este cel de a transpune programele scrise în mnemonice, în codul mașină direct executabil al procesorului.

Un asamblor cît de cît evoluat, nu se va limita la banala traducere, ci va trebui să ofere utilizatorului facilități și servicii suplimentare. Printre acestea amintim :

- posibilitatea de utilizare a unor simbolii și etichete ;
- calculul unor adrese de salt și deplasamente ;
- evaluarea unor expresii aritmetice simple, pentru ca programatorul să-și poată defini variabilele de lucru într-un mod cît mai general și flexibil ;
- posibilitatea asamblării condiționate a unor secvențe din program ;
- posibilitatea folosirii unor macroinstrucțiuni ;
- detectarea și semnalarea unor erori de sintaxă.

Vom prezenta principalele trăsături ale unui asemenea asamblor (de exemplu M80), specificînd că aceste trăsături satisfac exigențele și regulile universal formulate la adresa programelor de asamblare.

■ **Entitatea tratată** de către asamblor este linia program. O linie program poate conține maximum o instrucțiune scrisă în mnemonici. În cadrul liniei program distingem 4 cîmpuri, separate printr-un număr oarecare de spații sau virgulă :

a) **Cîmpul de etichetă** : este opțional. El va conține, dacă este cazul, un simbol urmat obligatoriu de semnul ":" (două puncte).

b) **Cîmpul de instrucțiune** : poate lipsi. În cazul în care există, el va conține una și numai una instrucțiune a procesorului considerat, sau o pseudoinstrucțiune a asamblorului.

c) **Cîmpul de operanzi** : va fi folosit în funcție de cerințele impuse de instrucțiunea sau pseudoinstrucțiunea care îl precede.

d) **Cîmpul de comentarii** : orice caracter va fi precedat obligatoriu de semnul ";" (punct și virgulă).

Asamblorul va admite și linii vide, sau linii ce conțin doar o etichetă și/sau un comentariu.

■ **Simbolurile** : sînt un șir de 6 (sau 8) caractere, care încep obligatoriu cu o literă, și pot conține orice cifră și majoritatea semnelor de punctuație. (Semnele de punctuație care nu se admit în numele simbolurilor sînt specificate pentru fiecare asamblor în parte). Asamblorile acceptă simboluri rezervate și simboluri utilizator.

Setul simbolurilor rezervate se constituie din *mnemonicile de instrucțiuni și operanzi* ai microprocesorului, precum și din *pseudoinstrucțiunile* asamblorului considerat.

Simbolurile utilizator sînt definite de către programator pe parcursul elaborării programului. Programatorul va trebui să *atribuie valori numerice simbolurilor* utilizate (folosind pseudoinstrucțiuni și/sau expresii aritmetice), cu excepția cazului în care un simbol este folosit ca etichetă. În acest caz, *asamblorul* va fi acela care îi *atribuie o valoare*: valoarea contorului program.

■ **Contorul program** (contorul de locații) al asamblorului nu se va confunda cu contorul program al microprocesorului.

Pe parcursul lansării asamblorului, contorul program *i* se atribuie o valoare (implicită, de către asamblor sau explicită, de către programator), urmînd ca pe parcursul asamblării, asamblorul să *incrementeze valoarea* acestui contor în funcție de necesarul de memorie al instrucțiunilor tratate. *Dacă se întîlnește o etichetă, atunci simbolului respectiv i se va atribui valoarea curentă a contorului program.*

■ **Etichetele** definite mai sus, sînt folosite pentru a adresa zone de memorie de program (la instrucțiunile de salt), sau pe cele de date (la instrucțiunile de transfer între regiștri și memorie).

■ **Pseudoinstrucțiunile** asamblorului *nu vor fi traduse în cod executabil*, ci ele dirijează diversele activități ale asamblorului. Le vom aminti pe cele mai uzuale:

a) **ORG** — este o pseudoinstrucțiune care permite *inițializarea* la o valoare dorită (exprimată pe doi octeți) a *contorului de program* al asamblorului.

b) **END** — este pseudoinstrucțiunea de *terminare a programului sursă* și va fi locată pe ultima linie de program.

c) **EQU** și **SET** — sînt *instrucțiuni de atribuire* de valori pentru simbolurile utilizator. Cu **EQU** se vor defini *simbolurile constante*: folosind această pseudoinstrucțiune, unui simbol *i* se va putea atribui o *singură dată* o valoare numerică, pe parcursul asamblării. **SET** se folosește pentru atribuirea de valori *simbolurilor variabile*. Prin **SET** se va putea referi un simbol de *mai multe ori*.

Aceste pseudoinstrucțiuni vor fi precedate de numele simbolului selectat, și vor fi urmate de o valoare numerică sau o expresie aritmetică (logică).

d) **DB**, **DW** și **DS** sînt pseudoinstrucțiuni de *încărcare directă și rezervare a memoriei*.

DB (Define Byte) încarcă în memorie, la adresa egală cu valoarea curentă a contorului program, un octet — evaluat pe baza specificării sale directe sau printr-o expresie aflată în cîmpul de operanzi al acestei pseudoinstrucțiuni.

În cîmpul de operand al pseudoinstrucțiunii **DB** pot apărea pînă la 8 valori, separate prin virgulă sau spațiu, care se vor încărca la adrese de memorie succesive.

DW (Define Word) — are o semnificație asemănătoare cu **DB**, doar că ea va încărca valori numerice exprimate pe 2 octeți.

DS (Define Storage) — nu încarcă în memorie, ci *incrementează valoarea curentă a contorului de program* al asamblorului, cu un număr egal cu valoarea specificată în cîmpul de operand al acestei pseudoinstrucțiuni. Astfel se rezervă loc pentru cîmpuri de date ce vor fi completate pe parcursul execuției programului.

e) **IF**, **ENDIF** sînt pseudoinstrucțiuni pentru *asamblare condiționată*. Segmentul program aflat într-un corp **IF—ENDIF** va fi asamblat doar dacă condiția impusă în enunțul instrucțiunii se dovedește a fi adevărată. Condiția se impune prin

operatori relaționali GT (mai mare decât), LT (mai mic decât), EQ (egal) cu ajutorul cărora valoarea unui simbol se compară cu o valoare numerică sau cu o expresie.

■ **Expresiile** vor fi evaluate pe parcursul asamblării. Valoarea calculată se exprimă pe 2 octeți, transporturile de la *bit15* în sus pierzându-se. Într-o expresie pot apărea simbolii și valori numerice. Ele vor putea fi supuse la diverse operații.

a) **Aritmetică** : + — * /

b) **Logică** : AND, OR, NOT (operațiile logice se efectuează între biții omonimi a două cuvinte).

c) **Operatori** : LOW și HIGH extrag octetul inferior sau cel superior al unei valori numerice de 16 biți.

■ **Macroinstrucțiunile** permit elaborarea unor programe elevate. Cu ajutorul lor, utilizatorul își poate defini instrucțiuni proprii, formate din secvențe de instrucțiuni și pseudoinstrucțiuni.

Macroul se definește o singură dată, într-un corp **MACRO—ENDM**, urmînd ca numele specificat în această definiție să poată fi folosit în întregul program ca și oricare alt nume de instrucțiune sau pseudoinstrucțiune acceptată de limbajul respectiv. Pe parcursul asamblării, în locul acestei instrucțiuni „fictive”, asamblorul va expanda secvența de cod din definiția macroului.

Spre deosebire de subrutine, macroul consumă spațiu de memorie, ele expandîndu-se la fiecare apelare, dar prezintă avantajul unor posibilități de formalizare pe care subrutinele nu le oferă (parametri formali și efectivi).

■ **Instrucțiunile de control a listingului**, **TITLE** și **PAGE** vor imprima în fiecare pagină un titlu specificat, respectiv vor cauza salturi de pagină pentru a se asigura formatul de listing dorit.

■ **Directivile EXTERNAL și PUBLICS** permit referirea unor simboluri definite în alt modul program, care se assemblează separat de modulul considerat. Atribuirea valorilor pentru simbolii externi se va face de către un alt program, numit editor de legături, pe baza unei tabele de referințe (simbolii) încrucișate, furnizată de asamblor.

Un asamblor care pune la dispoziția utilizatorului facilității de genul celor enunțate, trebuie să parcurgă textul sursă de cel puțin 2 ori. Pe parcursul primei baleieri se atribuie valori simbolurilor utilizate, se constituie tabelele de simbolii, macroui și referințe urmînd ca la a doua trecere să se genereze efectiv codul binar, efectuîndu-se translatarea propriu-zisă. De aceea acest asamblor va fi numit asamblor cu doi pași (cu 2 treceri).

Asamblorul poate genera un listing complet, ce se poate vedea în Cap. 17, pe care-l va dirija după cerere la unul din perifericele calculatorului. (de exemplu : imprimantă, consolă sau disc). Pe listing se marchează fiecare linie eronată, de obicei în coloana întâia, indicîndu-se și tipul erorii (simbol nedefinit, simbol multiplu definit, sintaxă eronată, operand ilegal, etc.).

Codul obiect rezultat se înregistrează de obicei, direct de către asamblor, pe un suport de memorie externă (de exemplu : disc).

Prezentarea făcută este foarte sumară, dar credem că va fi suficientă pentru a putea urmări programele pe care le vom elabora în studiul de caz care urmează.

SPECIFICAȚIA TEHNICĂ

Ne propunem să construim împreună o casă de marcat electronică, parcurgând toate etapele de lucru, începând cu *specificația constructivă și funcțională a produsului*, trecând prin definirea structurii *hardware*, implementând apoi *programele* și terminând cu elaborarea unor *documente scrise* care trebuie să completeze orice produs software, dacă se dorește ca acesta să poată fi înțeles și actualizat și peste câțiva ani de zile.

Microprocesorul *Z80* va ocupa un loc central în structura electronicii de comandă, el fiind responsabil de interfața om-mașină (tastatură, afișaj, imprimantă, sunet), de operațiile aritmetice pe care casa de marcat le are de efectuat, precum și de activitățile de gestiune și evidență a vânzărilor efectuate într-o sesiune (zi) de lucru.

Casa de marcat pe care o vom elabora este total virtuală, orice asemănare cu o casă existentă fiind o simplă coincidență.

11.1. Specificația constructivă

În continuare vom înșira principalele trăsături constructive ale casei electronice de marcat, notate cu **C.1.** — **C.8.** Împreună ele formează **specificația constructivă** (fără detalii de ordin electric și mecanic) și trebuie să stea la baza oricărui proiect, reprezentând ghidul de lucru al proiectantului hardist. El va trebui să o respecte în întregime, într-un caz ideal. La terminarea proiectului, în raportul său final, el va consemna toate abaterile de la specificația primită.

C.1. Casa de marcat va fi dotată cu o **tastatură proprie**, care va avea următoarele taste distincte :

— cifrele zecimale de la 0—9	10 buc
— punctul zecimal	1 buc
— tastă de adunare (+)	1 buc
— tastă de înmulțire (*)	1 buc
— tastă de ștergere a ultimului număr introdus	1 buc
— tastă pentru programarea unor funcții speciale	1 buc
— tastă pentru emiterea bonului client	1 buc

Total	16 buc
-------	--------

C.2. Tastatura va avea *hardware minimal*, fiind citită de microprocesor prin program.

C.3. Casa de marcat va fi dotată cu un **dispozitiv de afișaj** constituit din diode luminescente (LED) cu 7 segmente și punct zecimal. Se vor reprezenta numere zecimale cu 8 cifre semnificative.

C.4. Afișajul va avea *hardware minimal*, fiind controlat prin baleiere de microprocesor.

C.5. Casa de marcat va fi dotată cu o **imprimantă**, care permite tipărirea pe două fișii de hîrtie distincte a bonurilor de cumpărare. O fișie va ieși în exterior pentru a se putea rupe bonurile pentru client, iar cealaltă se va derula pe o rolă internă, pentru a se păstra istoria vânzărilor. Informația înregistrată pe cele două fișii va fi identică.

C.6. Casa de marcat va fi dotată cu un **difuzor** pe care se vor putea emite *semnale sonore avertizoare*, cît mai *ergonomice*.

C.7. Casa de marcat va fi **protejată la căderile de tensiune**. În lipsa tensiunii de rețea, casa de marcat nu va funcționa, dar va păstra *toate informațiile* cuprinse în memoria sa *intacte*, astfel încît la reparația tensiunii de rețea operația să poată continua din punctul în care ea fusese abandonată la apariția avariei.

C.8. Casa de marcat va fi prevăzută cu **două chei**. Cele două chei vor determina efectuarea sau neefectuarea unor operații. *Funcțiile de bază* (ex. : emiterea unui bon normal) se vor efectua în prezența primei chei (**KEY0**), iar cele *speciale* (ex. : anularea unui bon) se vor face doar în prezența ambelor chei (**KEY0** și **KEY1**).

11.2. Specificația funcțională. Nivel de detaliere 0

F.0.0. Casa de marcat va prelucra **numere** cuprinse în **gama** [0,99999999.99] Numerele pot fi numere *întregi sau zecimale*, cu 2 cifre semnificative în partea zecimală.

F.0.1. Numerele *mai mici decît 1* se vor putea introduce fie *începînd cu zeroul nesemnificativ* din partea întregă, fie *începînd direct cu punctul zecimal*.

F.0.2. La afișarea și/sau imprimarea unor numere, *zerourile nesemnificative* din partea întregă vor fi *substituite cu blancuri* (spații).

F.0.3. Orice *număr introdus greșit* va putea fi *șters* prin apăsarea tastei de ștergere (**CLEAR**), cu condiția ca înainte de tastarea ei să nu se fi apăsat nici o tastă de funcție.

F.0.4. *Introducerea unui preț* se va face cu specificarea **codului de sortiment**.

F.0.5. Distingem 100 de **clase distincte de mărfuri (sortimente)**, codificate cu numere zecimale [0,99].

F.0.6. La introducerea prețului unui produs, **codul de sortiment** poate lipsi. În acest caz se va genera automat **codul implicit** [99].

F.0.7. *Introducerea unui preț* se va termina prin apăsarea tastei „+”.

F.0.8. La apăsarea tastei „+” pe **dispozitivul de afișaj** se va vizualiza **suma curentă** (prețul) a mărfurilor marcate pentru client, iar pe **imprimantă** se va imprima **codul de sortiment și prețul introdus**.

F.0.9. *În timpul introducerii unui preț* se va putea folosi **operația de înmulțire** în locul adunărilor repetate. Astfel, dacă clientul cumpără mai multe bucăți din-

tr-un produs dat, prețul unitar va fi înmulțit cu numărul de bucăți, folosind tasta „*”. În acest caz, la apăsarea tastei „+” se imprimă prețul total alocat produsului respectiv. Pe afișaj apare suma curentă a mărfurilor marcate pentru client.

F.0.10. Bonul client-normal se emite, după introducerea ultimului preț, prin apăsarea tastei **TOTAL**. În acest caz, pe dispozitivul de afișaj va apare suma totală a clientului, iar pe bon se va imprima același număr precum și elemente de identificare a locului și datei calendaristice de emiterie a bonului.

F.0.11. Pe fiecare bon client se va imprima :

- numărul unității comerciale
- numărul casei
- numărul de identificare al casierului
- numărul curent al bonului, emis în sesiunea de lucru în curs
- data (ziua, luna, anul) emiterii bonului
- mesajul „VA MULȚUMIM”.

Aceste date se vor programa o singură dată, la începutul sesiunii de lucru.

F.0.12. Pentru cazul în care clientul nu posedă bani suficienți, în vederea plății, se va prevedea posibilitatea de anulare a bonului introdus. Operația de emiterie a bonului de anulare se va putea efectua doar în prezența șefului de unitate, prezență materializată prin existența cheii a doua (**KEY1**) introduse.

F.0.13. Pentru cazul în care clientul predă ambalaj de schimb (ex. : sticle) se va prevedea posibilitatea de a se scădea contravaloarea acestora din suma totală de plată. Rîndul care conține prețul ambalajelor restituite se va marca pe bon, el putîndu-se distinge de celelalte rînduri de preț, care reprezintă sume de încasat.

F.0.14. Codurile de sortiment rezervate pentru ambalaje vor fi primele 10 [0,9].

F.0.15. În memoria casei de marcat se vor genera următoarele sume :

- totalul vânzărilor/zi (sesiune de lucru)
- totalurile vânzărilor defalcate pe cele 100 de sortimente/zi.

F.0.16. Primele 10 coduri de sortiment [0,9] sînt rezervate pentru ambalaje (sticle, borcane, cutii). Ele nu se vînd niciodată, ci pot fi recuperate prin rîscum-părare de la client. De aceea numerele contorizate pentru primele 10 coduri de sortiment, vor reprezenta bani „ieșiți” din casă, valoarea lor scăzîndu-se din totalul vânzărilor/zi.

F.0.17. Casa de marcat va fi dotată cu funcții speciale care vor permite listarea, la cerere, a următoarelor date :

- totalul vânzărilor/zi
- totalul vânzărilor din cadrul unui sortiment/zi
- sinteza vânzărilor, caz în care se vor lista totalurile aferente tuturor celor 100 de clase de sortimente/zi.

Execuția acestor funcții speciale va fi condiționată de prezența Celei de-a 2-a chei (**KEY1**).

F.0.18. Funcțiile speciale se vor declanșa prin acționări consecutive ale tastei multifuncționale **FUNC**.

F.0.19. Casa de marcat va fi obligatoriu dotată cu programe de test pentru :

- memoria **RAM**
- memoria **EPROM**
- dispozitivul de afișaj

Testele se vor lansa automat în secvența de inițializare, la pornirea „rece”, a casei de marcat. Dacă la testele **RAM** sau **EPROM** se va detecta o eroare, casa

de marcat nu va deveni operațională. Validarea testului de afișaj rămâne sarcina operatorului.

F.0.20. În caz de operare greșită (secvențe eronate) casa va semnala evenimentul prin desconsiderare, sau prin avertismente sonore, sau ambele. Alegerea uneia din cele două soluții se lasă la libertatea implementatorului. Recomandarea este, ca el să se ghideze după criteriul ergonomicității.

11.3. Funcții de bază. Nivel de detaliere 1

Cititorului atent, aidoma proiectantului, îi mai rămân suficiente întrebări deschise privind modul detaliat de funcționare al echipamentului. Vom încerca să răspundem la ele în continuare.

F.1.0. La pornirea „rece”, după efectuarea inițializărilor și a testelor hardware, casa de marcat va șterge dispozitivul de afișaj și înscrie pe poziția cea mai puțin semnificativă (extrema dreaptă) cifra "0" (zero).

Operatorul va tasta obligatoriu tasta **TOTAL**, care va genera un bon vid și va imprima și antetul primului bon. Astfel se va putea testa și funcționalitatea imprimantei, înainte de emiterea primului bon real.

Astfel se ajunge în starea de repaus interbon. În această stare de așteptare se poate iniția oricare din comenzile (funcțiile) casei de marcat.

F.1.1. Din starea de repaus interbon se pot demara următoarele acțiuni :

- emiterea bonului client normal
- programarea parametrilor de stare a casei (data, nr. casă, ...)
- emiterea unui bon de anulare
- generarea totalului de vânzări pentru un sortiment dat
- generarea totalului general de vânzări
- generarea sintezei vânzărilor, defalcată pe cele 100 de sortimente.

F.1.2. Operațiile legate de emiterea unui bon client normal le numim funcții de bază, iar celelalte funcții speciale.

F.1.3. Emiterea unui bon client normal începe cu introducerea unui număr zecimal de la tastatură, număr care reprezintă prețul unui produs. Plecând din starea de repaus interbon, la apăsarea primei cifre, ea se va înscrie în locul zeroului inițial ("0") în poziția cea mai puțin semnificativă a dispozitivului de afișaj. Următoarea cifră tastată va deplasa conținutul dispozitivului de afișaj cu o poziție la stînga, înscriind noua cifră pe aceeași poziție, cea mai puțin semnificativă. Dacă se tastează mai mult de 8 cifre consecutive, primele cifre tastate vor părăsi dispozitivul de afișaj prin extrema stîngă, pierzîndu-se. Tastarea punctului zecimal va înscrie "." în dreapta cifrei celei mai puțin semnificative. Punctul zecimal defilează la stînga, împreună cu cifrele introduse, la apăsarea fiecărei noi taste de cifră.

F.1.4. Pe durata introducerii unui preț, tastele **FUNC** și **TOTAL** sînt desconsiderate. Tasta **CLEAR** este activă. La apăsarea ei se va șterge conținutul înscris pe dispozitivul de afișaj, revenindu-se la secvența de demarare a introducerii unui preț.

F.1.5. Secvența de introducere a prețului se poate termina tastînd "*" sau "+".

F.1.6. Dacă se tastează "*****", înseamnă că prețul introdus este un preț unitar, urmînd ca el să fie înmulțit cu numărul care urmează să fie tastat. Înmulțitorul se introduce după aceleași reguli ca și prețul. La tastarea primei cifre a înmulțitorului, dispozitivul de afișaj se șterge, noua cifră tastată fiind afișată.

F.1.7. Secvența de introducere a înmulțitorului se termină obligatoriu cu tasta "**+**". Tastele **FUNC** și **TOTAL** sînt refuzate. Tasta **CLEAR** șterge afișajul și repune casa de marcat la începutul citirii înmulțitorului.

F.1.8. La tastarea tastei "**+**" se prelucrează prețul introdus. Dacă în cursul introducerii prețului curent s-a folosit operația de înmulțire ("*****"), atunci ea va fi efectuată în acest moment. Prețul astfel obținut este adăugat la totalul clientului, și la totalul sortimentului. Suma curentă a clientului (totalul clientului) se afișează pe dispozitivul de afișaj. Pe imprimantă se imprimă numărul de cod al sortimentului și prețul recent introdus. Dacă s-a folosit "*****", se imprimă prețul înmulțit.

F.1.9. Astfel se ajunge în starea de repaus interpret. Din starea de repaus interpret se poate ieși tastînd un nou preț, sau tasta **TOTAL**.

F.1.10. La apăsarea tastei **TOTAL** conținutul dispozitivului de afișaj nu se schimbă, fiindcă el a indicat oricum totalul clientului. Pe imprimantă se marchează totalul clientului, precum și informațiile de stare care au fost enumerate la **F.0.11**. Formatul de imprimare se va specifica mai jos.

F.1.11. După prelucrarea tastei **TOTAL** se trece în repausul interbon, stare din care poate începe un nou ciclu de funcționare a casei de marcat.

F.1.12. În cazul prezentat nu s-a specificat codul de sortiment. El se generează în acest caz automat pentru valoarea 99.

F.1.13. Codul de sortiment se poate specifica în repausul interbon sau în repausul interpret. Procedura este următoarea: se tastează o singură dată tasta **FUNC**, după care se tastează obligatoriu 2 cifre. Orice altă tastă, diferită de cifră, va fi desconsiderată. Cele 2 cifre se înscriu în pozițiile cele mai puțin semnificative ale dispozitivului de afișaj, glisînd de la stînga la dreapta. După prima apăsare a tastei **FUNC**, pe extrema stîngă a afișajului (cifra cea mai semnificativă), se va înscrie "1". Cele două cifre introduse reprezintă codul de sortiment. La tastarea primei cifre de cod dispare numărătorul de tastări **FUNC** ("1" în cazul de față) din poziția extremă stîngă. Introducerea codului de sortiment se termină odată cu tastarea primei cifre din preț: deci la tastarea celei de a 3-a cifre după **FUNC**. Această a 3-a tastă poate fi și ".". În acest moment afișajul se șterge și prima cifră de preț se înscrie în poziția cea mai puțin semnificativă. În continuare, prețul se introduce așa cum s-a prezentat mai sus.

Pe durata introducerii codului de sortiment tasta **CLEAR** este activă.

F.1.14. Codul de sortiment introdus de la tastatură, sau cel implicit (99) se va imprima pe rîndul prețului curent.

F.1.15. Procedura de răscumpărare a ambalajului de la client, respectă cu mici diferențe procedura descrisă la introducerea unui preț cu cod de sortiment. Diferențele sînt:

— acțiunea se demarează cu două tastări succesive ale tastei **FUNC**, pornind din starea de repaus interpret;

— secvența nu se poate lansa în repausul interbon fiindcă ar genera sumă curentă client cu valoare negativă;

— urmează obligatoriu codul de sortiment, cuprins obligatoriu în domeniul [00,09]:

— la recepția tastei "**+**" se întreprind următoarele acțiuni:

— din totalul clientului se scade prețul ambalajelor, răscumpărate;

- la totalul sortimentului se adună acest preț;
- pe imprimantă se imprimă codul de sortiment, prețul introdus, urmat de semnul "—" pentru a putea fi distins de un preț direct.

După executarea acestei proceduri se revine în repausul interpret.

F.1.16. Dacă, pornind din repausul interbon numărul de tastări succesive ale tastei **FUNC** este mai mare decât 2, atunci se trece la una din funcțiile speciale.

11.4. Funcții speciale. Nivel de detaliere 2

Casa de marcat electronică va accepta pînă la 7 tastări succesive ale tastei **FUNC**. Fie n numărul de tastări succesive. Casa va întreprinde acțiuni diferite pentru valori diferite ale lui n :

- $n = 1$ s-a prezentat
- $n = 2$ s-a prezentat
- $n = 3$ programarea parametrilor de stare
- $n = 4$ emiterea unui bon de anulare
- $n = 5$ generarea totalului unui sortiment dorit
- $n = 6$ generarea totalului de vânzări
- $n = 7$ generarea sintezei vânzărilor

F.2.0. Tastările succesive ale tastei **FUNC** se contorizează în extrema stîngă a dispozitivului de afișaj. Tastările succesive se termină prin apăsarea oricărei taste diferite de **FUNC**.

La apăsarea tastei **CLEAR** se abandonează procedura **FUNC**, revenindu-se în repausul interbon. (De notat pentru operator).

F.2.1. Programarea sesiunii de lucru: $n = 3$.

Se va efectua obligatoriu următoarea secvență:

- a. se introduce numărul unității comerciale pe 3 cifre semnificative. Dacă se introduc mai mult de 3 cifre, se vor considera ultimele 3. Tasta **CLEAR** este activă. Celelalte taste ("**+**", "*****", **FUNC**) sînt desconsiderate. Introducerea parametrului se termină apăsînd tasta **TOTAL**.
- b. se introduce fără alte tastări prelabile numărul casei, pe 2 cifre semnificative. Dacă se introduc mai mult de 2 cifre, se vor considera ultimele 2. Tasta **CLEAR** este activă. Celelalte taste ("**+**", "*****", **FUNC**) sînt desconsiderate. Introducerea parametrului se termină apăsînd tasta **TOTAL**.
- c. se introduce fără alte tastări prelabile data calendaristică curentă, sub formă **zz.ll.aa**, unde **zz** este ziua, **ll** este luna și **aa** anul. Dacă se introduc mai multe semne atunci primele se pierd și se consideră ultimele 8 tastări. Tasta **CLEAR** este activă. Introducerea se termină cu tasta **TOTAL**.
- d. se introduce fără alte tastări prelabile, numărul casierului pe 3 cifre semnificative. Se consideră ultimele 3 tastări. Tasta **CLEAR** este activă. Introducerea se termină prin apăsarea tastei **TOTAL**.

Astfel se termină automat (la cea de-a 4-a tastare **TOTAL**) programarea sesiunii de lucru. Parametri de stare astfel introduși se vor imprima pe fiecare bon. Se revine în repausul interbon.

F.2.2. Emiterea unui bon de anulare : $n = 4$.

Necesită prezența celei de a doua chei (KEY1). Avînd bonul de anulat în față, se tastează pe rînd toate prețurile, urmate de "+". După tastarea tastei TOTAL, se emite bonul de anulare care va conține în dreptul fiecărui preț semnul "A", semnalînd astfel faptul că este un bon de anulare. Dacă în bonul de anulat există un preț de ambalaj (marcat cu "-"), atunci el va apare și în bonul de anulare cu marcajul "-". Pe afișaj va apare suma totală anulată. Suma totală și prețurile individuale se scad din totalul zilei și din totalurile de sortimente.

F.2.3. Generarea totalului de sortiment : $n = 5$

Necesită prezența celei de-a doua chei (KEY1). După $n = 5$ se tastează 2 cifre, care reprezintă codul sortimentului cerut. După a doua cifră se tastează TOTAL. Se generează totalul sortimentului cerut, care va fi afișat, și se va imprima la imprimantă sub forma unui bon special. Formatul bonului îl vom prezenta în fig. 11.3. Pe parcursul citirii codului, tasta CLEAR este activă.

F.2.4. Generarea totalului de vânzări : $n = 6$

Necesită prezența celei de-a doua chei (KEY1). Se tastează tasta TOTAL care va imprima și afișa totalul vânzărilor pe ziua respectivă. Formatul bonului se găsește în fig. 11.4.

F.2.5. Generarea sintezei vânzărilor : $n = 7$

Necesită prezența celei de-a doua chei (KEY1). Se tastează tasta TOTAL. Ca urmare se va genera un bon special cu 100 de rînduri. Pe fiecare rînd se marchează codul de sortiment și suma vânzărilor aferente. Afișajul este inhibat pe durata întregii imprimări. Formatul bonului de sinteză se găsește în fig. 11.5.

F.2.6. Dacă în F.2.3., F.2.4. sau F.2.5. prima tastă după FUNC sau cod (la F.2.3.) diferă de TOTAL, atunci operația se abandonează, și se revine în repausul interbon.

11.5. Formatul bonurilor

Se vor emite următoarele tipuri de bonuri :

- bon client normal
- bon client anulat
- bon cuprinzînd totalul vânzărilor pe un sortiment
- bon cuprinzînd totalul vânzărilor
- bon de sinteză, cuprinzînd totalurile celor 100 de sortimente.

Fiecare bon va avea un antet în care se va specifica numărul magazinului, al casei și data calendaristică curentă. Pe fiecare bon se va imprima jos un număr strict crescător de bon și codul de identificare al casei. Bonurile client se vor termina cu mesajul „VA MULȚUMIM”.

Formatul celor 5 tipuri de bonuri se exemplifică în fig. 11.1.—11.5.

În fig. 11.1. remarcăm faptul că fiecare preț va fi marcat cu "+", dacă codul lui este cuprins între 10—99. La restituirea ambalajelor (coduri cuprinse între 0—9) prețul se va marca cu "-". În stînga jos se găsește numărul strict crescător al bonurilor, iar în dreapta jos identificatorul casierului (018 în exemplul considerat).

UNIT. NR.	117
CASA NR.	08
	23.12.86
26	33.00 A
99	7.50 A
51	238.75 A
01	10.00 -
TOTAL :	269.25 *
1287	018
* VA MULTUMIM *	

Fig. 11.1. Bonul client normal

UNIT. NR.	117
CASA NR.	08
	23.12.86
26	33.00 +
99	7.50 +
51	238.75 +
01	10.00 -
TOTAL :	269.25 *
1286	018
* VA MULTUMIM *	

Fig. 11.2. Bonul client anulat

UNIT. NR.	117
CASA NR.	08
	23.12.86
TOTAL COD	28
	15226.50 *
1288	018

Fig. 11.3. Bon total sortiment

UNIT. NR.	117
CASA NR.	08
	23.12.86
TOTALUL ZILEI	75328.85 *
1289	018

Fig. 11.4. Bon total zi

UNIT. NR.	117
CASA NR.	08
	23.12.86
SINTEZA	
00	1287.85 *
01	22689.15 *
	5876.00 *
97	
98	2338.15 *
99	6532.90 *
1290	018 *

Fig. 11.5. Bon de sinteză

În fig. 11.2. se constată că fiecare preț de marfă va fi semnalat cu "A", exceptând ambalajul restituit, care și în acest caz se scade ("—").

Bonul de sinteză din fig. 11.5., este o fișie lungă care va conține numărul de cod de sortiment, crescînd de la 00 la 99, și vânzările aferente.

12

STRUCTURA CONSTRUCTIVĂ (HARDWARE)

În proiectarea oricărui echipament dotat cu microprocesor se disting clar două activități majore: proiectarea și elaborarea structurii fizice (**hardware**) și proiectarea și elaborarea programelor care se înscriu în memoria nevolatilă a echipamentului, pentru a fi executate de microprocesor (**software**). Cele două domenii diferențiindu-se destul de substanțial și oamenii care vor fi implicați în realizarea componentelor hardware și software vor fi alții. Pentru ca munca celor două colective să convergă eficient este necesar ca, până la un punct dat, munca lor să se desfășoare în comun. Prezentul capitol conține această etapă de demarare a proiectului, etapă în care se elaborează structura hardware-ului. Participarea colectivului de software, nu este necesară numai pentru a-și extrage datele pentru proiectul software, ci și pentru că în lumea microprocesoarelor chiar și hardware-ul este inimaginabil în absența unor programe de comandă. Capitolul vrea să constituie un exemplu în acest sens.

12.1. Subansamble funcționale

Vom prezenta în continuare soluțiile adoptate pentru realizarea interfețelor om-mașină conforme cu specificațiile constructive din paragraful 11.1., urmînd ca la sfîrșitul prezentului capitol să putem defini structura hardware a casei de marcat.

12.1.1. Tastatura

Vom realiza o tastatură controlată integral prin software. Cele 16 taste le vom dispune fizic astfel încît ele să satisfacă cerințele de ergonomie, iar din punct de vedere logic astfel încît codul care rezultă din interpretarea geometriei tastaturii să fie însăși codul pe 4 biți a celor 16 taste. În fig. 12.1. redăm dispunerea fizică a tastelor.

■ Grupînd tastele din punct de vedere logic într-o matrice de 8 coloane 2 rînduri vom putea exploata la maximum facilitățile pe care le oferă microprocesorul Z80, reducînd la minimum necesarul de hardware auxiliar.

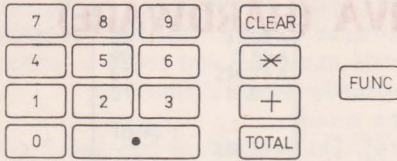


Fig. 12.1. Dispunerea fizică a tastelor

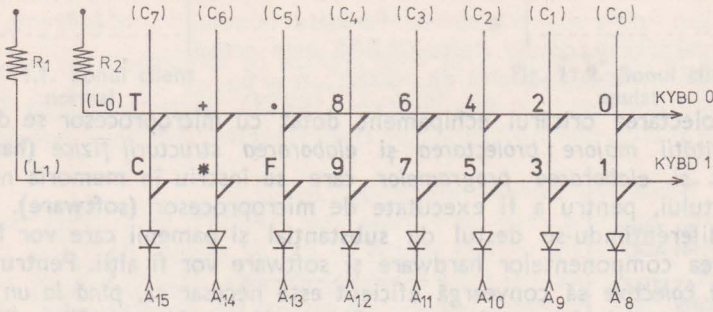


Fig. 12.2. Circuitul electric și dispoziția logică a tastelor

■ La intersecția coloanelor și a liniilor se vor dispune tastele astfel încît la apăsarea tastei care se află la intersecția coloanei x și a liniei y să se scurtcircuiteze coloana x cu linia y .

■ Dacă nici o tastă nu este apăsată, atunci indiferent de starea coloanelor ($C_0 - C_7$), liniile ($L_0 - L_7$) vor fi în starea logică "1", stare conferită de prezența celor 2 rezistențe R_1 și R_2 .

■ Citind cele două linii KYBD0 și KYBD1 printr-un port de intrare microprocesorul va putea identifica starea de repaus sau cea activată a tastaturii. Pentru a determina dacă oricare din cele 16 taste este apăsată, se va genera valoarea "0" pe toate cele 8 coloane (A15—A8). În acest caz scurtcircuitul generat între oricare linie și coloană va determina trecerea în "0" a uneia sau a ambelor linii. Eveniment detectabil pe cale software. Pentru a identifica o tastă individuală, se vor baleia coloanele cu "0" (toate "1" cu excepția uneia "0"). Constatînd care dintre cele două linii KYBD0 și KYBD1 va trece în zero, se poate detecta tasta apăsată. Numărul coloanei se poate codifica pe 3 bit ($8=2^3$), iar cel al liniei implicate, pe un bit. Atașînd cele 2 numere, rezultă un cod pe 4 bit, care determină univoc tasta apăsată.

■ Pentru activarea coloanelor s-au ales liniile superioare ale magistralei de adrese a microprocesorului Z80, datorită faptului că într-un ciclu de citire IN, ele conțin o informație ce provine dintr-unul din regiștri interni ai microprocesorului (A în cazul adresării directe, B — în cazul adresării indirecte „via” registrul C).

■ Diodele sînt menite să izoleze liniile de adresă A8—A15, prevenind astfel scurtcircuitarea lor, la apăsarea concomitentă a unor taste dispuse pe aceeași linie.

■ La elaborarea și implementarea algoritmului de citire a tastaturii va trebui să ținem cont și de regimurile tranzitorii (de origine mecanică) care apar la apăsarea și ridicarea unei taste (Vezi fig. 12.3).

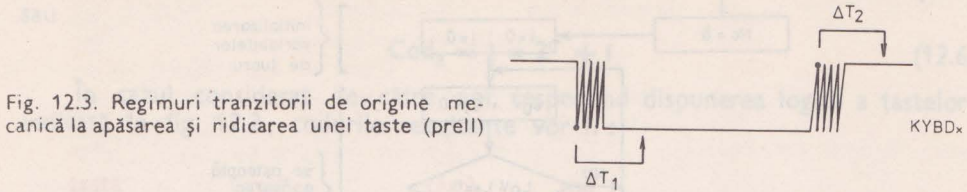


Fig. 12.3. Regimuri tranzitorii de origine mecanică la apăsarea și ridicarea unei taste (prell)

■ Pentru a evita citirea repetată a unei tastări (datorită vitezei mari de prelucrare a microprocesorului el s-ar putea să revină pentru citirea unei noi taste, iar regimul tranzitoriu provocat la ridicarea tastei să nu se fi terminat încă) nu este suficient a se marca prima trecere prin "1" a liniei detectate. Pentru a putea fi siguri de ridicarea tastei, vom genera și o temporizare ΔT_2 de aproximativ 0,1 s, a cărei valoare depinde de caracteristicile fizico-constructive ale tastaturii, și se va determina experimental.

Redăm în fig. 12.4. organigrama rutinei de citire a unei taste, INKEY.

Din organigramă se disting câteva activități majore :

- așteptarea apăsării ferme a unei taste (incluzînd temporizarea ΔT_1 pentru a se evita regimul tranzitoriu care apare la tastare ;
- identificarea coloanei pe care se află tasta apăsată, prin baleierea cu un "0" singular a tuturor coloanelor și citind starea liniilor ; la ieșirea din această secvență variabilă i conține numărul coloanei pe care se află tasta activată ($i \in [0,7]$;
- identificarea liniei pe care se află tasta apăsată ; se caută începînd cu prima linie (L_0) care dintre linii se activează ; incrementînd la fiecare iterație un numărător $j \in [0,1]$, la sfîrșitul secvenței el va conține numărul liniei pe care se află tasta căutată ;
- generarea codului pentru tasta detectată ; se atașează cele două numere i și j astfel încît ele să formeze un număr binar de 4 bit ; j se exprimă pe un bit și va fi pe poziția cea mai puțin semnificativă ;
- așteptarea ridicării definitive a tastei incluzînd temporizarea ΔT_2 , pentru a se evita regimul tranzitoriu.

Matematic, operația de generare a codului se poate descrie prin formula

$$\text{Cod} = 2 * i + j \quad (12.1)$$

Într-un caz general cu C coloane și L linii, numărul de biți b , necesari pentru a genera un cod unic, ar fi :

$$b = c + l \quad (12.2)$$

unde :

$$c = \text{int}(\log_2(C + \epsilon) + 1 - \epsilon) \quad (12.3)$$

$$l = \text{int}(\log_2(L + \epsilon) + 1 - \epsilon) \quad (12.4)$$

ϵ fiind un număr mic : $\epsilon \in [10^{-3}, 10^{-6}]$

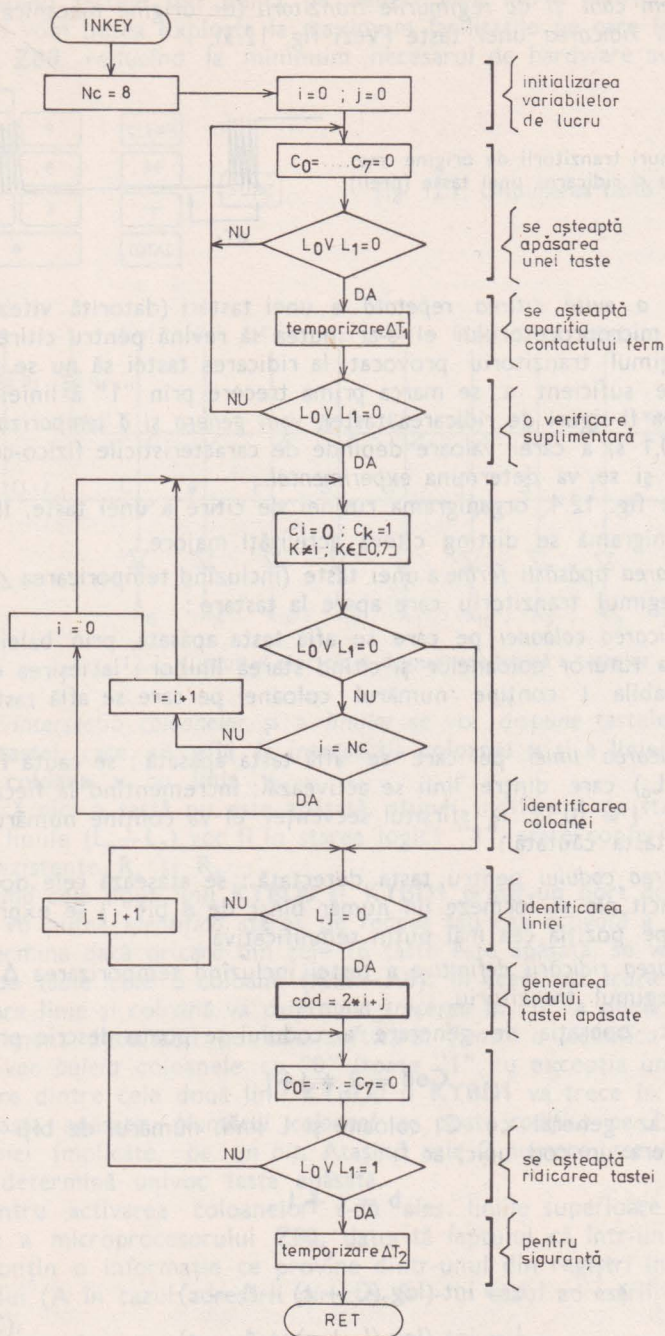


Fig. 12.4. Organigrama rutinei de citire a tastaturii (INKEY)

Codul binar se obține în acest caz aplicând una din relațiile :

$$\text{Cod}_1 = i * 2^i + j \quad (12.5)$$

sau

$$\text{Cod}_2 = j * 2^c + i \quad (12.6)$$

În cazul considerat de către noi, respectând dispunerea logică a tastelor, indicată în fig. 12.2., codurile rezultante vor fi :

tastă	cod	tastă	cod
0	00H	8	08H
1	01H	9	09H
2	02H	●	0AH
3	03H	Func	0BH
4	04H	+	0CH
5	05H	*	0DH
6	06H	Total	0EH
7	07H	Clear	0FH

Putem trece acum la realizarea primului program în limbaj de asamblare : **INKEY**. Dacă se apasă concomitent pe două taste, atunci rutina va retransmite codul aceleia pe care o detectează prima dată prin succesiunea de baleiere stabilită.

Înainte de a începe elaborarea efectivă a programului stabilim următoarele :

- Rutina nu va afecta nici un registru exceptând **A** și **F**.
- Codul tastei se va returna în registrul **A**.

■ Necunoscând încă configurația finală a portului de intrare prin care se vor citi liniile **KYBD0** și **KYBD1**, rutina va fi astfel concepută încât prin modificarea unui singur simbol ea să se poată genera pentru orice dispunere a acestor semnale la intrările portului. Condiția pe care o impunem este ca cele două semnale să fie legate la biți alăturați.

Atribuire în continuare funcții regiștrilor :

- Numărarea coloanelor (**i**) se va face în registrul **H**.
- Numărătorul de linii (**j**) va fi în registrul **L**.
- Adresa portului de intrare **SYSIN** va fi conținută în registrul **C**.
- Octetul de baleiere va fi conținut în registrul **B**.
- Registrul dublu **DE** se va folosi pentru a indica durata temporizărilor

$\Delta T1$ și $\Delta T2$.

■ Ne propunem să alegem *nume* de etichete sugestive și să structurăm programul astfel încât algoritmul prezentat în organigramă să se poată regăsi cât mai ușor.

Iată lista programului :

SHIFTN	EQU	2	
MASK1	EQU	0CH	
	PUSH	BC	;se salvează regiștrii
	PUSH	DE	
	PUSH	HL	
	LD	B,0	;se inițializează variabilele
	LD	C,SYSIN	;de lucru
	LD	HL,0	;i=H, j=L

WAITDOWN:	IN	A, (C)	;se așteaptă
	CPL		;apăsarea
	AND	MASK1	;unei
	JR	Z, WAITDOWN	;taste
	LD	DE, TDOWN	;se așteaptă apariția
	CALL	TMP	;contactului ferm
	IN	A, (C)	;o
	CPL		;verificare
	AND	MA: K1	;suplimentară
	JR	Z, WAITDOWN	;nu strică
	LD	B, OF: H	
SCAN :	IN	A, (C)	;identificarea coloanei
	CPL		
	AND	MA: K1	
	JR	NZ, KEY	
	RLC	B	
	INC	H	
	JR	C, SCAN	
	LD	H, 0	
	JR	SCAN	
KEY :	LD	B, SHIFTN	;identificarea liniei
	RRCA		
	DJNZ	KEY	
SEARCH :	RRCA		
	JR	C, FOUND	
	INC	L	
	JR	SEARCH	
FOUND :	LD	A, H	;determinarea codului
	RLCA		;tastei
	OR	L	
	LD	H, A	
WAITUP :	IN	A, (C)	;se așteaptă
	CPL		;ridicarea
	AND	MASK1	;tastei
	JR	NZ, WAITUP	;apăsate
	LD	DE, TUP	;pentru
	CALL	TMP	;siguranță
	LD	A, H	;se transferă codul
	POP	HL	;tastei în A
	POP	DE	
	POP	BC	;regiștri restaurați
	RET		
TMP :			;rutină de temporizare
	DEC	DE	;4 T _{Cy}
	LD	A, E	;4 T _{Cy}
	OR	D	;4 T _{Cy}
	JR	NZ, TMP	;12(7) T _{Cy}
	RET		;10 T _{Cy}

Urmărind etichetele regăsim principalele secvențe subliniate la descrierile organigramei.

- **WAITDOWN** — se așteaptă apăsarea unei taste
- **SCAN** — baleiere (identificare coloană)
- **KEY** — tastă detectată
- **SEARCH** — caută (identificare linie)
- **FOUND** — „găsit” (se generează codul tastei)
- **WAITUP** — se așteaptă ridicarea tastelor.

Fiind la primul program conceput împreună, ne permitem să *detailăm câteva din tehnicile folosite,*

- În secvența de așteptare a apăsării unei taste :

```
WAIT :   IN      A,(C)
         CPL
         AND     MASK1
         JR      Z,WAIT
```

Conținutul citit de pe port se inversează datorită faptului că tastele sînt active în zero. Pentru a le putea identifica concomitent, după ce toți ceilalți biți au fost eliminați, această operație de complementare este necesară datorită faptului că microprocesorul Z80 nu posedă un flag care să indice apariția unui octet în care toți biții să fie "1". În schimb **flagul Z (zero)** se va înscrie dacă toți biții din **A** devin "0".

- **MASK1** este un octet care se alege astfel încît toți biții săi să fie "0" exceptînd cei doi biți pe care se citesc liniile tastaturii, biți care vor avea valoarea "1". Astfel după execuția instrucțiunii logice **SI**, în acumulator toți biții vor fi „0” exceptîndu-i pe cei selectați prin **MASK1**, care vor reda starea complementată a liniilor **KYBD0** și **KYBD1**.

- În secvența de baleiere (**SCAN**) remarcăm că ea începe totdeauna prin activarea primei coloane (**C₀**), emițîndu-se "0" pe bitul cel mai puțin semnificativ :

$$OFE_H = \overset{D_7}{1} \overset{D_6}{1} \overset{D_5}{1} \overset{D_4}{1} \overset{D_3}{1} \overset{D_2}{1} \overset{D_1}{1} \overset{D_0}{1}$$

Activarea coloanelor se face în cel de-al treilea ciclu mașină al instrucțiunii **IN A,(C)** cînd liniile superioare **A₈—A₁₅** vor conține octetul de baleiere iar cele inferioare **A₇—A₀**, vor conține adresa portului care se dorește a fi citit.

- **Baleierea efectivă** se realizează prin instrucțiunea **RLC B** ce deplasează zeroul cu o poziție la stînga. La fiecare baleiere se incrementează numărătorul de coloană din registrul **H (INC H)**

- **Terminarea unui ciclu de baleiere** (opt citiri urmate de rotiri) se detectează prin apariția valorii "0" în indicatorul de transport **Carry**. La terminarea unui ciclu de baleiere numărătorul de coloane se reinițializează la valoarea "0" (**LD H,0**).

- După identificarea coloanei (fapt consfințit prin efectuarea saltului **JR NZ,KEY**) cei doi biți **KYBD0** și **KYBD1** se deplasează la dreapta pînă cînd **KYBD0** ajunge pe bitul cel mai puțin semnificativ din **A**.

- **SHIFTN** este o variabilă care conține numărul deplasărilor necesare în acest sens.

- În acest moment va începe identificarea liniei, rotîndu-se la dreapta conținutul acumulatorului pînă cînd bitul aferent liniei activate va trece în indica-

torul de transport. Concomitent cu fiecare deplasare se incrementează numărul de linie din registrul L (INC L).

● Dacă execuția programului ajunge la instrucțiunea din dreptul etichetei **FOUND**, registrul H va conține numărul coloanei, în registrul L regăsindu-se numărul liniei pe care se află tasta apăsată.

● Un programator versat va detecta imediat o modalitate de îmbunătățire a secvenței de identificare a liniei. Iată-o :

```
KEY   : LD      B, SHIFN
SEARCH: RRCA
        JR C,FOUND
        INC L
        JR SEARCH
FOUND : LD A,L
        SUB B
        RLC H
        OR     H
        LD     H,A
```

Această secvență presupune că octetul primit la intrarea în **KEY** are toți biții reșetați, cu excepția celor afectați liniilor **KYBD0** și **KYBD1**. Numărătorul de linie va conține în dreptul etichetei **FOUND** și rotirile suplimentare efectuate pentru a disloca **KYBD0** pe poziția cea mai puțin semnificativă din **A (D₀)**. De aceea numărul de rotiri suplimentare **SHIFN** se va scădea pentru a regăsi numărul liniei activate (**SUB B**). Folosind această secvență se poate câștiga timp și spațiu, reducându-se astfel atât lungimea programului cât și timpul de execuție.

● Instrucțiunea **RLCA** din programul enunțat și **RLC H** din exemplul de sus efectuează o deplasare la stânga a numărului de coloane, deplasare echivalentă cu înmulțirea cu 2.

● **Observație** : Remarcăm sintaxa instrucțiunilor de salt relativ precum și a celei **DJNZ** în care operandul de deplasament este înlocuit cu o adresă fizică, materializată prin prezența unei etichete.

Exemplu :

DJNZ KEY.

În acest caz **KEY** nu este un deplasament ci adresa fizică a instrucțiunii la care se dorește a se efectua saltul. Calculul deplasamentului respectiv rămîne sarcina asamblorului. Astfel se ușurează munca programatorului, și se elimină o sursă ineupizabilă de erori (chiar și cei mai experimentați programatori greșesc des la calculul acestor deplasamente).

● Rutina de temporizare **TMP** se bazează pe decrementarea unui registru dublu de la o valoare inițială la zero.

Plaja de temporizare care se poate acoperi cu ajutorul acestei rutine este cuprinsă între valorile : (În cazul unui microprocesor funcționînd la 2,5 MHz, $T_{Cy} = 400$ ns)

$$DE = 0001$$

$$T_{min} = 29 * T_{Cy} * 1 = 11,6 \mu s$$

$$DE = 0000$$

$$T_{max} = 65536 * 34 * T_{Cy} - 5 * T_{Cy} = \\ = 891.277,6 \mu s$$

● Valorile **TDOWN** și **TUP** se determină experimental datorită faptului că ele *depind de caracteristicile mecanice* ale unei tastaturi. În cazul tastaturilor cu folie a calculatoarelor **PRAE** valorile $T_1=0,03$ s și $T_2=0,07$ s conferă o *bună siguranță* de tastare, *fără să reducă sensibil viteza de reacție* a tastaturii.

În cap. 17 (Lista programului) la p. 1—56, se găsește lista rutinei **INKEY** care rezolvă aceeași problemă, dar este implementată în mod diferit de cazul prezentat. Pentru a nu încărca programul am eliminat temporizările de la apăsarea și ridicarea unei taste.

● Rutina din listing se distinge principal de cea prezentată, prin faptul că *baleierea coloanelor nu începe mereu de la coloana 0, ci aleator*.

Am folosit acest prilej pentru a *exemplifica o instrucțiune mai rar folosită LD A,R*, care în acest caz este folosită *ca generator de numere aleatoare*. Știm deja că fiecărui număr de coloană trebuie să i se asocieze și un octet de baleiere care conține un singur "0". În rutina prezentată în listing această asociere am rezolvat-o prin generarea unei tabele **KEYTAB** în care din doi în doi pași se regăsește de octetul baleiere și un cod de tastă aferent. Informația din **KEYTAB** se citește o singură dată, la apelarea rutinei după ce s-a generat numărul aleator pentru începerea baleierii.

Adresarea elementelor din tabelă se face adunînd la adresa de bază (**KEYTAB**) un deplasament calculat din numărul de coloană : $d = i * 2$, datorită faptului că *tabela este structurată pe doi octeți*.

Structura tabelii **KEYTAB** este deci :

KEYTAB : octet de baleiere 0

cod posibil 0

+ 2x1 octet de baleiere 1

cod posibil 1

.

.

.

+ 2x7 octet de baleiere 7

cod posibil 7

Tehnica folosită, cea de a începe baleierea la valori aleatoare nu aduce aproape *nici un câștig* în cazul unei tastaturi, la care *fenomenele sînt foarte lente* în raport cu viteza de lucru a procesorului. Am prezentat-o totuși datorită faptului că în aplicații, mai rapide ea ar putea fi de folos, precum și fiindcă ne-a permis prezentarea unor tehnici de programare suplimentare.

● Revenind la ideea de a declanșa baleierea coloanelor din punct fix rutina prezentată în listing se poate scurta sensibil, fără a se pierde din explicitatea ei.

Vom avea :

INKEY	: PUSH	HL
	PUSH	BC
	LD	C,SYSIN
	LD	B,0FEH
CYCLE	: LD	L,0FFH
NXTLIN	: IN	A,(C)
	CPL	
	AND	MASK1
	JR	Z,NOKEY
	LD	B,SHIFTN

SHIFTA	: DEC	B
	JR	Z, TSHIFT
	SRL	A
	JR	SHIFTA
TSHIFT	: CP	3
	JR	NZ, OTASTA
	DEC	A
OTASTA	: ADD	A, L
	PUSH	AF
	LD	BC, SYSIN
ASTEPT	: IN	A, (C)
	CPL	
	AND	MASK1
	JR	NZ, ASTEPT
	POP	AF
	POP	BC
	POP	HL
	RET	
NOKEY	: INC	L
	INC	L
	RLC	B
	JR	C, NXTLIN
	JR	CYCLE

Astfel s-ar reduce necesarul de memorie a rutinei de la **82** octeți la **50**.

Îl invităm pe cititor să încerce să îmbunătățească rutina prezentată, reducându-i și mai mult lungimea, căci se poate.

Încheiem prezentarea tastaturii sintetizînd *necesarul hardware pentru interfațarea tastaturii*:

- un șir de 8 diode (izolatoare) + 2 rezistențe
- 2 biți pe portul de intrare **SYSIN**

12.1.2. Dispozitivul de afișaj

Conform specificației tehnice (punctul C.3.) dispozitivul de afișaj va fi realizat cu 8 *afișoare* constituite din 7 segmente și un punct zecimal (**LED—Light Emitting Diode** — diodă luminiscentă). În fig. 12.5. redăm *structura principală* a unui asemenea element de afișaj.

■ Pentru „*apriinderea*” celor 7 segmente și a punctului sînt prevăzute piciorușe. Dacă pe pin-ul aferent unui segment sau cel al punctului se aplică un semnal TTL de nivel "0" atunci segmentul sau punctul respectiv va lumina. Pentru a semnala *logica negativă*, tuturor semnalelor active în starea "0" le vom atașa prefixul "N" (NSEG0, ..., NDECPPOINT).

■ Segmentele aferente semnalelor care au starea logică "1" rămîn „stînse”.

■ Elementul de afișaj mai este prevăzut cu un *semnal de selecție NSEL*, activ în starea "0" care permite validarea sau inhibarea lui. Dacă **NSEL=0** atunci pe afișor vor putea lumina segmentele conform valorii instantanee a semnalelor

de intrare. Dacă $NSEL=1$ atunci apare starea inhibată : toate segmentele și punctul zecimal vor fi stinse, indiferent de valoarea semnalelor de intrare.

Prezența acestui semnal este importantă, deoarece ea permite baleierea mai multor elemente de afișaj alăturate. Această tehnică, numită și *multiplexare*, reduce necesarul de hardware : nu mai este necesară atașarea unui element de memorare la fiecare LED, ci se va folosi unul singur în care se va înscrie rând pe rând valoarea aferentă fiecărui LED și se va activa LED-ul respectiv. Dacă această comutație se face cu o frecvență suficient de mare, ochiul uman va sesiza imaginea static, ca și cum fiecare afișor ar fi activat în mod continuu.

■ Curentul mediu care parcurge o diodă luminiscentă, și deci și luminozitatea ei, nu depinde de frecvența de baleiere, ci de numărul elementelor acționate. Considerând 8 elemente afișoare, factorul de utilizare a fiecăruia este de $k=1/8$. Pentru a genera o intensitate luminoasă echivalentă unui regim continuu, impulsurile de curent injectate în starea activată a LED-urilor vor trebui să fie de 8 ori mai puternice. Acesta este factorul care limitează numărul de elemente care se pot atașa într-o grupă de baleiere.

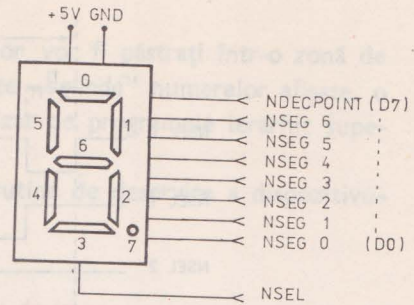
În fig. 12.6 redăm secvența de baleiere.

■ Pentru a minimiza hardware-ul renunțăm la folosirea unui circuit integrat decodificator, care ar decodifica numerele binare în octeți de comandă a segmentelor. Această acțiune va fi implementată pe cale software : pe un port de ieșire (să-l numim LEDPORT) vom genera direct octeții de comandă a segmentelor.

■ Ieșirile portului LEDPORT formează magistrala de afișaj (NSEG0, ..., NSEG6, NDECPOINT) așa cum se arată în fig. 12.7. Această magistrală ajunge la toate elementele de afișaj.

■ Selecția unui element din cele 8 se va face acționând în mod succesiv afișoarele prin semnalele lor de selecție NSEL. Cele 8 semnale de selecție NSEL0 — NSEL7 vor fi ieșirile unui circuit decodificator de tip CDB 442 (SN 74442), care va decodifica un număr binar de 3 bit obținut pe 3 linii ai portului de ieșire sistem (să-l numim SYSOUT).

■ Codul binar de selecție (LED2, LED1, LED0) va fi generat de către microprocesor. Pentru a asigura o baleiere continuă și uniformă, microprocesorului i se vor aplica impulsuri de întrerupere cu o intermitență de 2 ms. În cadrul deservirii fiecărei cereri de întrerupere codul de selecție afișor va fi incrementat cu 1 și va fi emis pe portul SYSOUT pentru a selecta LED-ul următor, iar pe LEDPORT se va emite octetul de activare a segmentelor LED-ului respectiv.



Exemple:

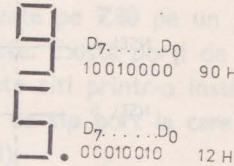


Fig. 12.5. Dioda luminiscentă cu 7 segmente și punct zecimal

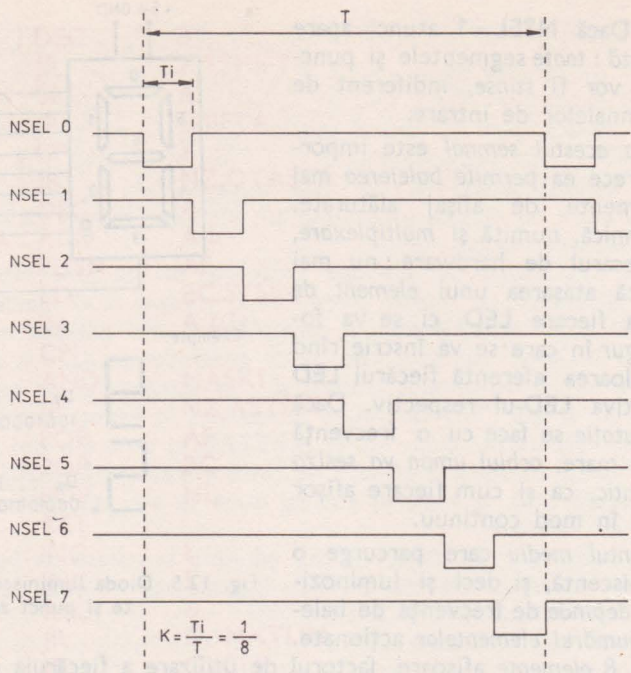


Fig. 12.6. Secvență de baleiere a unui dispozitiv de afișaj format din 8 elemente

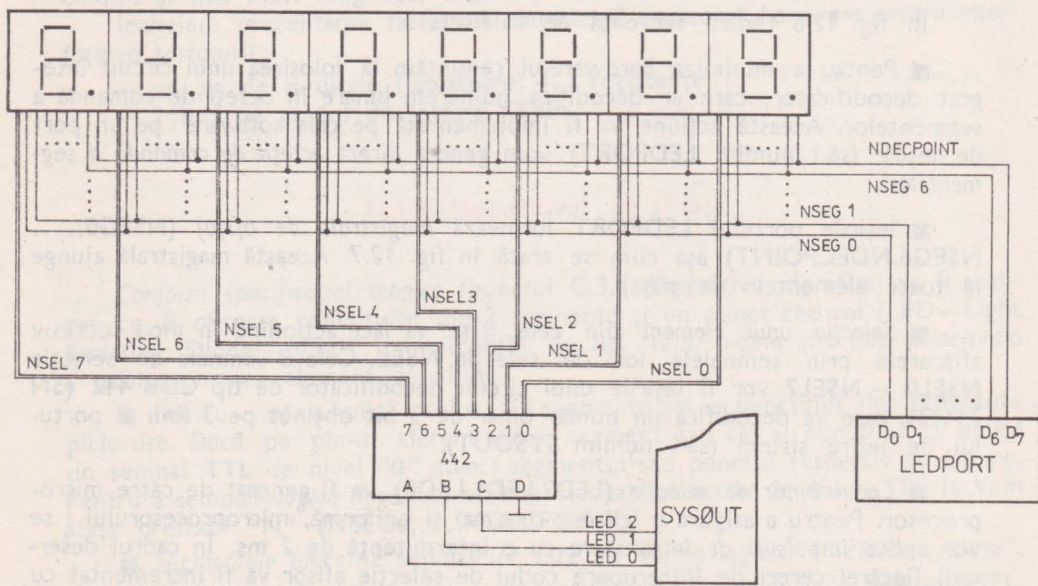


Fig. 12.7. Schema electrică de principiu a dispozitivului de afișaj

■ Cei 8 octeți de activare a segmentelor vor fi păstrați într-o zonă de memorie RAM. Această zonă tampon, care este „oglină” numerelor afișate, o vom numi LEDBUF, conținutul ei fiind actualizat de programele ierarhic superioare ale casei de marcat.

Înainte de a elabora organigrama și apoi rutina de deservire a dispozitivului de afișaj introducem o noțiune nouă :

Celula martor a unui port de ieșire

În majoritatea preponderentă a proiectelor bazate pe Z80 pe un port de ieșire se pot genera 8 semnale de comandă distincte. Există porți de ieșire a căror stare instantanee (port bidirecțional) se poate citi printr-o instrucțiune de tip IN (cazul circuitului PIO) și există alte circuite de tip port la care această manevră nu are efect (port de ieșire unidirecțional).

Ne putem imagina o structură în care anumiți biți ai unui port să aibă o funcție dedicată, iar alții să fie înzestrați cu sarcini total diferite. Acționând unul sau un grup de semnale aferente unui subsansamblu funcțional, modificarea semnalelor de comandă ale unui alt subsansamblu este nedorită, uneori chiar condamnabilă.

Ținând cont că la efectuarea unei instrucțiuni de tip ieșire (OUT, OUTI, etc.) transferul afectează toți biții portului selectați, este necesară constituirea unei celule martor în memoria RAM, celula a cărei conținut va fi permanent identic cu conținutul portului de ieșire tratat.

De aceea, fiecare rutină care tratează un port de ieșire unidirecțional, multi-funcțional, va citi celula martor a portului respectiv, va modifica acei biți care în secvența respectivă prezintă interes, și va rescrie celula martor. Abia după aceea se va efectua transferul octetului în portul dorit.

În cazul proiectului de față am definit deja un port SYSOUT. În configurația acestuia am identificat deja 3 semnale, urmînd ca restul să-l completăm în continuare, în cadrul acestui capitol dedicat definitivării hardware-ului. Portului de ieșire SYSOUT îi atașăm celula martor cu numele WITNESS.

Celălalt port (LEDPOR) deservind un singur subsansamblu funcțional, nu necesită constituirea unei celule martor, deoarece toate semnalele sale vor fi acționate concomitent și ele deservesc un singur modul funcțional.

În fig. 12.8 redăm organigrama rutinei de tratare a întreruperilor, adică baleierea dispozitivului de afișaj.

■ Pentru a minimiza timpul de execuție a acestei rutine, care într-un regim de lucru normal va fi apelată cu o frecvență de 500 Hz, astfel încît viteza de lucru a casei de marcat să nu fie afectată sesizabil, vom păstra variabilele de lucru-indicatorul de adresă din bufferul de afișaj, contorul de selecție, adresa portului de afișaj- în regiștri interni ai microprocesorului.

■ Ca elemente de memorare vom folosi *registri secundari* ai microprocesorului Z80, realizând astfel un *exemplu model* privind utilizarea acestui set de registre.

■ Regiștrii secundari vor fi inițializați la trezirea sistemului. Rutina INT nu va trebui să afecteze nici unul din registre primari.

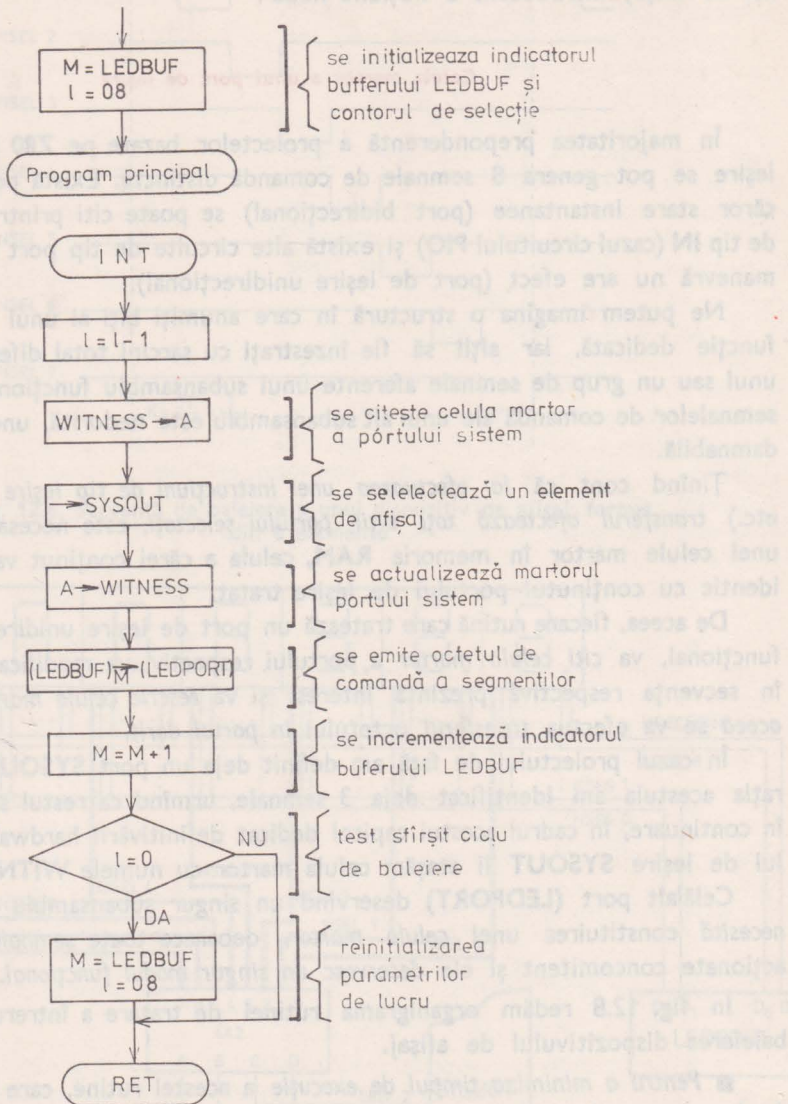


Fig. 12.8. Organigrama rutinei de deservire a întreruperilor (baleierea dispozitivului de afișaj)

Iată rutina :

```
INT :   EX      AF,AF'
        EXX
        OUTI    ;se transferă cuvîntul de comandă
        LD      A,(WITNESS) ;segmente
        AND     MASK2      ;poziționarea celulei martor
        OR      B
        LD      (WITNESS),A
        OUT     (SYSOUT),A ;se selectează LED-ul adecvat
        XOR     A
        OR      B
        JR      NZ,INT1
        LD      HL,LEDBUF ;sfîrșitul unui ciclu de baleiere
        LD      B;BUFLEN
INT1 :   EXX
        EX      AF,AF' ;se reinițializează variabilele de
        EI      ;lucru
        RETI
MASK2 EQU 0F8H
BUFLEN EQU 8
```

Rutina se regăsește în listingul din cap. 17 p. 1—69.

● Reamintim doar faptul că instrucțiunea **OUTI** transferă un octet din memorie de la adresa ind. dată de **HL** în portul de ieșire selectat prin conținutul registrului **C**, incrementează registrul **HL** și decrementează registrul **B**.

● Instrucțiunea **XOR A** șterge conținutul registrului acumulator : $A = 0$.

● Modificînd variabilele **MASK1** și **BUFLEN** rutina poate fi reasamblată pentru un dispozitiv de afișaj de orice lungime cuprinsă între [1,255] cu condiția ca **LED**-urile să admită un curent de vîrf adecvat, care va crește proporțional cu numărul elementelor baleiate.

12.1.3. Imprimanta

Ținînd cont de specificația constructivă, punctul **C.5.**, vom alege o imprimantă de tipul **MIM—40**, care datorită simplității sale constructive va permite, ba chiar mai mult, va impune formarea aptitudinilor noastre în domeniul elaborării de software.

■ *Imprimanta aleasă are un cap de scriere cu 7 ace dispuse pe o verticală. Acele pot fi acționate cu ajutorul a 7 electromagneți. Capul poate fi deplasat pe orizontală cu ajutorul unui motorăș care învîrte un cilindru pe care s-a realizat un canal elicoidal. Dacă motorul se învîrte într-un singur sens, capul de imprimare va efectua o mișcare de dute/vino. Una din cele două curse se efectuează cu viteză aproape constantă, pe parcursul căreia vom efectua scrierea, acționînd prin program acele, astfel încît ele să imprime textul dorit.*

Pentru efectuarea avansului de rînd și a returului de car nu este necesară întreprinderea nici unei activități de comandă. Datorită construcției sale mecanice, imprimanta va efectua la sfîrșitul cursei directe în mod automat returul

de car, însoțit în mod obligatoriu și de un avans de rînd a hîrtiei de imprimare.

La extremitatea dreaptă a cursei capului se află montat un senzor de capăt de cursă. La detectarea semnalului furnizat de acest senzor se poate comanda oprirea motorului de antrenare.

Reținem deci că mișcarea odată lansată, trebuie terminată o mișcare completă dute/vino. De aceea imprimarea se va face rînd cu rînd și nu caracter cu caracter.

Cunoscînd aceste caracteristici putem elabora schema electrică de principiu a interfeței de imprimantă. Ea va fi constituită din 2 blocuri de electronică de forță, unul pentru comanda bobinelor acelor, iar celălalt pentru cuplarea respectiv decuplarea motorului. Interfața de imprimantă va fi deservită de un port de ieșire (să-l numim PRTPORT) și de un bit al portului de intrare SYSIN, amintit deja la interfața de tastatură. Acest din urmă semnal CARLIMIT va permite detectarea pe cale software a capătului de cursă a capului de imprimare (Vezi fig. 12.9.).

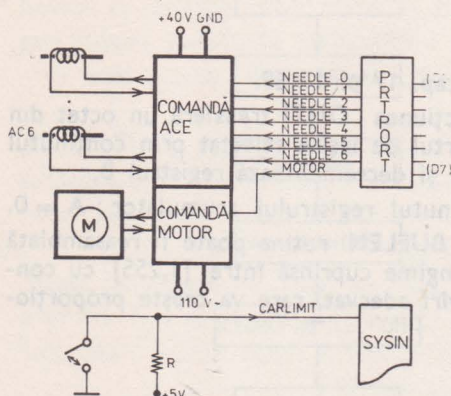


Fig. 12.9. Schema electrică de principiu a interfeței de imprimantă

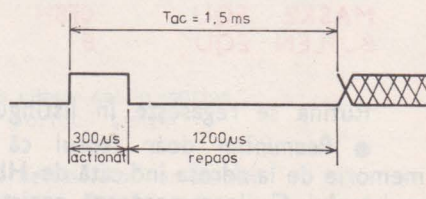


Fig. 12.10. Diagrama de timp pentru acțiunea unui ac

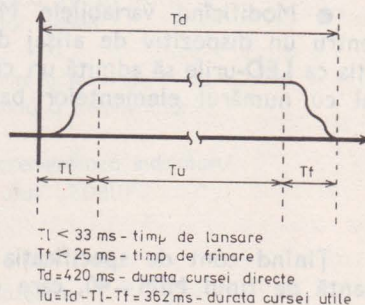


Fig. 12.11. Variația vitezei de deplasare a capului de imprimare pe durata cursei directe

Cunoscînd faptul că intervalul minim dintre 2 acțiuni succesive ale unui ac este de 1,5 ms (vezi fig. 12.10) precum și faptul că durata cursei utile a capului de imprimare (vezi fig. 11.11) este de 362 ms rezultă că vom putea imprima aproximativ 40 de caractere a 6 coloane.

$$N = \frac{T_u}{T_{ac} * N_{col}} = \frac{360}{1,5 * 6} = 40 \quad (12.7)$$

■ Reamintindu-ne că imprimarea trebuie făcută pe două fișii de hîrtie (specificația tehnică C.5.) lungimea oricărei linii de pe bonuri nu va putea depăși 18 caractere. Vom putea rezerva astfel o lățime de aproximativ 4 caractere ca spațiu între cele două fișii.

Putem trece atunci la elaborarea programelor ce vor deservi imprimanta.

În fig. 12.12 redăm organigrama rutinei de imprimare a unui rînd. Rutina PRTLINE va imprima de două ori același text, cuprins într-o zonă tampon numită PRTBUF.

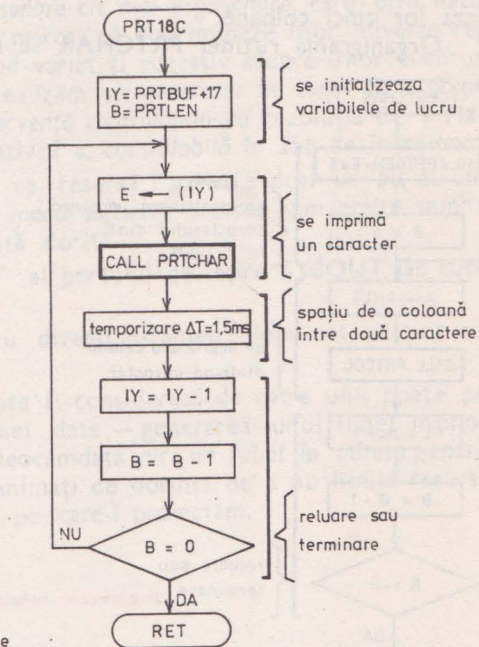
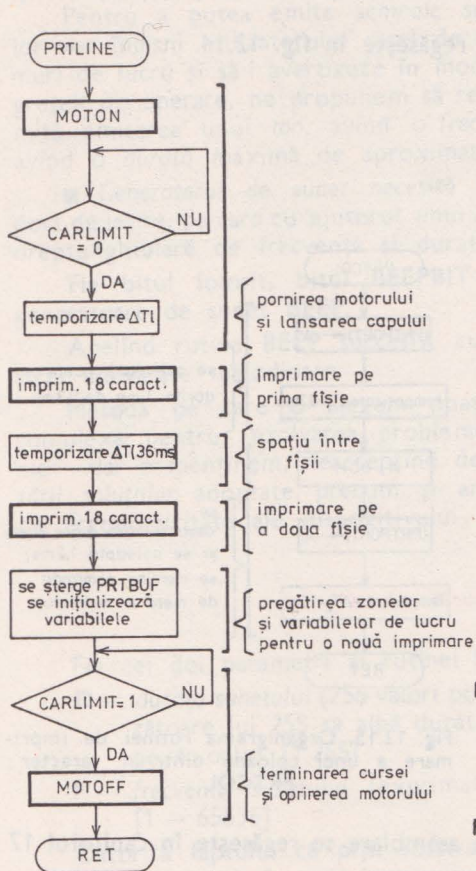


Fig. 12.13. Organigrama rutinei de imprimare a unei fișii : PRT18C

Fig. 12.12. Organigrama rutinei de imprimare a unui rînd : PRTLINE

Elaborarea unui program în limbaj de asamblare pentru implementarea unei astfel de rutine nu poate reprezenta nici o problemă. Ea se regăsește în listingul sursă (cap. 17 p. 1—62). Amintim că între terminarea imprimării celei de-a doua fișii și a comenzii de oprire a motorului se poate intercala lejer rutina de reinițializare a bufferului de imprimare, deoarece capul de imprimare execută între timp cursa inversă.

Imprimarea unei fișii este rezolvată de rutina PRT18C a cărei organigramă se găsește în fig. 12.13.

Programul aferent se regăsește în listing (cap. 17 p. 1—63).

■ Imprimarea unui caracter se realizează cu ajutorul rutinei **PRTCHAR**, care primește codul caracterului de imprimat în registrul **E**. Fiecărui caracter imprimabil i se rezervă cinci octeți într-o tabelă, numită generator de caractere pentru imprimantă (**PRTGEN**). Fiecare octet conține o mostră de biți specifică acelei coloane. Biții care au valoarea "1" vor imprima un punct. Știind că bitul cel mai puțin semnificativ (D_0) acționează acul superior, iar D_6 acul inferior, se poate genera conținutul generatorului de caractere așa cum este descris în cap. 13.

■ Descrierea caracterelor imprimabile este făcută în **PRTGEN**, în ordinea crescătoare a codurilor caracterelor. Rutina **PRTCHAR** va localiza, pe baza codului din **E**, adresa de început a secvenței de 5 octeți aferente și va imprima pe baza lor cinci coloane.

Organigrama rutinei **PRTCHAR** se regăsește în fig. 12.14.

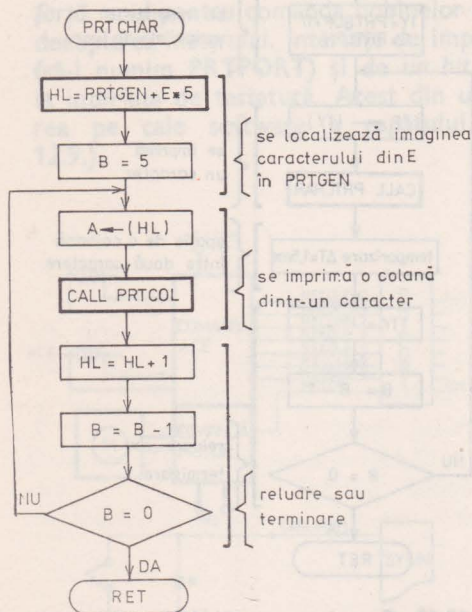


Fig. 12.14. Organigrama rutinei de imprimare a unui caracter : **PRTCHAR**

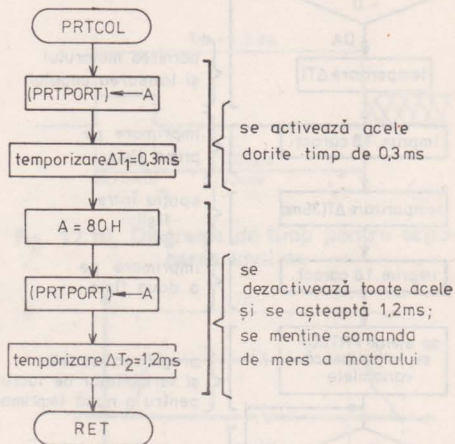


Fig. 12.15. Organigrama rutinei de imprimare a unei coloane dintr-un caracter : **PRTCOL**

Lista programului scris în limbaj de asamblare se regăsește în capitolul 17 pag. 1—64.

■ În sfârșit, imprimarea unei coloane nu ridică nici o problemă deoarece sarcina ei este doar de a activa și a dezactiva acele dorite și de a efectua cele 2 temporizări necesare, conform diagramei de timp din fig. 12.10.

Organigrama rutinei **PRTCOL** se regăsește în fig. 12.15.

Lista programului se regăsește în cap. 17 pag. 1—65.

■ **PRTCOL** folosește ca și celelalte rutine ierarhic superioare o rutină de temporizare de bază, **DELAY**, care realizează temporizări egale cu un multiplu întreg al celei mai mici cuante de timp, semnificativă în imprimantă : **BASEDEL** = 300 μs.

■ Împreună cu rutinele de pornire și oprire a motorului de antrenare a capului de imprimare (**MOTON** și **MOTOFF**), rutina **DELAY** se regăsește în capitolul 17 pag. 1—65, 1—66.

12.1.4. Generatorul de sunet

Pentru a putea emite *semnale sonore cât mai ergonomice*, care-prin natura lor-vor înlesni utilizatorului casei de marcat să discrimineze ușor diverse regi-muri de lucru și să-l avertizeze în mod variat și sugestiv asupra unor eventuale greșeli de operare, ne propunem să realizăm un *generator de sunet* care să per-mită emiterea unui *ton*, avînd o *frecvență controlabilă* cu rezoluția de 1 Hz și avînd o *durată* maximă de aproximativ 2 s, *controlabilă* în 256 de incremente.

■ *Generatorul de sunet* necesită ca resursă hardware doar un *bit* al unui port de ieșire, pe care cu ajutorul unui *modul software dedicat* vom emite semnale dreptunghiulare de frecvență și durată dorită.

Fie bitul folosit, bitul **BEEPBIT** al portului de ieșire **SYSOUT** iar rutina generatoare de sunet **BEEP**.

Apelînd rutina **BEEP** succesiv, cu diverși parametri de sunet, se pot con-strui adevărate melodioare.

Metoda pe care o alegem poate fi considerată de către unii poate prea complexă pentru rezolvarea problemei date — generarea unui sunet monofonic — dar o menținem, neacceptînd deocamdată *nici un rabat la adresa generali-tății soluțiilor* adoptate, precum și animați de dorința de a nu limita resursele de ergonomie ale dispozitivului pe care-l proiectăm.

12.1.4.1. Modelul matematic

Fie cei doi parametri ai rutinei **BEEP** :

D — *durata sunetului* (256 valori posibile, astfel ca valoarea maximă corespun-zătoare lui 255 să aibă durata de aproximativ 2 s.) — se va reprezenta pe 1 octet [1,255]

F — *frecvența sunetului* (exprimată în Hz — se va reprezenta pe 2 octeți [1 — 65536])

Datorită faptului că prin software este relativ ușor a se genera impulsuri de durată controlată, vom transpune al 2-lea parametru **F** în perioada : **T**.

Temporizarea egală cu o semiperioadă **T/2** o vom realiza folosind un ciclu de întârziere de bază, avînd durata de **TCYCLE**.

Din considerente de implementare (modul de realizare a ciclului de întârziere de bază ; reprezentarea aleasă pentru parametri folosiți) *ne propunem ca TCYCLE să fie egal cu durata a 50 de tacti procesor.*

$$TCYCLE = 50 * TCLOCK \quad (12.8)$$

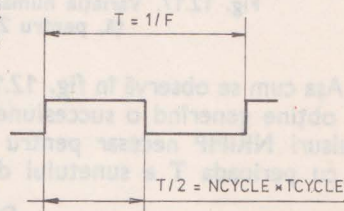


Fig. 12.16. Perioada **T** a semnalului de frecvență **F**

În cazul unui microprocesor Z80 excitat de o frecvență de tact de 2,5 MHz, cum este și cazul calculatorului PRAE, $TCYCLE=20 \mu s$. Această valoare permite teoretic generarea unor semnale cu frecvență limită superioare egală cu 50 KHz, valoare oricum neinteresantă în cazul nostru.

Numărul de repetări necesare ale ciclului de temporizare de bază $TCYCLE$, pentru a obține un semnal de frecvență F , va fi deci :

$$NCYCLE = \frac{T}{2 \cdot TCYCLE} = \frac{1}{2 \cdot TCYCLE \cdot F} = \frac{1}{100 \cdot TCLOCK \cdot F} \quad (12.9)$$

Valoarea $NCYCLE$ se va recalcula la fiecare apelare a rutinei BEEP. Pentru a reduce timpul de lucru al procesorului, precalculăm expresia (12.9.) : putem efectua o împărțire și o înmulțire calculând constanta K .

$$K = \frac{1}{100 \cdot TCLOCK} \quad (12.10)$$

Dorind să acordăm un nume mai sugestiv acestei constante, observăm că ea reprezintă întocmai numărul $NCYCLE$ de temporizări de bază pentru obținerea unui semnal de frecvență 1 Hz.

Rezultă deci :

$$ONEHZ = \frac{1}{100 \cdot TCLOCK} \quad (12.11)$$

În cazul calculatorului PRAE—M ($TCLOCK=400 ns$) $ONEHZ=25000$ iterații. Înainte de a se emite un sunet, rutina BEEP va calcula deci parametrul $NCYCLE$ aferent.

$$NCYCLE = \frac{ONEHZ}{F} \quad (12.12)$$

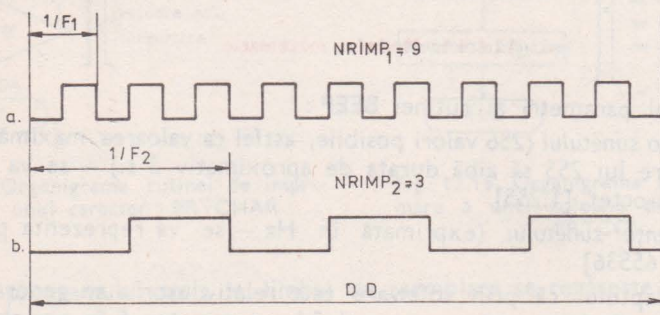


Fig. 12.17. Variația numărului de impulsuri în funcție de frecvență, pentru 2 sunete de durată identică

Așa cum se observă în fig. 12.17 durata DD a unui semnal sonor de frecvență F se obține generând o succesiune de impulsuri având perioada $1/F$. Numărul de impulsuri $NRIMP$ necesar pentru obținerea duratei DD este invers proporțională cu perioada T a sunetului de frecvență F .

$$NRIMP = \frac{DD}{T} = \frac{DD}{TCYCLE \cdot NCYCLE} \quad (12.13)$$

$$K1 = \frac{DD}{TCYCLE} \text{ — este o variabilă proporțională cu durata sunetului dorit.}$$

O vom transforma astfel încât să calibrăm durata sunetului conform cu enunțul problemei. De aceea și din considerente de implementare (simplitate și volum de calcul cât mai redus) înlocuind **K1** cu

$$TIME = D \cdot 128 \tag{12.14}$$

unde $D \in [1,255]$ este parametrul de durată folosit la apelarea rutinei **BEEP**. Din (12.13) și (12.14) rezultă :

$$NRIMP = \frac{TIME}{NCYCLE} \tag{12.15}$$

Folosind expresiile (12.12), (12.14), (12.15) cei doi parametri interni, **NCYCLE** și **NRIMP** pot fi determinați direct din parametri externi **F** — frecvența și **D** — durată. Vom adopta aceste expresii pentru a *minimiza necesarul de rutine aritmetice*. Folosirea unei împărțiri este oricum inevitabilă pentru rezolvarea problemei, iar înmulțirea cu 128 este în aritmetica binară de-a dreptul banală.

12.1.4.2. Algoritm

Pe baza celor expuse mai sus, algoritmul de generare a sunetului se oferă de la sine.

- Rutina **BEEP** va primi la apelare frecvența **F** și durată **D** a sunetului.
- Se calculează variabilele :

TIME, NCYCLE și NRIMP

după care se trece la emiterea sunetului.

ată organigrama algoritmului de implementat.

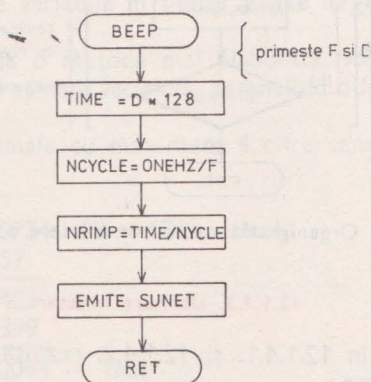


Fig. 12.18. Organigrama rutinei **BEEP**

Emiterea sunetului se poate concepe cum urmează :

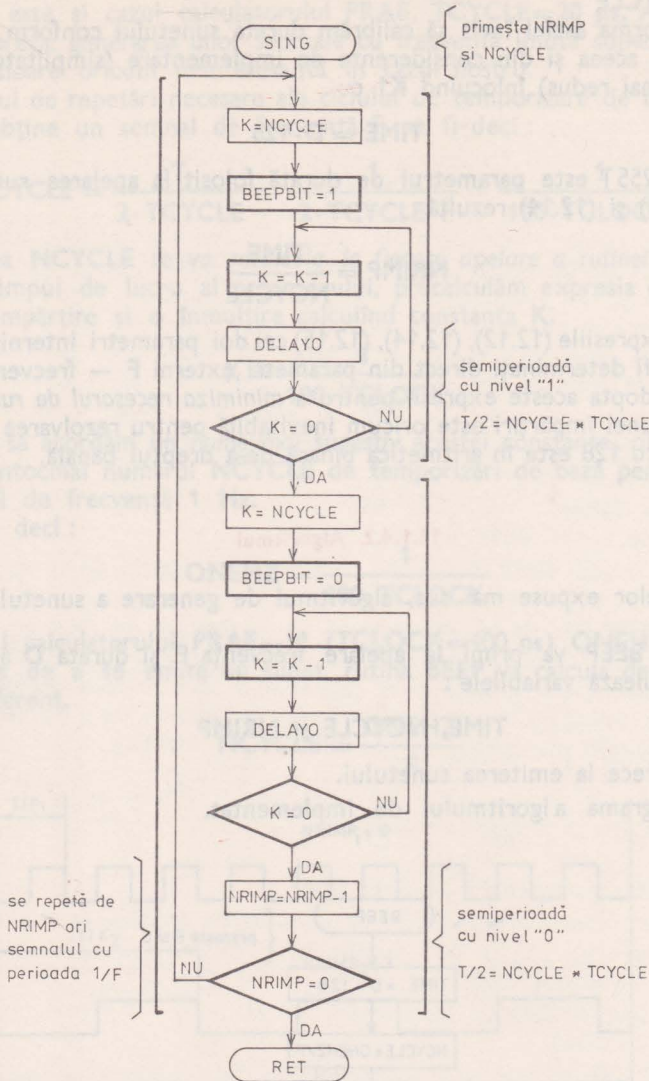


Fig. 12.19. Organigrama rutinei de emitere a sunetului : SING

12.1.4.3. Utilitare necesare

Din cele expuse în 12.1.4.1. și 12.1.4.2. rezultă necesitatea elaborării a 2 rutine de bază și anume :

- rutină de temporizare care să dureze o semiperioadă : $T/2 = NCYCLE \cdot TCYCLE$, unde $TCYCLE$ să dureze 50 de tați procesor. O vom numi : DELAYO.

- rutina de împărțire a două numere binare exprimate pe 2 octeți, cu posibilitatea de rotunjire. O vom numi : DIVIDE.

Rutina DELAY0

■ Rutina va asigura decrementarea unei variabile **NCYCLE** de la o valoare inițială la 0, astfel încât fiecare iterație să dureze 50 de tați procesor. Gama de valori a variabilei **NCYCLE** este cuprinsă în plaja

$$\text{NCYCLE} = 1,25 \text{ pentru } F = 20.000 \text{ Hz}$$

$$\text{NCYCLE} = 1250 \text{ pentru } F = 20 \text{ Hz}$$

Pentru reprezentarea acestei variabile este necesară utilizarea unui registru dublu avînd 16 biți.

Se oferă următoarea secvență :

```
DELAY0 :                               ; DE = NCYCLE
        DEC        DE                   ; 6
        JR        DELAY1                ; 12 — salt de temporizare
DELAY1 : JR        DELAY2                ; 12 — idem
DELAY2 : LD        A,E                   ; 4 — test
        OR        D                     ; 4 — pentru
        JR        NZ,DELAY0             ; 12 — DE = 0?
        RET                               ; Total 50 tați procesor
        ; 10
```

În comentarii am marcat numărul de tați ai fiecărei instrucțiuni folosite.

Rutina DIVIDE

Pentru împărțirea a două numere binare se pot imagina mulți algoritmi. Cazul cel mai simplu ar fi cel de a efectua scăderi succesive a împărțitorului din numărul de împărțit, pînă la obținerea unui număr negativ. Contorizînd scăderile se obține cîtul împărțirii. Această metodă are dezavantajul de a fi banală, și de a avea o durată de execuție variabilă în limite foarte largi, în funcție de numărul scăderilor de efectuat.

Pentru a putea aborda o metodă mai elaborată, vom reconsidera în cele ce urmează, împărțirea a două numere zecimale, semnalînd diferențele între aritmetica binară și cea zecimală.

Fie două numere zecimale cu maximum 4 cifre semnificative : 9734 și 0057

Efectuăm împărțirea :

$$\begin{array}{r} 9734 : 57 = 170 \text{ — cîtul} \\ \underline{57} \\ 403 \\ \underline{399} \\ 0044 \text{ — rest} \end{array}$$

Procedura de sus, cunoscută tuturor, este prezentată doar pentru a fi criticată : ea beneficiază din plin de inteligența umană.

Primul și poate cel mai important pas îl reprezintă calibrarea celor 2 numere : oricine poate vedea în acest exemplu, că prima secvență de numere care poate

fi împărțită cu 57 este 97. Se observă de asemenea la prima vedere, că cifra care reprezintă cîțul operației a $403 : 57$ este 7.

Această ușurință o vom avea ori de câte ori vom dori să împărțim două numere concrete. Dacă dorim în schimb să împărțim două numere oarecare

$$a_3 a_2 a_1 a_0 : b_3 b_2 b_1 b_0$$

unde a_{0-3} și b_{0-3} pot fi orice cifră zecimală cuprinsă între 0 și 9, va trebui să elaborăm un algoritm mai mecanic.

Să considerăm următorii regiștri :

REST — pentru generarea restului împărțirii

DEIMP — conține inițial deîmpărțitul la sfîrșitul operației el va conține cîțul

IMP — conține împărțitorul

Toți acești regiștri au aceeași lungime (număr de cifre semnificative) în cazul nostru 4. Pentru ușurință completăm și zerourile ne semnificative. Registrul **TEMP** este un registru cu o singură cifră semnificativă care va conține cîțul unei împărțiri intermediare. Valoarea acestuia va fi totdeauna cuprinsă între 0 și 9.

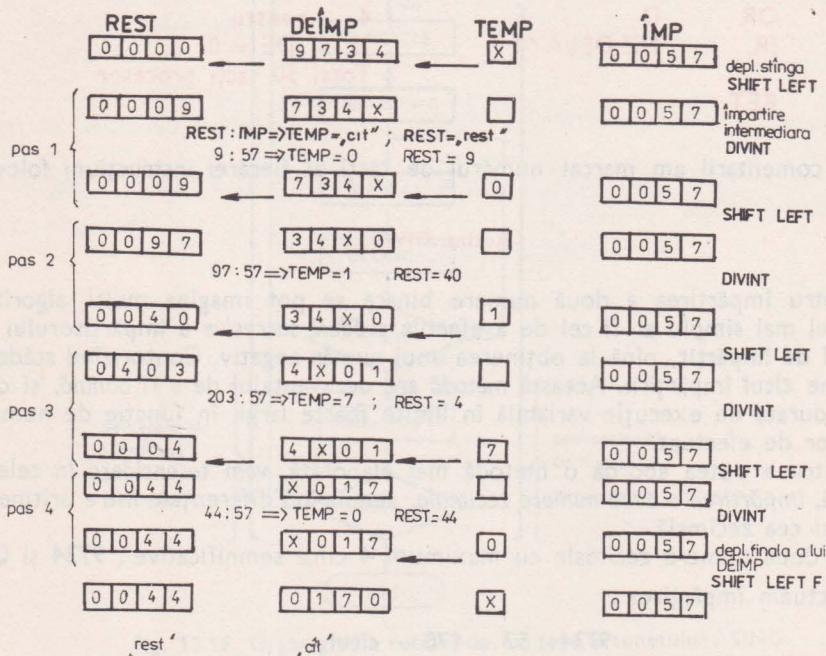


Fig. 12.20. Ilustrarea pașilor algoritmiizați ai unei împărțiri zecimale (numere cu 4 cifre semnificative)

Am obținut astfel pe cale intuitivă o metodă ușor algoritimizabilă pentru împărțirea a două numere. Singura operație care ar putea ridica probleme la implementare este procedeul de efectuare a unei împărțiri intermediare. Ținând însă cont de faptul că raportul celor două numere implicate în această împărțire intermediară este întotdeauna mai mic decât un ordin de mărime, rezultă că în arit-

metica binară problema poate fi rezolvată printr-o simplă scădere (Singurele cifre „cît” posibile fiind 0 și 1 nu există posibilitatea apariției unei cereri de iterație.)

Sintetizăm operațiile efectuate :

SHIFT LEFT : această operație *deplasează* cu o poziție *la stînga* conținutul regiștrilor **DEIMP** și **REST**, astfel ca cifra cea mai semnificativă din **DEIMP** să treacă pe poziția cea mai puțin semnificativă a lui **REST**.

În poziția cea mai puțin semnificativă a lui **DEIMP** este avansată cifra conținută în regiștrul **TEMP**.

DIVINT : este o *împărțire intermediară* între două numere a căror raport este totdeauna mai mic decît un ordin de mărime.

Se împarte **REST** cu **IMP**. Cîtul operației (o cifră) se depune în **TEMP**, iar restul în regiștrul **REST**.

SHIFT LEFTF : este o *deplasare la stînga finală*. Afectează doar **DEIMP** și **TEMP**. Scopul este de a genera "cît-ul" final al împărțirii. Conținutul lui **DEIMP** este rotit prin **TEMP** la stînga. Astfel ultimul *cît* intermediar se transferă din **TEMP** pe poziția cea mai puțin semnificativă a lui **DEIMP**, care devine astfel regiștru rezuiz. z .

Putem construi organigrama din Fig. 12.21.

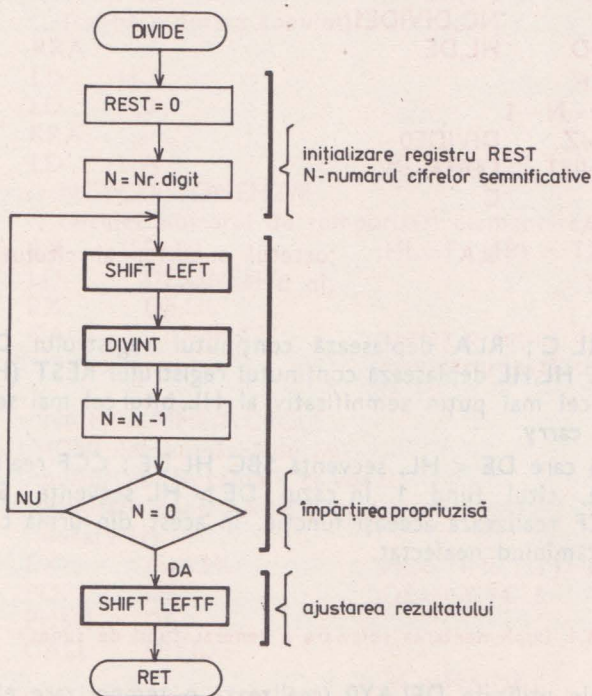


Fig. 12.21. Organigrama rutinei de împărțire : DIVIDE

Transpunerea acestui algoritm în limbaj de asamblare, pentru două numere a câte 16 biți oferă, așa cum se va vedea, mult spațiu creativității și ingeniozității.

- Fie **A** și **C** deîmpărțitul (**A** octetul mai semnificativ).
- Fie **D** și **E** împărțitorul (**D** octetul mai semnificativ).
- Vom realiza o rutină care generează câtul împărțirii **AC/DE** în registrul **BC**, restul obținându-se în **HL**.
- Rolul registrului **TEMP** folosit în exemplul de mai sus este preluat de flagul **Cy** (**carry**).

Iată rutina :

```

DIVIDE :
        LD     A,H           ;transfer deîmpărțitul în
        LD     C,L           ;registri de lucru A și C
        LD     HL,0          ;inițializez restul
        LD     B,10H
        ;SHIFT LEFT

DIVIDE0 :
        RL     C
        RLA
        ADC   HL,HL
        ;DIVINT
        SBC   HL,DE
        JR    NC,DIVIDE1
        ADD   HL,DE

DIVIDE1 :
        CCF
        ;N=N-1
        DJNZ  DIVIDE0
        ;SHIFT LEFT
        RL
        RLA
        LD   B,A           ;octetul superior al câtului
        RET                ;în B
    
```

● Secvența **RL C ; RLA** deplasează conținutul registrului **DEIMP** (**AC**) la stînga, iar **ADC HL,HL** deplasează conținutul registrului **REST** (**HL**) la stînga, înserînd pe bitul cel mai puțin semnificativ al **HL**, bitul cel mai semnificativ al **AC**, aflat acum în **carry**.

● În cazul în care **DE < HL**, secvența **SBC HL,DE ; CCF** realizează împărțirea intermediară, câtul fiind 1. În cazul **DE > HL** secvența **SBC HL, DE ; ADD HL,DE ; CCF** realizează aceeași funcție. În acest din urmă caz câtul este 0, restul din **HL** rămînînd neafectat.

12.1.4.4. Implementarea software a generatorului de sunet

Avînd modulele utilizate **DELAY0** (realizează o temporizare elementară de 50 tacți procesor) și **DIVIDE** (realizează împărțirea a două numere binare de 16 biți) constituite, putem trece la elaborarea rutinei **BEEP**.

■ Vom prelua cu mici modificări, rutina elaborată de către mat. Kiss Alexandru, și implementată pe calculatorul PRAE—M ca funcție BEEP în PRAE—BASIC V3.5.

Folosind organigrama din fig. 11.18, programul se scrie ușor :
BEEP :

```

; subrutină generatoare de sunet
; după PRAE BASIC V3.5
;
; Parametri de intrare :
;   HL — conține frecvența F în Hz
;   A  — conține un număr proporțional cu durata D
;
; registre BC,DE, IX, IY — rămân neafecți
;
; salvez registre
PUSH BC
PUSH DE
PUSH HL

;șterg Cy
SCF
CCF
; TIME=D * 128
; (calibrez durata tonului)
RRA
LD H,A
LD A,0
RRA
LD L,A ;HL=A * 128
; NCYCLE=ONEHZ/F
; calculez numărul de temporizări elementare
EX (SP),HL ;HL=F, (SP) = TIME
LD DE,ONEHZ
EX DE,HL
CALL COMPUTE ;apelez o rutină de împărțire
;care returnează citul rotunjit
;în HL
;NRIMP=TIME/NCYCLE
;(Calculez numărul de im-
;pulsuri cu frecvența F, ce vor fi
;generate pentru a se obține
;durata D)
EX (SP),HL ;HL=TIME,(SP)=NCYCLE
POP DE ;DE=NCYCLE
PUSH DE
CALL DIVIDE
;BC=NRIMP
INC BC ;evit BC=0
;emit sunetul
POP DE

```



```

CALL    SING                ;emite NRIMP (BC) impulsuri
                                ;cu durata dată de NCYCLE(DE)
POP     DE
POP     BC
RET

```

■ Folosind rutina **DELAY0** elaborată la § 12.1.4.3. se poate scrie rutina **SING** a cărei organigramă se află în fig. 12.19.

SING :

```

                ;subrutină de emitere a sunetului
                ;Parametrii de intrare :
                ;      BC = NRIMP
                ;      DE = NCYCLE
                ;
SING1 : PUSH    DE                ;salvez NCYCLE
LD          A,(WITNESS)          ;13 folosesc celula martor
SET        BEEPBIT,A            ; 8 poziționez 1
POP        DE                    ;10
PUSH      DE                    ;11
OUT        (SYSOUT),A           ;11 emit 1
CALL      DELAY0                ;17+DE*50+10
LD        A,(WITNESS)          ;13
RES       BEEPBIT,A            ; 8 poziționez 0
POP        DE                    ; 10
PUSH      DE                    ; 11
OUT        (SYSOUT),A           ; 11 emit 0
CALL      DELAY0                ; 17+DE*50+10
DEC       BC                    ; 6
LD        A,C                    ; 4 NRIMP=NRIMP-1
OR        B                      ; 4
JP        NZ,SING1              ; 10
POP        DE                    ; restaurez stiva
RET

```

● Bucla program care emite **NRIMP** impulsuri, este cuprinsă între linia **SING1** : și instrucțiunea **JP NZ,SING1**. În dreapta fiecărei linii program am notat numărul de tați procesor aferenți instrucțiunii respective. Se observă că pe lângă temporizările generate în rutina **DELAY0** (proportionale cu **NCYCLE**), apar întârzieri iminente datorită vehiculării datelor. Frecvența sunetului emis va fi astfel mai mică decât cea teoretică. În § 12.1.4.5. vom prezenta abaterea frecvenței reale de cea teoretică.

■ Ultima problemă de rezolvat a rămas elaborarea rutinei **COMPUTE**, care va efectua o împărțire binară de 16 biți și va rotunji câtul.

Vom folosi rutina **DIVIDE** elaborată în § 12.1.4.3.

COMPUTE :

```

                ; rutină de împărțire cu rotunjirea câtului
                ; parametri de intrare :
                ;      HL — deîmpărțitul
                ;      DE — împărțitorul

```

```

; parametri de ieşire :
; HL — cîtul rotunjit
CALL    DIVIDE
PUSH    BC           ; salvez cîtul nerotunjit
ADD     HL,HL        ; dublez restul
CALL    COMP         ; dacă restul dublat
POP     HL
RET     C            ; este mai mic decît împărţitorul,
; atunci cîtul nu se schimbă
INC     HL           ; altfel, el este incrementat,
; rotunjindu-se, rezultatul
RET

```

COMP :

```

; rutina compară regiştri DE şi HL
; nu afectează nici un registru decît F
OR      A            ; Cy=0
PUSH    HL
SBC     HL,DE
POP     HL
RET     ; Cy=1 dacă DE > HL

```

■ Pentru ca modulul **BEEP** să fie complet, vom specifica valorile constantelor. Unde se poate le vom defini prin expresii, lăsînd evaluarea lor pe seama asamblorului. Conferim astfel un plus de elasticitate modulului realizat.

```

TCLOCK    EQU    400           ; în nanosecunde
ONEHZ     EQU    50000/TCLOCK * 200
EXTERNAL  BEEPBIT,SYSOUT,WITNESS ; aceste constante se
; definesc în alt modul software,
; ierarhic superior.

```

12.1.4.5. Studiul abaterii frecvenţei reale de cea teoretică

Din implementarea enunţată rezultă faptul că frecvenţa sunetului emis, diferă de valoarea F_T stabilită la apelul rutinei **BEEP**. Ne propunem să determinăm valoarea acestei abateri.

Pentru a obţine perioada exactă T_R a sunetului emis în subrutina **SING**, vom însuma numărul de taçti procesor aferenţi fiecărei instrucţiuni ai subrutinei :

$$N = 13 + 8 + 10 + 11 + 11 + 17 + DE * 50 + 10 + 13 + 8 + 10 + 11 + 11 + 17 + DE * 50 + 10 + 6 + 4 + 4 + 10$$

$$N = 182 + 100 * DE$$

unde DE conţine numărul de iteraţii **NCYCLE**.

Rezultă perioada reală

$$T_R = (182 + 100 * NCYCLE) * TCLOCK \quad (12.16)$$

unde **TCLOCK** este perioada tactului procesor, 400 ns în cazul nostru.

Apare deci o abatere de la valoarea teoretică

$$T_T = 100 * NCYCLE * TCLOCK,$$

datorită întârzierilor suplimentare care apar în timpul execuției rutinei **SING**.
Totalul întârzierilor se cifrează la :

$$\Delta T = 182 * TCLOCK \quad (12.17)$$

ΔT reprezintă prima sursă de erori.

Cea de a doua sursă de erori apare la calculul numărului de iterații **NCYCLE**, în care rezultatul împărțirii **ONEHZ/F** este rotunjit.

Dorind să elaborăm un program **BASIC** pentru determinarea erorii relative a frecvenței sunetului emis notăm :

$$NCYCLE = INT(ONEHZ/F+0,5) \quad (12.18)$$

unde **F** este frecvența teoretică impusă la apelul rutinei **BEEP**.

Frecvența reală va fi :

$$F_R = \frac{1}{\Delta T + 100 * INT(ONEHZ/F) * TCLOCK} \quad (12.19)$$

Dorim să stabilim variația erorii relative a frecvenței :

$$\epsilon = \frac{F_R - F_T}{F_T} * 100\% \quad (12.20)$$

■ Următorul program **BASIC**, va stabili eroarea relativă minimă și maximă, și va desena curba de variație a erorii relative a frecvenței pe întregul domeniu audio (20 Hz – 20.000 Hz).

```

10 'Calculul erorii relative a funcției BEEP
20 TCLOCK= 400E-9
30 UNHZ=1/TCLOCK/100
40 DT=182*TCLOCK 'DELTA T
50 MIN=1 : F1=0 : MAX=0 : F2=0 'INITIALIZEZ MINIMUL ȘI
   MAXIMUL
60 DIM EPS (2000) 'ALOC SPAȚII TABELEI CU ERORI
70 I=0
80 CST = 100*TCLOCK 'PENTRU CREȘTEREA VITEZEI DE CALCUL
90 CLS : PRINT "LASAȚI-MA SA MA GINDESC"
100 FOR FT=20 TO 20000 STEP 10
110 EPS(I)= ((1/(DT+CST*INT(UNHZ/FT)))-FT)/FT
120 IF ABS(EPS(I)) < MIN THEN MIN=EPS(I) : F1=FT
130 IF ABS(EPS(I)) > MAX THEN MAX=EPS(I) : F2=FT
140 I=I+1
150 NEXT FT
160 CLS
170 PRINT USING "EROAREA MINIMA###.##% LA FT= ;"MIN*100 : F1
180 PRINT
190 PRINT USING "EROAREA MAXIMA###.##% LA FT= ;"MAX*100 :
   F2
200 INPUT "DORIȚI GRAFICUL ERORII RELATIVE D/N" : A$
210 IF A$="N" THEN END
220 GOSUB 270 'TRASAREA ȘI MARCAREA AXELOR

```

```

230 FOR I=2T02000
240 PLOT 80 * LOG(I/2)/LOG(10), EPS(I) * 400+10
250 NEXT I
260 GOTO 260
270 CLS 'TRASAREA și MARCAREA AXELOR
280 'ABSCISA
290 PLOT 0,10 : DRAW255,10 :DRAW 252,8 :PLOT 255,10 : DRAW252,12
300 FOR I=1 TO4
310 PRINTAT (1+7 * (I-1)) * 8,0 ;2 * 10^I ;
320 PLOT 80 * (I-1),10 :DRAW 80 * (I-1),8
330 NEXT I
340 PRINTAT 212,20 ;"FT" ;
350 'ORDONATA
360 FOR I=1 TO 5
370 PRINTAT 0,40 * I+4 ; 10 * I ;
380 PLOT 2,40 * I+10 : DRAW 0,40 * I+10
390 NEXT I
400 PLOT 0,10 : DRAW0,240 :DRAW2,237
410 PRINTAT 16,248 ; "-EPS%" ;
420 RETURN

```

Executînd acest program obținem abaterile minime și maxime ale funcției.

$\varepsilon_{\min} = -0.14 \%$ la frecvența $f_1 = 20 \text{ Hz}$

$\varepsilon_{\max} = -55.67 \%$ la frecvența $f_2 = 20000 \text{ Hz}$

Reprezentarea abaterii relative de frecvență o facem scalînd logaritmic axa frecvențelor (vezi fig. 12.22).

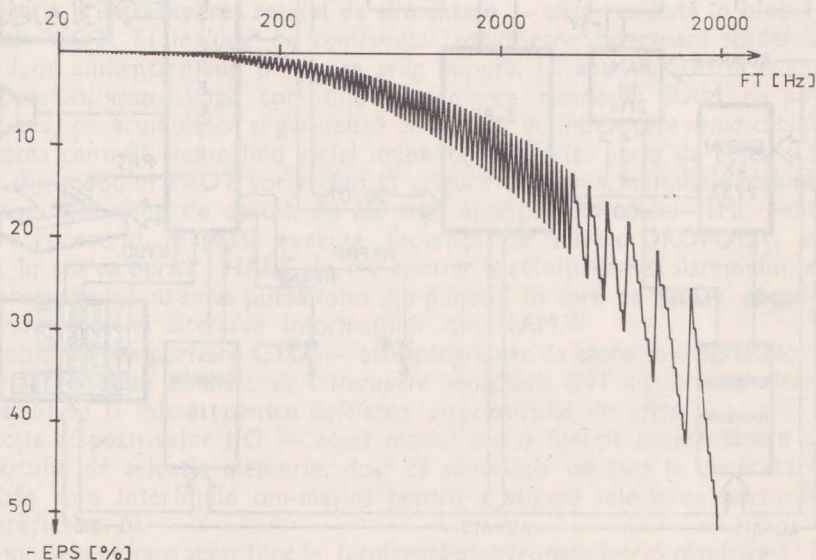


Fig. 12.22. Abaterea relativă a frecvenței generate de BEEP

Putem conchide afirmînd cã generatorul de sunet pe care l-am elaborat este destul de bun în gama frecvențelor joase (20.2000 Hz), gamã în care abaterea relativã este mai micã de 10 %.

12.2. Structura hardware a echipamentului

Sintetizînd cele expuse în paragrafele precedente, vom putea defini structura hardware a echipamentului. Structura o vom organiza pe blocuri funcționale, detalizîndu-le pe alocuri la nivel de semnale electrice. Acesta este momentul în care cele două activități, proiectare hardware și software se despart, urmînd ca ele sã se desfășoare cuasiindependent, pînã la momentul cheie, acela al punerii la punct a echipamentului.

Ușurința cu care se va face joncțiunea, depinde de buna definire a detaliilor comune. În cazul nostru aceste detalii se referã la cantitatea și adresele blocurilor de memorie, precum și la structura circuitelor de intrare/ieșire.

În fig. 12.23 am elaborat schema bloc a casei de marcat.

Analizînd fig. 12.23 putem identifica blocurile funcționale majore ale echipamentului.

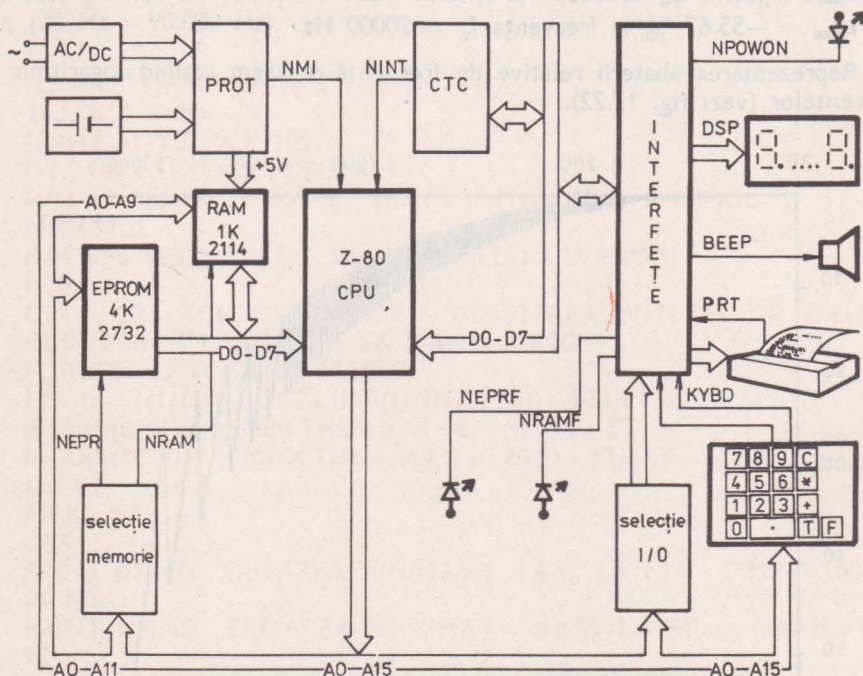


Fig. 12.23. Schema bloc a casei de marcat

1. **Unitatea centrală** — cuprinde microprocesorul **Z80**, amplificatoarele de semnal precum și oscilatorul pilot al sistemului.
2. **Memoria EPROM** — estimată grosier la **4 kbyte**, ea va fi constituită dintr-o singură capsulă de **4 kbyte** : **I2732**. În ea vor rezida toate programele casei de marcat.
3. **Memoria RAM** — estimată la **1 kbyte** va fi realizată cu 2 circuite de memorie NMOS, de tipul **I2114**. În ea vor rezida variabilele de lucru, stiva, zonele tampon ale software-ului și datele memorate de casa de marcat, conform specificațiilor funcționale.

Datorită acestor din urmă date, memoria RAM va trebui să fie nevolatilă, prevăzându-se alimentarea ei de la baterie pentru cazul în care apar întreruperi ale rețelei de curent alternativ.

4. **Selecția memoriei** — format din circuite logice combinaționale, acest bloc funcțional are menirea să decodifice starea magistralei de adrese **A0—A15** și să genereze cele două semnale de selecție :

- **NEPR** — va selecta memoria EPROM în spațiul de adrese (**0,0FFF_H**).
- **NRAM** — va selecta memoria RAM în spațiul de adrese (**8000_H, 83FF_H**).

S-a păstrat intenționat o zonă „moartă” între sfârșitul memoriei EPROM și începutul memoriei RAM, prevăzându-se astfel posibilitatea extinderii ușoare a hardware-ului prin introducerea unei memorii EPROM mai mari (ex. **I2764=8k** sau **I27128=16k**).

Observație

În listingul din capitolul 17 precum și pe caseta **VISIBLE—Z80** programul pe care îl elaborăm împreună este generat pentru spațiul de adrese **7000H, 7FFFH** (partea dedicată pentru EPROM), iar zona RAM este inițializată la adresa **6C00H**. Explicația este simplă : la adresa 0, în ambele calculatoare (**PRAE** și **aMIC**) rezidă EPROM-urile proprii care conțin rutinele sistem și interpretorul **BASIC** al calculatoarelor. La adresa **8000H** se află însăși codul programului **VISZ80**. Menționăm că aceste abateri nu împietesc cu nimic asupra înțelegerii prezentului studiu de caz.

5. **Protecția la întreruperea rețelei de alimentare** — este realizată în blocul funcțional **PROT**. El include un *comparator rapid*, care detectează scăderea tensiunii de alimentare sub o valoare prag impusă. La apariția acestui eveniment *comparatorul basculează*, comutând alimentarea memoriei RAM de pe sursa de rețea, pe acumulator și generează un semnal de întrerupere nemascăbilă către unitatea centrală, semnalînd astfel iminența de avarie. Sursa de rețea și circuitele din modulul **PROT** vor trebui să asigure tensiune, în limitele parametrilor impuși, încă timp de aprox. **50 ms** după apariția semnalului **NMI**, pentru ca microprocesorul să poată executa secvența de salvare **DROPOUT**, intrînd apoi în starea oprită : **HALT**. În caz contrar la reinițializarea sistemului, execuția programului nu se va putea relua din punctul în care ea fusese abandonată, riscîndu-se astfel alterarea informațiilor din RAM.
6. **Circuitul de temporizare CTC** — este programat de către microprocesor astfel încît să genereze semnale de întrerupere mascabile **INT** cu o intermitență de **2 ms**. El va fi folosit pentru baleierea dispozitivului de afișaj.
7. **Selecția dispozitivelor I/O** — acest modul are o funcție asemănătoare cu cea a blocului de selecție memorie, doar că semnalele pe care le generează vor fi dirijate spre interfețele om-mașină pentru a asigura selectarea porturilor de intrare/ieșire.

Semnalele pe care acest bloc le furnizează sînt grupate într-o magistrală. Detalierea lor se poate vedea în fig. 11.24.

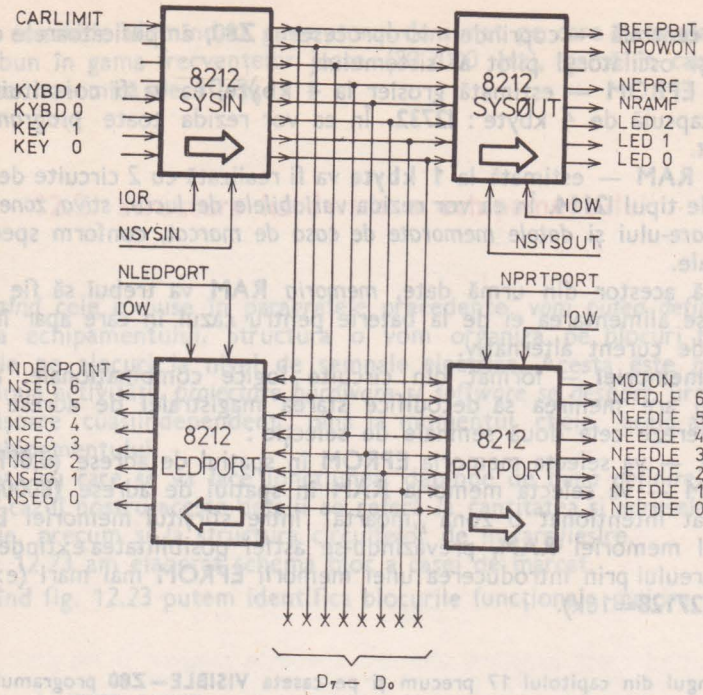


Fig. 12.24. Structura și semnalele circuitelor de intrare/ieșire

- IOR — activ în starea "1" ; semnalează execuția unui ciclu mașină IN.
- IOW — activ în starea "1" ; semnalează execuția unui ciclu mașină OUT.
- NSYSIN — este o valoare decodificată a liniilor A0—A7 și asigură selectarea portului de intrare SYSIN.
- NSYSOUT — este o valoare decodificată a liniilor A0—A7 și asigură selectarea portului de ieșire SYSOUT. Poate fi identic cu SYSIN.
- NLEDPORT — este o valoare decodificată a liniilor A0—A7 și asigură selectarea portului de comandă a dispozitivului de afișaj LEDPORT.
- NPRTPORT — este o valoare decodificată a liniilor A0—A7 și asigură selectarea portului de comandă a imprimantei PRTPORT.

8. INTERFETE — este un *modul complex* care cuprinde pe lângă porturile de intrare/ieșire și circuite electronice formate din componente discrete pentru ajustarea nivelului energetic și de tensiune a tuturor semnalelor care realizează schimbul de informații între UC și periferice.

Din punctul de vedere al obiectivului pe care această carte și-l propune, electronica discretă de interfață este neinteresantă. Vom insista în schimb asupra structurii logice, asupra semnalelor porturilor de I/O.

În fig. 12.24 regăsim cei 4 porți de intrare/ieșire precum și semnalele de comandă și stare pe care le-am prevăzut.

■ **SYSIN** — este **portul de intrare sistem**. Semnalele care pot fi citite prin acest port sînt :

● **KEY0** și **KEY1** sînt cele 2 chei a căror prezență a fost impusă în specificația constructivă (vezi pct. C.8 din § 11.1)

● **KEY0** este cheia operatorului

● **KEY1** — este cheia de control

Ambele semnale sînt active în starea "1"

● **KYBD0** și **KYBD1 (D2 și D3)** — sînt cele două linii ale tastaturii. Apăsarea unei taste se materializează prin apariția unui nivel "0" pe unul din cele două linii. Următorii trei biți ai portului au fost lăsați neutilizați, prevăzîndu-se astfel posibilitatea extinderii liniilor de tastatură, pentru o eventuală dezvoltare a tastaturii pînă la 40 de taste.

● **CARLIMIT (D7)** — este semnalul generat de detectorul capăt de cursă al imprimantei. Este activ în starea "1".

■ **SYSOUT** — este portul de ieșire sistem. Conținutul lui va fi memorat permanent în memoria RAM, în celula martor WITNESS. Semnalele care pot fi generate prin acest port sînt :

● **LED0—LED2 (D0—D2)** — reprezintă contorul de selecție al unuia din cele 8 LED-uri de afișaj. Valorii binare 000 îi corespunde LED-ul din extrema dreaptă, iar valoarea 111 selectează LED-ul din extrema stîngă a dispozitivului de afișaj.

● **NRAMF (D3)** — este un semnal activ în starea "0", el va „aprinde” o diodă luminiscentă dispusă pe circuitul imprimat al unității centrale, semnăind astfel personalului de service eroare la memoria RAM, eroare detectată în secvența de inițializare a casei de marcat.

● **NEPRF (D4)** — este un semnal activ în starea "0", funcția lui este asemănătoare cu cea a semnalului NRAMF, cu deosebirea că el semnălează eroare la memoria EPROM. Următorul bit (D5) s-a păstrat ca rezervă pentru o semnalizare suplimentară, ce se va putea introduce pe viitor.

● **NPOWON (D6)** — este un semnal activ în starea "0"; el va „aprinde” o diodă luminiscentă amplasată pe panoul frontal al casei de marcat, semnăind prezența tensiunii de alimentare. Ba chiar mai mult, operatorul va ști că dispozitivul trăiește, din moment ce pînă la „aprinderea” LED-ului de prezență tensiune, microprocesorul va fi executat o serie întregă de rutine.

● **BEEPBIT (D7)** — este linia pe care se vor genera semnale sonore, formînd pe cale software trenuri de impulsuri de durată și frecvență dorită.

■ **LEDPORT** — este un port de ieșire care asigură comanda dispozitivului de afișaj. Semnalele lui de ieșire sînt cablate la toate elementele de afișaj, formînd astfel magistrala locală a dispozitivului de afișaj. Toate semnalele sînt active în starea "0"

● **NSEG0—NSEG6 (D0—D6)** — reprezintă biții de activare a segmentelor, iar **NDECPOINT** „aprinde” punctul zecimal.

■ **PRTPORT** — este un port de ieșire, avînd rolul de a comanda imprimanta. Toate semnalele lui sînt active în stare "1".

● **NEEDLE0—NEEDLE6 (D0—D6)** activează acele, iar **MOTON** pornește și/sau oprește motorul de antrenare a capului de imprimare.

Odată cu aceste elemente, considerăm terminată definirea structurii hardware a casei electronice de marcat.

Măsura în care această structură va rezista „furtunilor” ce apar pe parcursul proiectării detaliate a hardware-ului și mai ales în faza de punere la punct a echipamentului, depinde în bună măsură de conștiințiozitatea cu care ea a fost elaborată, precum și de experiența proiectantului.

DEFINIREA STRUCTURILOR DE DATE

Avînd hardware-ul definit, vom reanaliza de mai multe ori *specificația funcțională*, pentru a ne putea forma o *imagine de ansamblu* asupra întregului software pe care urmează să-l realizăm. Așa cum am arătat în Cap. 10, *definirea structurilor majore de date, identificarea principalelor variabile* ale unui program, trebuie să precedă demararea elaborării programelor în sine.

Pe baza experienței acumulate se poate afirma că *durata de elaborare și implementare* a componentelor software poate fi *substanțial redusă*, dacă ea este precedată de un efort suplimentar, pentru definirea elementelor sus menționate.

În cele ce urmează vom încerca să *delimităm* toate structurile care se pot *întrezări* pe baza specificațiilor constructive și funcționale, precum și cele impuse de periferice, a căror tratare s-a prezentat în Cap. 12.

La sfîrșitul acestei activități, cea de definire a principalelor structuri de date, vom fi în măsură să *elaborăm o primă schiță a fluxului de date în casa de marcat*, schiță pe care o vom reconsidera la sfîrșitul lucrării, după ce vom fi elaborat întregul software.

13.1. Structuri de bază

Dacă pentru prezentarea celor mai eficienți algoritmi de înmulțire, împărțire, sau a celor de grafică bi-și tridimensională s-au scris zeci de cărți, și bibliografia aferentă structurilor de date este deosebit de bogată. Numim *date*, absolut toate informațiile care deși nu reprezintă program (cod) executabil, sînt incluse într-un program, sau se generează pe parcursul execuției acestuia menționînd că existența lor este absolut necesară pentru programul în cauză. Toate informațiile referitoare la natura datelor, la codul și forma lor de reprezentare, precum și la modul în care ele sînt vehiculate, determină structura datelor.

Noțiunea structurilor de date este o clasă deschisă, care se îmbogățește mereu, putîndu-se imagina o infinitate de reprezentări și manevre. Este greu să aplici clasificări pe mulțimi infinite. Rolul de spirit ordonator al acestei „jungle” îl pot avea, înainte de toate, tipurile de aplicații. Într-adevăr, în jurul lor (le-am putea numi focare) s-au cristalizat și se cristalizează mereu soluții tip. Aceasta este

calea de prezentare aleasă în majoritatea lucrărilor de strictă specialitate, dedicate structurilor de date. O vom urma și noi, prezentînd — în ordinea importanței lor — structurile de date legate de virtuala casă de marcat.

13.1.1. Codurile interne

Înainte de toate vom stabili codurile de reprezentare internă pentru toate caracterele tastaturii, a celor afișabile, precum și a celor care se vor imprima. Călea cea mai rapidă ar fi aceea de-a adapta un set standard. De exemplu, cel mai răspîndit set de coduri, cel **ASCII** (American Standard Code for Information Interchange).

Această soluție ar prezenta avantajul universalității și unei anumite portabilități. Ea ar fi deosebit de interesată în cazul în care casa de marcat ar trebui să fie înglobată într-un sistem informațional complex.

Analizînd specificația dispozitivului de afișaj, a tastaturii, precum și formatul codurilor, ajungem la concluzia că aplicația noastră nu solicită decît un set restrîns din totalitatea caracterelor incluse în standardul **ASCII**. De aceea *ne vom permite să elaborăm un sistem propriu de coduri*.

Sîntem conștienți că această decizie s-ar putea să fie contestată de unii, dar o menținem, dorînd să-i oferim cititorului un exemplu de acțiune în acest sens.

Iată setul de caractere propus, și codurile aferente :

Cod	Car	Cod	Car	Cod	Car	Cod	Car	Cod	Car
00	0	08	8	10	A	18	O	20	spațiu
01	1	09	9	11	C	19	P	21	:
02	2	0A	.	12	D	1A	R	22	—
03	3	0B	FUNC	13	E	1B	S		
04	4	0C	+	14	I	1C	T		
05	5	0D	*	15	L	1D	U		
06	6	0E	TOTAL	16	M	1E	V		
07	7	0F	CLEAR	17	N	1F	Z		

Tab. 13.1. Codurile interne ale casei de marcat

În primele două grupe verticale din Tab. 13.1 se regăesc codurile caracterelor tastaturii. Faptul că, *codurile interne alocate cifrelor 0—9, coincid cu valoarea binară cifrei reprezentate*, nu este o întîmplare. Astfel ne scutim de efortul realizării unei conversii suplimentare pe cale software. Intenționat am acordat următorul cod (0A_H), semnului "." (punct), deoarece el va fi folosit în majoritatea cazurilor ca punct zecimal. Adiacența codului său cu codurile cifrelor va putea ușura pe alocuri implementarea rutinelor de conversie zecimal — binar, binar — zecimal. *Ordinea alocării codurilor pentru celelalte cinci caractere ale tastaturii*

(FUNC, +, *, TOTAL, CLEAR) este *neinteresantă*. Le-am preluat bineînțeles pe acele care rezultă din structura logică a tastaturii, așa cum le furnizează rutina **INKEY**. În continuare am alocat coduri literelor utilizate, așezate în ordine alfabetică (valorile sînt hexazecimale).

În ultima grupă am inclus caracterul „spațiu” și cele două semne de punctuație " : " (două puncte) respectiv " - " (minus). Faptul că, *codul alocat caracterului "spațiu" (20_H)*, este *identic* cu codul aceluiași caracter în setul **ASCII**, este o *pură întâmplare*.

13.1.2. Generatorul de caractere (s gmente) pentru dispozitivul de afișaj : LEDGEN

Așa cum s-a arătat în paragraful 11.2.2., vizualizarea numerelor se va face pe un dispozitiv de afișaj care înglobează 8 elemente de afișaj (**LED-uri** cu 7 segmente). Fiecărei cifre afișabile (**0—9**) îi corespunde un cuvînt binar de 7 bit unic determinat. Aplicînd acest cuvînt la intrările **NSEG0—NSEG6** ale **LED-ului**, el va determina "apriinderea" segmentelor a căror biți de comandă au valoarea "0".

În fig. 12.5. am ilustrat modul de determinare a cuvintelor de comandă pentru cifrele **9** și **5**, aceasta din urmă avînd în exemplul prezentat, punctul zecimal activat. *Reținem că segmentul activ înseamnă bit de comandă "0"*.

Procedînd în mod similar pentru toate cele zece cifre vom obține codurile de comandă ale segmentelor. Grupîndu-le într-o listă, realizăm o tabelă pe care o numim *generator de caractere a dispozitivului de afișaj*. Îi atașăm numele simbolic **LEDGEN**. Această tabelă de **10 octeți** o vom înscrie în **EPROM**, începînd de la adresa **LEDGEN**. Codurile de comandă a segmentelor fiind așezate în ordinea crescătoare a cifrelor, *referirea* oricăruia dintre ele se va putea face simplu, *adăugînd la adresa de bază LEDGEN* însăși *codul intern al cifrei* dorite. Octetul citit din memorie, de la adresa astfel obținută, va fi *codul de comandă segmente al cifrei* respective. Acest octet poate fi depus pe magistrala locală a dispozitivului de afișaj pentru a vizualiza pe un **LED** cifra în cauză.

În fig. 13.1 redăm structura și conținutul generatorului de caractere **LEDGEN**.

	0	1	2	3	4	5	6	7	8	9	
	C0	F9	A4	B0	99	92	82	F8	80	90	} cifra codul de comandă
LEDGEN	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	

Fig. 13.1. Generatorul de caractere pentru dispozitivul de afișaj : LEDGEN

Codurile de comandă din **LEDGEN** sînt exprimate în sistem hexazecimal. Se poate observa că punctul zecimal nu este activat, în niciunul din cele **10 coduri de comandă**. „Aprinderea” punctului zecimal va cădea în sarcina software-ului.

13.1.3. Generatorul de caractere (puncte) pentru imprimantă : PRTGEN

Din descrierea interfeței de imprimantă (paragraf 11.2.3) reținem că rutina de imprimare a unui caracter, **PRTCHAR**, folosește codul intern al caracterului, cu ajutorul căruia localizează în memorie cinci octeți de comandă ace, imprimând succesiv cele cinci coloane ale caracterului. Astfel din mișcarea corelată a capului de imprimare și a acelor, se poate obține — într-o matrice de 5×7 puncte desenul oricărui caracter imprimabil.

În fig. 13.2. exemplificăm modul în care utilizatorul își poate defini forma caracterelor; stabilind în același timp și codurile de comandă ale acelor. Reamintim faptul că un bit de comandă cu valoarea "1", înseamnă punct imprimat. Tot aici specificăm că imprimarea se face de la dreapta la stânga.

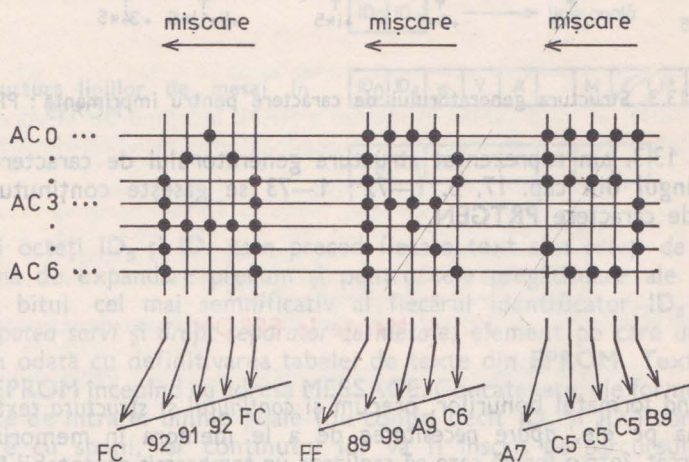


Fig. 13.2. Exemplu pentru constituirea generatorului de caractere a imprimantei : PRTGEN

Grupind octeții de comandă ai acelor (aferele caracterelor) în ordinea crescătoare a codurilor interne, obținem generatorul de caractere **PRTGEN**. Amintim faptul că pe bitul D_7 al portului de ieșire **PRTPORT** se comandă mișcarea motorului de antrenare a capului de imprimare. Acest bit va avea valoarea "1" în toți octeții de comandă **PRTGEN**, deoarece acționarea acelor cu motorul oprit nu are sens.

Pentru a ușura localizarea codurilor de comandă ace, aferente unui caracter, în **PRTGEN** vom rezerva câte cinci octeți și celor trei coduri neimprimabile : **FUNC** [$0B_H$], **TOTAL** [$0E_H$] și **CLEAR** [$0F_H$]. Conținutul acestor zone este lipsit de interes. Astfel obținem o listă care o vom stoca în **EPROM**, începând cu adresa **PRTGEN**. Lista va avea 35 de puncte de intrare, ale căror adrese se obțin ușor, aplicând formula :

$$\text{ENTRY} = \text{PRTGEN} + I * 5 \quad (13.1)$$

Fiecare intrare reprezintă adresa de început a grupei de comandă ace, aferentă unui caracter. În cadrul grupei, octeții sînt depuși în ordine inversă, începînd cu coloana din extrema dreaptă a desenului caracterului, datorită faptului că imprimarea are loc de la dreapta la stînga.

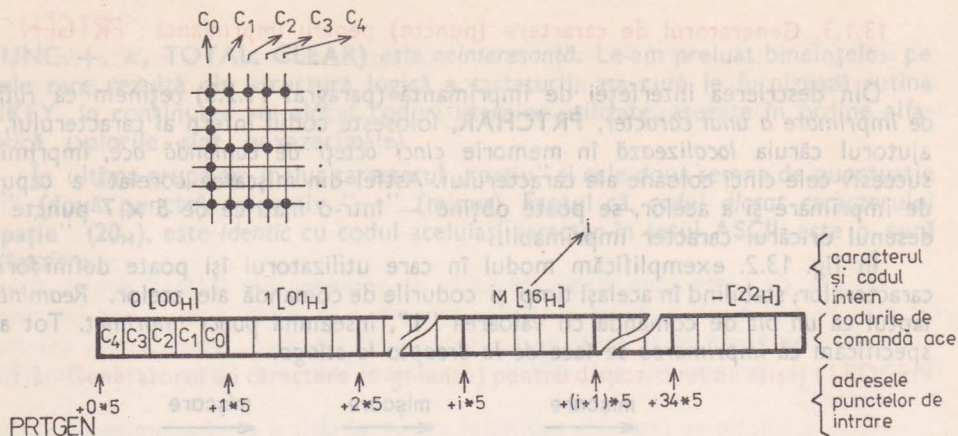


Fig. 13.3. Structura generatorului de caractere pentru imprimantă : PRTGEN

În fig. 13.3. am reprezentat structura generatorului de caractere **PRTGEN**. În listingul din cap. 17, p. 1-72 ; 1-73 se găsește conținutul întregului generator de caractere **PRTGEN**.

13.1.4. Mesaje în EPROM

Analizând formatul bonurilor, precum și conținutul și structura textelor care se vor imprima pe ele, apare necesitatea de a le memora în memoria **EPROM** a casei de marcat, într-o formă care să realizeze un compromis acceptabil între necesar de memorie pentru tabelele și efort de programare. Cele două deziderate sînt contradictorii și nu pot fi minimizate concomitent.

Soluția banală, care ar necesita un efort de implementare minim ar fi aceea de a stoca în **EPROM** mesajele fiecărui bon în parte, structurate conform cerințelor de imprimare impuse prin formatul bonurilor. Soluția aceasta se respinge datorită faptului că ar consuma o cantitate substanțială de memorie **EPROM** :

4 tipuri * 9 linii * 15 caractere = 675 octeți, aproape 16% din totalul de spațiu EPROM.

Ca altă extremă o reprezintă soluția de a memora pur și simplu fiecare mesaj în parte (**TOTAL, CASA NR., UNIT NR., * VA MULȚUMIM ***, etc.) reducînd astfel necesarul de octeți pentru mesaje la 75. În acest caz se poate însă întrezări o încălcare peste măsură a programelor de editare a bonurilor, programe care vor trebui să rezolve generarea formatului de bon impus.

În această situație soluția de compromis devine necesară. Propunem următoarea procedură : în **EPROM** i se va rezerva fiecărui tip de rînd un cîmp de informații. Acest cîmp va fi completat cu informații ajutătoare pentru programele de emisie a bonurilor, precum și cu textul mesajului (doar în rîndurile în care ele vor trebui să existe). Informațiile ajutătoare se vor referi la tipul de bon. Doi octeți ar fi suficienți în acest sens. Caracterele spațiu cuprinse între ultimul caracter de text și sfîrșitul liniei nu se înscriu în **EPROM**.

Se va elabora o rutină **EXPAND** care va transfera la trezire mesajele din **EPROM** în **RAM**, completându-le cu spațiile necesare. Zona în care aceste mesaje se transferă o numim zonă de editare pentru imprimare. Asupra structurii ei vom reveni în acest capitol. Specificăm acum doar atât că, în această zonă, fiecare linie va avea deja structura „oglindă” a imaginii ei de pe bon.

În **EPROM**, liniile vor avea lungimi diferite, în funcție de existența sau inexistența unei secvențe de text pe rândul respectiv. Este evident că dacă lungimea a două texte diferă, atunci și lungimea liniilor respective în **EPROM** va diferi.

În zona de editare pentru imprimare din memoria **RAM** toate rândurile vor avea aceeași lungime, de 17 caractere.

Structura liniilor de mesaj în **EPROM** o redăm în fig. 13.4.

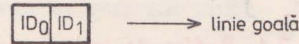
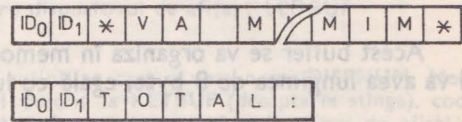


Fig. 13.4. Structura liniilor de mesaj în **EPROM**.



Cei doi octeți ID_0 și ID_1 , care preced fiecare text sînt octeți de identificare pentru rutina de expandare precum și pentru cele pregătitoare ale imprimării. Poziționînd bitul cel mai semnificativ al fiecărui identificator ID_0 la valoarea "1", el va putea servi și drept separator de mesaje, element pe care urma oricum să-l stabilim odată cu definitivarea tabelii de texte din **EPROM**. Textele le vom amplasa în **EPROM** începînd cu adresa **MESSAGE**. Concatenate, ele formează o listă cu 14 puncte de intrare. Liniile goale (nu conțin decît ID_0 și ID_1) vor fi umplute la expandare cu spații, iar conținutul lor va fi înscris pe parcursul operațiilor legate de emiterea unui bon.

Adoptînd această structură de informație lungimea tabelii de texte în **EPROM** va fi de 114 octeți, iar cea a zonei de editare în **RAM** $14 \times 17 = 238$ octeți.

Apreciînd faptul că rutina de expandare precum și cea de pregătire a imprimării nu vor depăși lungimea de 100 octeți considerăm obiectivul propus, cel de a găsi un compromis între necesar de memorie și efort de programare, ca fiind atins.

În listingul din Cap. 17, pag. 1—70, 1—71, se găsește conținutul exact al tabelii de texte din **EPROM** (**MESSAGE**). În dreptul fiecărei linii conținutul ei se indică în clar-cu majuscule, sub forma unui comentariu. În dreptul liniilor vide se menționează (în paranteze, cu litere mici) destinația lor, adică informația care se va înscrie în ele pe parcursul operațiilor legate de emiterea unui bon.

13.2. Zone de manevră

Structurarea programelor este adesea înlesnită prin folosirea unor zone tampon de memorie, pentru vehicularea datelor. Aceste zone de manevră poartă numele de *buffer*. Constituirea lor permite o delimitare mai clară a activităților diverselor

module de software. Astfel se poate urmări mai ușor fluxul informațional realizat în orice echipament. Utilizarea bufferelor este caracteristică atât rutinelor de intrare/ieșire cât și modulelor software ierarhic superioare.

Pornind de la aceste premise ne propunem ca, codul fiecărei taste citite să fie depus într-un buffer (KEYBUF). Așa vom proceda, și cu caracterele care urmează să fie vizualizate (se depun în LEDBUF) precum și cu cele de imprimat, (PRT-BUF). Aceste manevre vor fi efectuate de către modulele software imediat superioare rutinelor de tratare fizică a interfețelor, prezentate în Cap. 12.

În afara acestor buffere constituite pentru deservirea interfețelor vom defini și zone de manevră, care vor servi comunicația cu module software ierarhic superioare.

13.2.1. Buffer de intrare date de la tastatură : KEYBUF

Acest buffer se va organiza în memoria RAM, începînd de la adresa KEYBUF, și va avea lungimea de 8 byte, egală cu lungimea dispozitivului de afișaj.

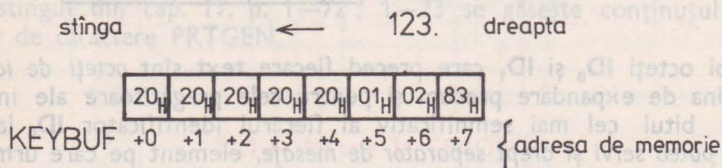


Fig. 13.5. Buffer de intrare date de la tastatură : KEYBUF

Trăsături :

- KEYBUF conține codurile interne ale ultimelor 8 cifre tastate ;
- punctul zecimal "." se reprezintă setînd (valoarea "1") bitul cel mai semnificativ al ultimei cifre tastate (KEYBUF+7) ;
- poziția KEYBUF+7 este cea mai puțin semnificativă și apare pe dispozitivul de afișaj în extrema dreaptă ;
- încărcarea KEYBUF se va face cu o rutină pe care o vom numi STORENUM. Ea va deplasa conținutul întregului buffer cu o poziție la stînga, noul caracter (cod intern) fiind introdus pe poziția KEYBUF+7 ;
- cu ocazia deplasării caracterul cel mai semnificativ (de pe poziția KEYBUF+0) se va pierde.

13.2.3 Buffer de ieșire pentru dispozitivul de afișaj : LEDBUF

LEDBUF se va organiza în memoria RAM, începînd de la adresa LEDBUF, și va avea lungimea de 8 byte, egală cu lungimea dispozitivului de afișaj.

Trăsături :

- LEDBUF conține codurile binare de comandă segmente, pentru cele 8 LED-uri cu 7 segmente și punct zecimal ;
- segmentul aprins apare ca bit de valoare "0", iar cel stins cu valoarea "1" ;
- punctul zecimal se reprezintă înscriind valoarea zero în bitul cel mai semnificativ (D₇) al octetului corespunzător LED-ului dorit ;

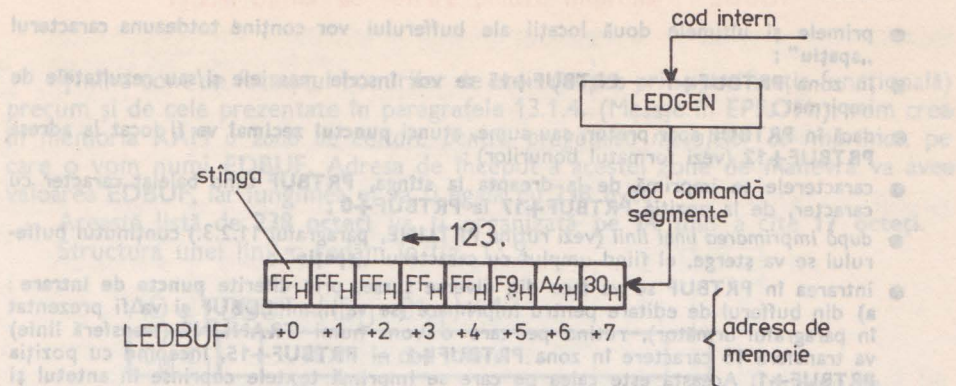


Fig. 13.6. Buffer de ieșire pentru dispozitivul de afișaj : LEDBUF

- **Încărcarea LEDBUF** se va face cu o rutină pe care o vom numi **DISPNUM**. Mecanismul de încărcare seamănă cu cel descris la **KEYBUF** (dreapta la stînga), codul introdus pe poziția **LEDBUF+7** obținându-se transcodind codul cifrei de afișat cu ajutorul generatorului de caractere (segmente) a dispozitivului de afișaj **LEDGEN** ;
- poziția **LEDBUF+7** este cea mai puțin semnificativă și apare pe dispozitivul de afișaj în poziția extremă dreaptă ;
- **transferul conținutului LEDBUF** la dispozitivul de afișaj va fi efectuat de rutina de servire a întreruperilor mascabile **INT**, care va fi activată din 2 în 2 ms ;
- deosebit un caz special, în care încărcarea **LEDBUF** nu se va face conform procedurii enunțate mai sus.

În cazul folosirii oricărei proceduri, declanșată prin apăsarea succesivă a tastei **FUNC** (1-7 tastări) poziția **LEDBUF+0** (extrema stînga a dispozitivului de afișaj) va fi folosită ca numărător de tastări succesive, **FUNC** (vezi specificația funcțională **F2.0**).

În acest caz **înscrierea codului de afișat se va face direct pe poziția LEDBUF+0**, fără a deplasa conținutul bufferului. Această operație va fi efectuată de o rutină pe care ne propunem să o numim **CNTFUNC**.

13.2.3. Buffer de ieșire pentru imprimantă : PRTBUF

PRTBUF va fi organizat în memoria RAM, începînd cu adresa **PRTBUF**, și va avea lungimea de **18 byte**. În acest buffer se va genera imaginea (reprezentată în coduri interne) a unui rînd care se va imprima cu ajutorul rutinei deja prezentate **PRT18C**.

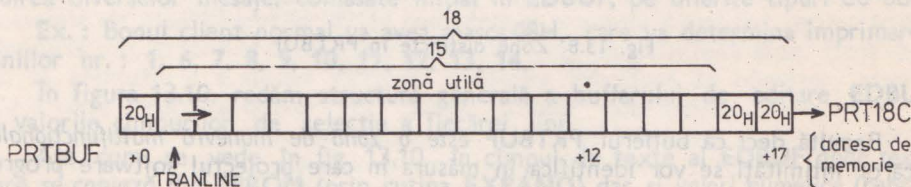


Fig. 13.7. Structura generală a bufferului de ieșire la imprimantă : PRTBUF

Trăsături :

- primele și ultimele două locații ale bufferului vor conține totdeauna caracterul „spațiu”;
- în zona $PRTBUF+1 - PRTBUF+15$ se vor înscrie mesajele și/sau rezultatele de imprimat;
- dacă în $PRTBUF$ apar prețuri sau sume, atunci punctul zecimal va fi locat la adresa $PRTBUF+12$ (vezi formatul bonurilor);
- caracterele se imprimă de la dreapta la stînga, $PRTBUF$ fiind baleiat caracter cu caracter, de la poziția $PRTBUF+17$ la $PRTBUF+0$;
- după imprimarea unei linii (vezi rutina $PRTLINE$, paragraful 11.2.3.) conținutul bufferului se va șterge, el fiind umplut cu caracterul „spațiu”;
- intrarea în $PRTBUF$ se va face din diverse surse, prin diferite puncte de intrare :
 - a) din bufferul de editare pentru imprimare (se va numi $EDBUF$ și va fi prezentat în paragraful următor), rutina pe care o vom numi $TRANLINE$ (transferă linie) va transfera 15 caractere în zona $PRTBUF+1 - PRTBUF+15$, începînd cu poziția $PRTBUF+1$. Aceasta este calea pe care se imprimă textele cuprinse în antetul și la sfîrșitul bonurilor.
 - b) pe parcursul operației se vor imprima însă și linii individuale cuprinzînd coduri de sortiment și prețuri, precum și un marcaj locat în prima coloană din dreapta, privind tipul operației efectuate (vezi formatul bonurilor și specificația funcțională F.1.8.). Încărcarea bufferului pentru efectuarea acestor operații se va face după tastarea tastei „+” printr-o procedură pe care ne propunem să o denumim $OPFORBUY$ (operații pentru client). În această conjunctură $PRTBUF$ (sau zone ale lui) va fi încărcat direct prin diverse rutine a căror structură exactă nu se poate întrezări în acest moment :
- putem identifica cîteva zone distincte :
 - $PRTBUF+1 - PRTBUF+2$ – va conține codul de sortiment
 - $PRTBUF+4 - PRTBUF+14$ – va conține prețuri (sau totaluri)
 - $PRTBUF+12$ – va conține punctul zecimal
 - $PRTBUF+15$ – va conține un semn (+, -, A sau *) specific operației efectuate ;
- știînd că după apăsarea tastei „+”, dispozitivul de afișaj va trebui să conțină suma curentă a clientului, iar pe imprimantă se va imprima ultimul preț (sau valoare) introdus, precum și în ideea de a reduce numărul zonelor de manevră RAM, ne propunem ca $PRTBUF$ (zona $PRTBUF+4 - PRTBUF+14$) să fie folosit ca zonă tampon pentru pregătirea afișării. Rutina pe care o vom numi $DISPNUM$ (a nu se confunda cu $DISPNUM$), va transfera numărul din $PRTBUF$ în $LEDBUF$, comprimîndu-l totodată pentru a elimina octetul ce conține punctul zecimal.

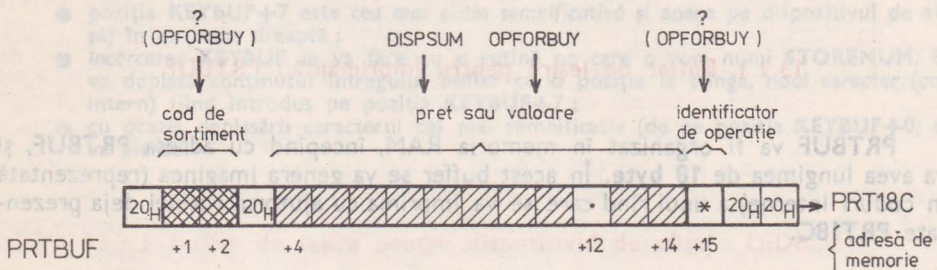


Fig. 13.8. Zone distincte în $PRTBUF$

Rezultă deci că bufferul $PRTBUF$ este o zonă de manevră multifuncțională, a căror întimități se vor identifica în măsura în care proiectul software progresa.

13.2.4. Buffer de editare pentru imprimare : EDBUF

Ținând cont de formatul bonurilor de emis (impus prin specificația funcțională) precum și de cele prezentate în paragrafele 13.1.4. (Mesaje în EPROM.), vom crea în memoria RAM o zonă de editare pentru pregătirea mesajelor de imprimat, pe care o vom numi EDBUF. Adresa de început a acestei zone de manevră va avea valoarea EDBUF, iar lungimea ei va fi egală cu 238 byte.

Această listă de 238 octeți va fi organizată pe 14 linii a câte 17 octeți. Structura unei linii o redăm în fig. 13.9.

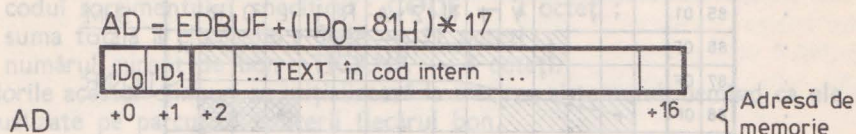


Fig. 13.9. Structura unei linii în EDBUF

În fiecare linie se disting două câmpuri :

- a) câmpul de identificare — constă din 2 octeți de identificare ID₀ și ID₁
- b) câmpul de text — constă din 15 octeți și conține codurile interne ale caracterelor ce vor fi imprimate pe un rând

ID₀ — îl alegem, poate redundant, pentru a-i atribui 2 funcții :

- având bitul cel mai semnificativ înscris (D₇=1) el va putea servi drept separator în tabela de date din EPROM (MESSAGE), trăsătură utilizată de rutina EXPAND care va genera la trezirea casei de marcat conținutul inițial al EDBUF

- pe primii patru biți ai acestui octet (D₀—D₃) vom înscrie un număr de ordine, care poate, ne va fi util.

ID₁ — este conceput ca un câmp de selecții. Se utilizează primii patru biți ai săi (D₀—D₃). Știind că pe diversele tipuri de bonuri care trebuiesc emise, apar linii comune, ne propunem ca rutina pe care o vom concepe, PRTTICK, să primească din rutina apelantă o mască (1 octet) de selecție.

Masca de selecție va avea un singur bit cu valoarea 1, situat pe una din pozițiile D₀ — D₃, restul biților fiind zero. Cele 4 măști preconizate sînt :

- mască pentru bon client : BUYMASK — 08H
- mască pentru bon de sortiment : SORTMASK — 04H
- mască pentru bon totalizator : DAYMASK — 02H
- mască pentru bon de sinteză : SYNTMASK — 01H

PRTTICK compară masca de selecție primită cu câmpul ID₁ al fiecărei linii din EDBUF și va imprima linia respectivă dacă bitul corespunzător din ID₁ este egal cu cel solicitat prin mască. Astfel se poate concepe destul de elegant distribuirea diverselor mesaje, comasate inițial în EDBUF, pe diferite tipuri de bon.

Ex. : Bonul client normal va avea masca 08H, care va determina imprimarea liniilor nr. : 1, 6, 7, 8, 9, 10, 11, 12, 13, 14,

În figura 13.10. redăm structura generală a bufferului de editare EDBUF și valorile câmpurilor de selecție a fiecărei linii.

Așa cum se vede în fig. 13.10., în câmpul de texte al EDBUF apar texte care se copiază din EPROM (prin rutina EXPAND) dar și valori numerice (reprezentate în cod intern) care se generează în timpul funcționării casei de marcat.

Aceste din urmă elemente le grupăm în *constante* și *variabile*. **Constantele** și **cîmpurile** lor aferente sînt :

- numărul de identificare al casierului : **CLRKN** — 3 octeți ;
- numărul de identificare al casei : **DESKN** — 2 octeți ;
- numărul de identificare al magazinului : **SHOPN** — 3 octeți ;
- data curentă : **DATE** — 8 octeți.

Aceste cîmpuri se înscriu la programarea sesiunii de lucru conform specificației funcționale F.2.1. Ele rămîn nemodificate pînă la nouă reprogramare.

Variabilele și cîmpurile lor aferente cuprinse în **EDBUF** vor fi :

- codul sortimentului specificat : **CODE** — 2 octeți ;
- suma totală a clientului : **SUM** — 11 octeți ;
- numărul curent de bon : **TICKNR** — 4 octeți.

Valorile acestor cîmpuri se inițializează la trezirea sistemului, urmînd ca ele să fie actualizate pe parcursul emiterii fiecărui bon.

Adresele de început ale acestor elemente **prezintă puncte de intrare distincte în EDBUF**. *Lor li se vor atribui nume simbolice*, identice cu numele variabilelor și se vor referi relativ față de adresa de început a zonei (**EDBUF**), conferind astfel programului un plus de flexibilitate : ele *nu vor trebui modificate dacă într-un nou proiect memoria RAM (sau EDBUF) se va dispune într-o altă zonă*. În acest caz se va indica doar noua adresă de început a bufferului.

CODE : = $EDBUF + 2 * 17 - 4$

SUM : = $EDBUF + 5 * 17 + 5$

TICKNR : = $EDBUF + 7 * 17 + 3$

CLRKN : = $TICKNR + 10$

SHOPN : = $EDBUF + 11 * 17 - 4$

DESKN : = $EDBUF + 12 * 17 - 4$

DATE : = $EDBUF + 12 * 17 + 5$

Aceste operații se regăsesc în listingul din Cap. 17., pag. 1—1.

13.2.5. Tabela de distribuire a parametrilor de stare : **SETUPTAB**

Această zonă nu reprezintă de fapt o zonă de manevră, ci este, mai exact spus, o *tabelă de distribuire* a unor manevre. În procesul de programare a casei de marcat (specificația funcțională F.2.1.), este prevăzută introducerea de la tastatură a parametrilor de stare a casei de marcat : numărul casierului, casei, a magazinului și data curentă. În vederea implementării acestei trăsături vom elabora o rutină pe care o vom numi **SETUP**. Știm că *structura datelor* ce urmează a fi citite prin această procedură este eterogenă : unele sînt numere „pure” (numerele de identificare) iar altele comportă și prezența punctului zecimal (data calendaristică), el servind ca separator între zi și lună, respectiv lună și an. Dacă dorim să efectuăm și o analiză sintactică a datelor introduse de operator, atunci cele a căror structură diferă, vor trebui tratate diferențiat.

Dacă intrarea va trebui tratată diferențiat, atunci *ne propunem să tratăm măcar ieșirea în mod unitar*. De aceea vom constitui în **EPROM** o tabelă de distribuție **SETUPTAB** care conține adresele de distribuție a parametrilor de stare, și lungimea cîmpului care va fi transferat (lungimea fiecărui parametru, exprimată în octeți). Vom elabora și o rutină care va fi apelată din **SETUP** și va transfera

datele, deja validate din KEYBUF în zonele lor dedicate din EDBUF. Să numim această rutină TRANSPAR (transfer de parametri). Avînd informația structurată conform SETUPTAB elaborarea rutinei TRANSPAR nu va crea probleme.

Folosind pseudoinstrucțiunile asamblului M80, vom defini SETUPTAB după cum urmează :

```

SETUPTAB : DW  SHOPN      ; adresa număr unitate
           DB  SHOPNL     ; lungime număr unitate
           DW  DESKN      ; adresa număr casă
           DB  DESKNL     ; lungime număr casă
           DW  DATE       ; adresa data curentă
           DB  DATEL      ; lungimea cîmpului rezervat
           DW  CLRKN      ; adresa număr casier
           DB  CLRKNL     ; lungime număr casier
    
```

Ordinea disponerii celor 4 cîmpuri din SETUPTAB a fost impusă de succesiunea specificată în F.2.1.

În fig. 13.11. redăm structura SETUPTAB.

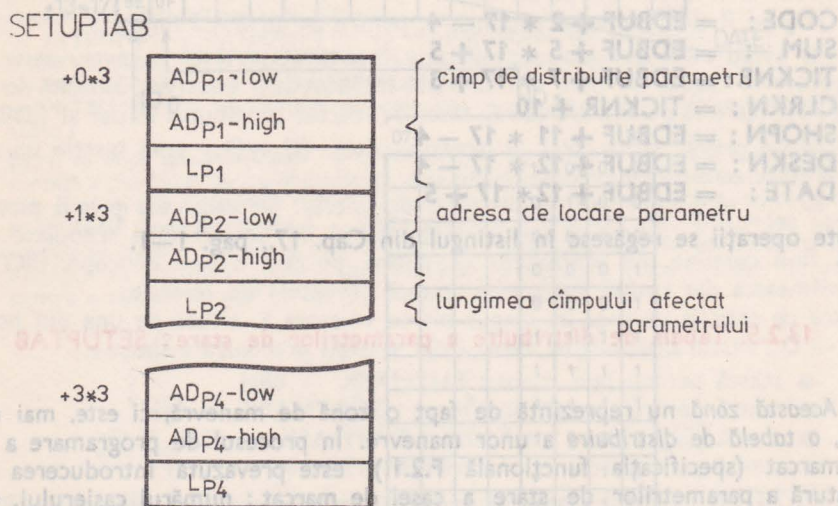


Fig. 3.11. Structura tabelii de distribuire a parametrilor de stare: SETUPTAB

13.3. Regiștri RAM

Știm deja că sumele pe care le vom vehicula cu programele casei de marcat nu pot fi reprezentate nicidecum pe unul sau doi octeți. Forma de reprezentare internă a numerelor (prețuri) asupra cărora va trebui să efectuăm operații aritmetice o vom alege cea BCD, cu punct zecimal fix. Suma cea mai mare care ur-

mează să fie vehiculată de această casă, are lungimea de 10 cifre semnificative, din care 2 sînt rezervate pentru partea zecimală.

Vom încerca să definim rutinele noastre de aritmetică BCD cu virgulă fixă, într-o manieră asemănătoare cu cea a microprocesorului. Microprocesorul Z80 execută toate operațiile aritmetice între 2 numere (binare cu 8-bit) astfel încît unul din parametri va fi locat într-un registru dedicat (A), iar celălalt într-un alt registru. Rezultatul este generat în același registru A, suprascriindu-se peste vechea valoare din A.

Ținînd cont de faptul că numerele vehiculate de casa de marcat sînt reprezentate pe 5 octeți, este clar că ele nu vor putea fi locat în regiștri interni ai microprocesorului. De aceea vom rezerva zone dedicate în memoria RAM, pe care programele noastre le vor înzestra cu funcții asemănătoare regiștrilor interni ai microprocesorului. De aceea aceste zone le vom numi regiștri RAM.

13.3.1. Regiștrul datelor în format BCD extins : EBCD

Numerele introduse de la tastatura casei de marcat se regăsesc în KEYBUF. Reprezentarea lor poartă următoarele caracteristici :

- fiecare cifră este înscrisă într-un octet ;
- punctul zecimal apare pe bitul D_7 al cifrei după care a fost introdus
- numărul poate fi nenormalizat, adică să conțină mai mult decît 2 cifre zecimale, sau nici una ;
- zerourile ne semnificative sînt substituie cu blanchuri (caracter spațiu).

Pentru programele de aritmetică ale casei de marcat, aceste numere trebuie transpuse într-un format unitar caracterizat prin :

- lungimea numărului este unică : 5 octeți ;
- în fiecare octet apar 2 cifre BCD ;
- punctul zecimal nu apare explicit, ci el va fi considerat ca existent între octetul cel mai puțin semnificativ și următorul ;
- zerourile ne semnificative trebuie să fie într-adevăr "0".

Transpunerea o vom efectua în două etape :

1. Numărul original se prelucrează și se aduce într-un format concretizat prin :
 - fiecare cifră ocupă un octet ;
 - cifrele zecimale excedentare sînt eliminate ;
 - punctul zecimal nu apare explicit ;
 - zerourile ne semnificative sînt într-adevăr "0".

Acest format îl numim format BCD extins.

2. Numerele reprezentate în format BCD extins sînt convertite în numere BCD, reprezentate conform specificației de mai sus.

Pentru crearea numerelor în format BCD extins vom rezerva o zonă tampon în memoria RAM, pe care o vom numi registru EBCD.

În fig. 13.12. redăm structura regiștrului EBCD, care este plasat în memoria RAM, începînd cu adresa EBCD și are lungimea de 10 octeți.

Distingem două cîmpuri :

- EBCD+0 — EBCD+7 — conține partea întregă a numărului ;
 - EBCD+8 — EBCD+9 — conține partea zecimală a numărului.
- (EBCD+0 este cifra cea mai semnificativă a numărului).

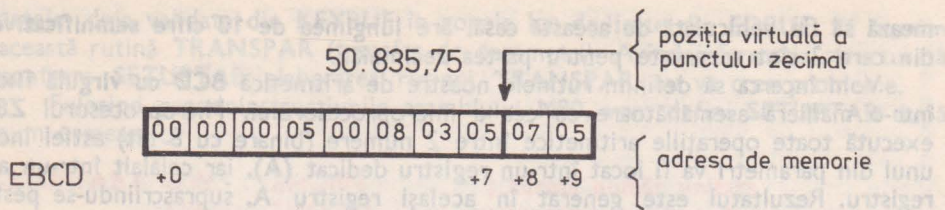


Fig. 13.12. Registrul datelor în format BCD extins : EBCD

Structura unui octet din EBCD este :

- $D_7 - D_4$ - nesemnificativ (0)
- $D_3 - D_0$ - cifra zecimală în cod BCD

13.3.2. Regiștrii de aritmetică BCD

Definim : Toate operațiile aritmetice vor avea forma :

$$TMPBCD := TMPBCD \text{ OP } DATBCD$$

Cei doi regiștri BCD vor avea o structură identică, caracterizată prin :

- lungimea regiștrului este de 5 octeți ;
- partea zecimală apare în octetul din extrema dreaptă, pe octetul cu cea mai mare adresă de memorie ;
- partea întregă apare în primii 4 octeți ;
- cifra cea mai semnificativă apare pe biții $D_7 - D_4$ și celulei $xBCD+0$. (unde x poate fi TMP sau DAT)

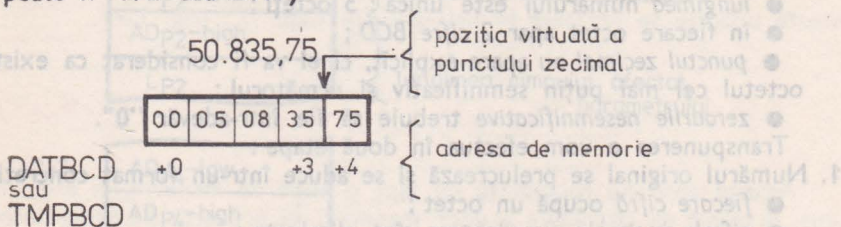


Fig. 13.13. Structura regiștrilor aritmeticii BCD

Preconizăm ca în $DATBCD$ să fie depuse numerele introduse de la tastatură. În $TMPBCD$ se va depune celălalt operand al unei operații aritmetice. El servește și drept registru rezultat, avînd o funcție numită în general : *acumulator*.

13.3.3. Regiștri BCD de uz general, în memorie

Așa cum rezultă din specificația funcțională a casei de marcat, aceasta va trebui să fie capabilă să și memoreze anumite valori : *suma curentă a clientului*, *totalul vânzărilor* (suma curentă din casă), *sumele vânzărilor pe cele 100 clase de sortimente*.

Aceste valori se vor stoca în formă cea mai comprimată : BCD. Suma totală a clientului se va crea într-un registru RAM pe care-l numim GUESTBCD.

Suma totală a vânzărilor din sesiunea respectivă de lucru o vom crea în alt registru RAM. Fie numele simbolic al acestui registru : DAYBCD. Structura acestor registre este identică cu cea a regiștrilor DATBCD respectiv TMPBCD (vezi fig. 13.13.).

Pentru memorarea vânzărilor, defalcată pe dormimente, va trebui să constituim, o tabelă în RAM, care va avea lungimea egală cu 100 de regiștri BCD. Cei 500 de octeți, astfel, rezervați vor consuma aproape jumătate din memoria RAM disponibilă fizic.

Tabela defalcată a vânzărilor o vom loca la adresa SORTTOT. Ea va avea 100 intrări corespunzătoare celor 100 de coduri de sortiment.

SORTTOT

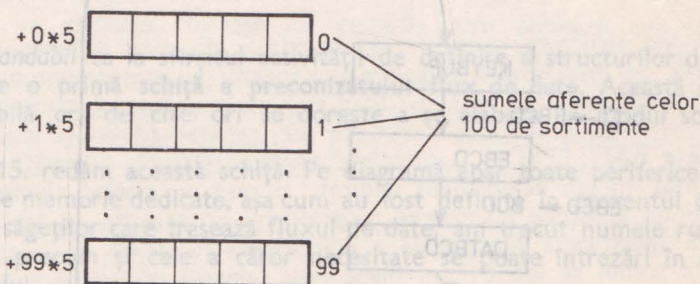


Fig. 13.14. Structura tabelii defalcate a vânzărilor.

Pentru a regăsi totalul aferent unui cod de sortiment dat vom aplica relația :

$$ADSORT_i := SORTTOT + 5 * i$$

unde ADSORT_i este adresa de început a cîmpului afectat sortimentului cu codul de sortiment i.

13.4. Parametri și variabile

Acest paragraf ar trebui să poarte titlul „Diverse”. În faza actuală a proiectului nu întrezărim decît 2 elemente :

1. Celula martor pentru portul de ieșire SYSOUT, despre rolul căreia s-a vorbit în paragraful 11.2.2.

2. Indicatorul tipului de trezire, cel pe baza căruia se va decide dacă secvența de trezire pe care o execută microprocesorul este o trezire caldă, sau una rece. Trebuie să distingem cazul în care microprocesorul execută o primă secvență de trezire, la începutul sesiunii de lucru, de situația în care repornirea se face la reparația tensiunii de alimentare. În primul caz se vor executa o serie de inițiali-

zări iar în cel de-al doilea activitatea trebuie continuată din punctul în care ea a fost abandonată la dispariția tensiunii de alimentare.

Pentru a putea să luăm o decizie în acest sens, vom compara o secvență din tabela de texte MESSAGE din EPROM, cu prima linie din EDBUF, aflat în memoria RAM. Știm că la trezirea sistemului (pornire rece) se va activa rutina EXPAND care transpune MESSAGE în EDBUF. Dacă cele două secvențe sînt

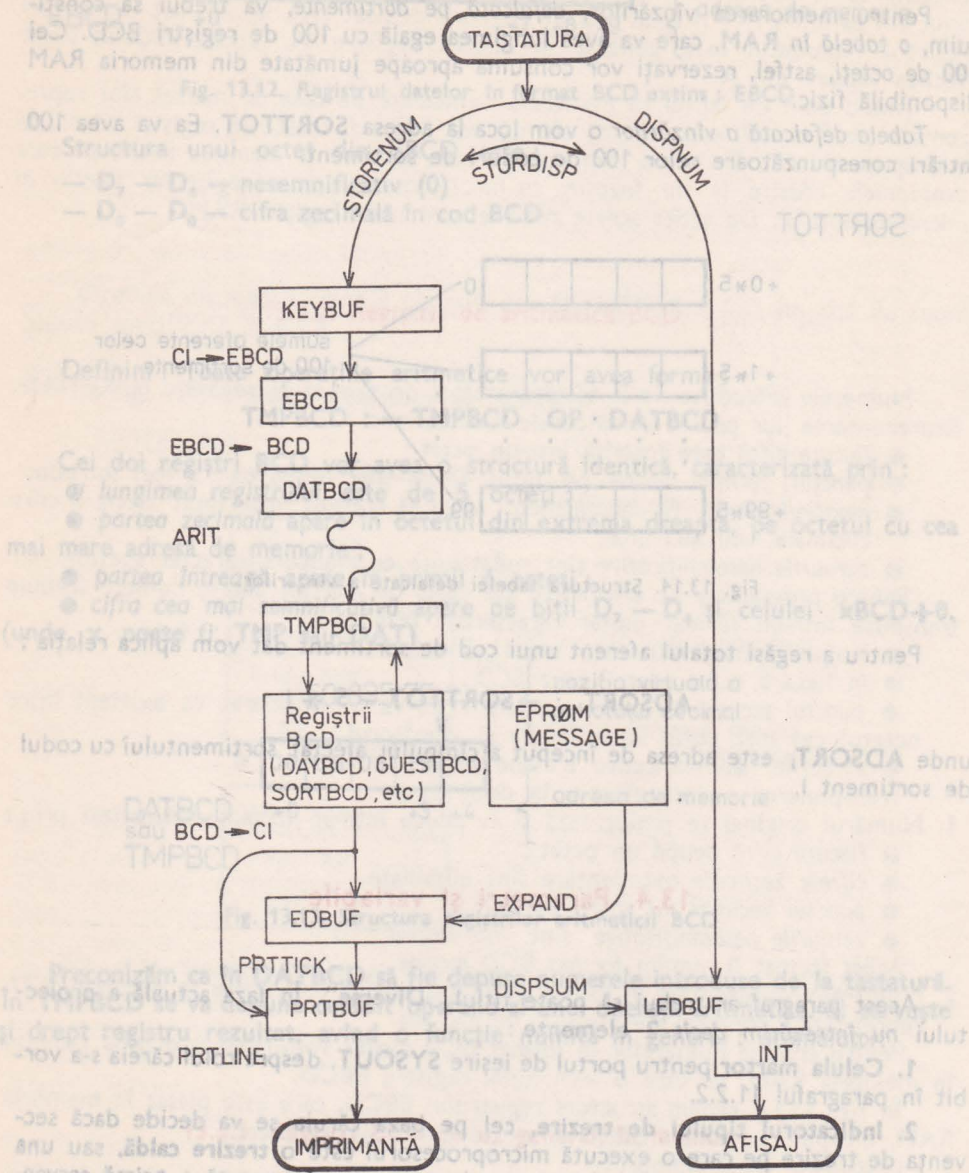


Fig. 13.15. Fluxul de date în casa de marcat (o primă aproximație).

identice, atunci trezirea curentă va fi cea „caldă”. Acest indicator poate fi bun, deoarece este puțin probabil ca memoria RAM proaspăt alimentată cu tensiune, să se trezească, la adresele respective cu un conținut identic cu cel din EPROM.

Primului element *i* se va rezerva spațiu în memorie : **WITNESS** — se rezervă un octet al cărui conținut va fi identic cu conținutul portului de ieșire **SYSOUT**.

Pentru indicator de trezire nu se va mai rezerva spațiu RAM, deoarece el este inclus în **EDBUF**.

În fine, menționăm că este improbabil să fi intuit toate variabilele și datele care pot apărea pe parcursul elaborării proiectului. Nutrind speranța că n-am omis nici una din cele importante, încheiem prezentarea structurilor de date.

Starea de repaus interconectează testarea unei taste, și ea va fi abandonată în momentul în care apare o testă semnificativă :

13.5. Fluxul de date în casa de marcat (o primă aproximație)

TOTAL pentru emiterea unui bon vid

PUNC, care va de-lansa funcție de

Este recomandabil ca la sfârșitul activității de definire a structurilor de date, să se elaboreze o primă **schită** a **preconizatului flux de date**. Această constatare este valabilă ori de câte ori se dorește a se elabora un modul software complex.

În fig. 13.15. redăm această schiță. Pe diagramă apar toate perifericele precum și zonele de memorie dedicate, așa cum au fost definite în prezentul capitol.

În dreptul săgeților care trasează fluxul de date, am trecut numele rutinelor deja elaborate, precum și cele a căror necesitate se poate întrezări în această fază a proiectului.

Această schiță va fi urmată la sfârșitul elaborării proiectului software, în mod obligatoriu de o **schită finală** care va însoți documentația programului.

mandările făcute în Cap. 10... privind succesiunea de elaborare și de implemen-
tare a programelor... vom începe cu modulele terarhic-superioare... coborând treptat
câte cele mai simple. Vom lucra permanent cu atenția distribuită pentru a
putea cuprinde problemele în totalitatea ei. De aceea un ne vom îndrăzni în a
elabora o ramură oricare a programului până la ultimul detaliu, înainte de a
fi scris modulele terarhic superioare ale tuturor ramurilor.
Succesiunea de apariție a paragrafelor din acest capitol este însăși o ierar-
hizare a rutinelor. Credem că acest lucru rezultă și din numele paragrafelor :

- 14.1. Bucii principale
- 14.2. Programe de prelucrare
- 14.3. Algoritmi BCD cu virgulă fixă
- 14.4. Analiza sintactică și conversii de coduri
- 14.5. Veculcare de date

în ceea ce privește metoda generală de lucru predărm la început vom
descrie algoritmul rutinei studiate, apoi prin organizarea, și în limbajul de
nivel înalt propus. Cele două aproximații grosiere vor fi urmate de prezentarea
programului în limbaj de asamblare. După ce familiarizarea cu limbajul descriptiv
de nivel înalt se va fi făcut, vom renunța treptat la organizarea în măsura în
care lucrarea avansează, coborând spre nivele terarhic inferioare, obținând
plificarea problemelor vom renunța și la descrierea în limbaj de nivel înalt, obținând
doar de la caz la caz, o secvență semnificativă de program, scris în limbaj de
asamblare, și sau schițe menite să exemplifice tehnici de implementare adoptate.

FUNC este conținutul tuturor succesiunii testelor

14

IMPLEMENTAREA PROGRAMULUI

Avînd structura hardware și structurile de date definite, se poate demara acțiunea de elaborare a software-ului specific al echipamentului luat în studiu. Pachetul de programe pe care-l vom elabora în prezentul capitol, este acela care va transforma microcalculatorul de uz general (dotat cu microprocesor Z80, memorie RAM, EPROM și interfețe pentru tastatură, dispozitiv de afișaj și imprimantă) într-un echipament dedicat : o casă de marcat electronică. De aceea, înainte de a începe elaborarea programului propriu-zis, va trebui să recitim cu atenție **specificația funcțională** a echipamentului (Cap. 11.).

Va trebui să identificăm bucla (sau ramura) principală a activităților casei. Doar în acel moment se va declanșa elaborarea software-ului : de la esență spre detalii.

Pe parcursul întregului capitol vom încerca să respectăm cît mai exact recomandările făcute în Cap. 10., privind succesiunea de elaborare și de implementare a programelor : vom începe cu modulele ierarhic superioare, coborînd treptat către cele mai simple. Vom lucra permanent cu atenția distribuită pentru a putea cuprinde problema în totalitatea ei. De aceea nu ne vom hazarda în a elabora o ramură oarecare a programului pînă la ultimul detaliu, înainte de a fi scris modulele ierarhic superioare ale tuturor ramurilor.

Succesiunea de apariție a paragrafelor din acest capitol este însăși o ierarhizare a rutinelor. Credem că acest lucru rezultă și din numele paragrafelor :

14.1. Bucla principală

14.2. Programe de prelucrare

14.3. Aritmetică BCD cu virgulă fixă

14.4. Analiza sintactică și conversii de coduri

14.5. Vehiculare de date

În ceea ce privește metoda generală de lucru precizăm : la început vom descrie algoritmul rutinei studiate, atît prin organigramă, cît și în limbajul de nivel înalt propus. Cele două aproximații grosiere vor fi urmate de prezentarea programului în limbaj de asamblare. După ce familiarizarea cu limbajul descriptiv de nivel înalt se va fi făcut, vom renunța treptat la organigrame. În măsura în care lucrarea avansează, coborînd spre nivele ierarhice inferioare, odată cu simplificarea problemelor vom renunța și la descrierile în limbaj de nivel înalt, păstrînd doar de la caz la caz, cîte o secvență semnificativă de program, scris în limbaj de asamblare, și sau schițe menite să exemplifice tehnica de implementare adoptată.

Din analiza specificației funcționale, bucla principală a programului se cristalizează în jurul „repausului interbon”. Acesta este punctul la care se ajunge după execuția secvențelor de trezire (F.1.0.). Din acest punct se lansează activitățile și tot în acest punct se va reveni după executarea oricăreia din funcțiile specifice ale casei de marcat (F.1.1.).

Starea de repaus interbon așteaptă tastarea unei taste, și ea va fi abandonată în momentul în care apare o tastă semnificativă :

- cifră sau punct pentru introducerea unui preț ;
- TOTAL pentru emiterea unui bon vid ;
- FUNC, care va declanșa, funcție de tastările care urmează, una din cele 7 acțiuni programabile : introducerea unui preț cu cod de sortiment (F.1.13.), introducerea unui preț de ambalaj (F.1.15.), programarea sesiunii de lucru (F.2.1.), emiterea unui bon de anulare (F.2.2.), generarea totalului de sortiment (F.2.3.), generarea totalului de vânzări (F.2.4.), și generarea sintezelor vânzărilor (F.2.5.): Celelalte taste (CLEAR, +, *) sînt fără semnificație, dacă ele sînt tastate în repausul interbon.

Pe baza acestor considerente constatăm că vor fi două activități majore care se vor putea lansa din repausul interbon :

- introducerea și prelucrarea unui preț (bon)
- prelucrarea (procesarea) tastei FUNC

Astfel se poate concepe organigrama buclei principale : MAINLOOP (vezi fig. 14.1).

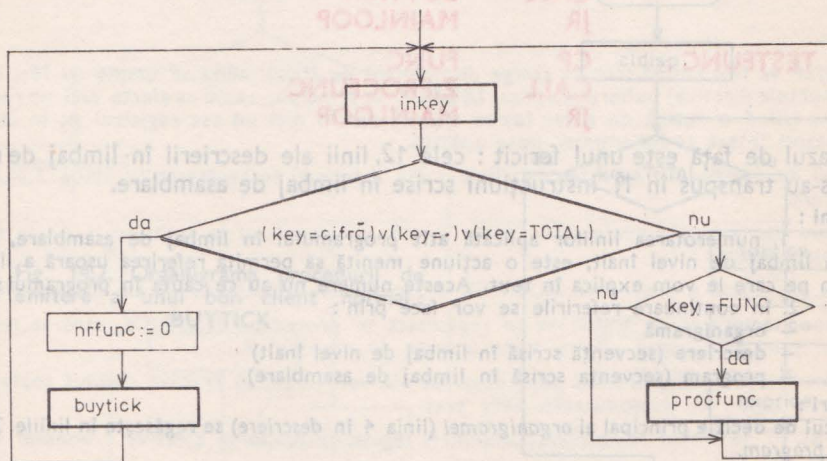


Fig. 14.1. Organigrama buclei principale : MAINLOOP

Sintetizînd : cifrele, punctul zecimal și TOTAL declanșează procedura legată de emiterea unui bon (BUYTICK), iar tasta FUNC va determina activarea unei proceduri de prelucrare (PROCFUNC).

nrfunc — este contorul tastărilor succesive a tastei FUNC

Descrierea aceleiași organigrame în limbaj de nivel înalt este :

```

1 repeat
2   begin
3     inkey
4     if (key=cifră)V(key=punct)V(key=TOTAL)
5       then
6         begin
7           nrfunc:=0
8           buytick
9         end
10      else
11        if key = FUNC then procfunc
12      end

```

Transpunând acest program în limbaj de asamblare obținem :

```

1  MAINLOOP:  CALL  INKEY
2              CP   NRRPRT
3              JR   C,SIMPBUY
4              CP   TOTAL
5              JR   NZ,TESTFUNC
6  SIMPBUY:   LD    C,0
7              CALL BUYTICK
8              JR   MAINLOOP
9  TESTFUNC:  CP   FUNC
10             CALL Z,PROCFUNC
11             JR   MAINLOOP

```

Cazul de față este unul fericit : cele 12, linii ale descrierii în limbaj de nivel înalt, s-au transpus în 11 instrucțiuni scrise în limbaj de asamblare.

Mențiuni :

1. numerotarea liniilor aplicată atât programului în limbaj de asamblare, cât și celui în limbaj de nivel înalt, este o acțiune menită să permită referirea ușoară a liniilor program pe care le vom explica în text. Aceste numere nu au ce căuta în programul sursă.
2. în continuare referirile se vor face prin :
 - organigramă
 - descriere (secvență scrisă în limbaj de nivel înalt)
 - program (secvență scrisă în limbaj de asamblare).

Adnotări :

- Blocul de decizie principal al organigramei (linia 4 în descriere) se regăsește în liniile 2 – 5 din program.
- Ramura din stînga a organigramei, (liniile 5 – 8, 12 din descriere) se regăsesc în liniile 6 – 8 ale programului.
- Ramura din dreapta organigramei (liniile 10 – 12 din descriere) se regăsesc în liniile 9 – 11 ale programului.
- Pentru delimitarea cifrelor și a punctului de restul tastelor, am folosit constanta NRRPRT = 0BH (linia 2 din program), plecînd de la cunoștința codurilor interne ale caracterelor : cifrele și punctul zecimal au coduri mai mici (00–0AH) decît NRRPRT. Dacă în urma comparației numărul rezultat este negativ (Cy=1), atunci tasta apărută a fost cifră sau punct.
- Ca numărător al tastărilor succesive ale tastei FUNC, vom folosi registrul C (linia 7 din descriere, respectiv linia 6 din program), element de reținut.

Buclo principală se regăsește în listingul din Cap. 17., pag. 1—9.

Trecem în continuare la elaborarea grosieră a celor două proceduri folosite în **MAINLOOP** : **BUYTICK** și **PROCFUNC**.

14.1.1. Bonul client normal : **BUYTICK**

Rememorînd specificația tehnică, rezultă că activitățile legate de emiterea unui bon client normal se pot sintetiza în felul următor :

- ea este declanșată prin introducerea unui preț ;
- pe parcursul ei se pot introduce oricîte prețuri separate prin tasta "+" ; între două prețuri se stă în repausul interpret (F.1.9.) ;
- procedura se termină prin apăsarea tastei **TOTAL**, care va declanșa emiterea fizică (imprimarea) a bonului client normal (F.1.10.) ;
- după terminarea secvenței se revine în repausul interbon (**MAINLOOP**) (F.1.11.).

Rezultă deci, că bucla principală a procedurii **BUYTICK** va fi cea care pornește și se întoarce în repausul interpret.

În fig. 14.2. redăm organigrama care se poate constitui pe baza celor enunțate mai sus.

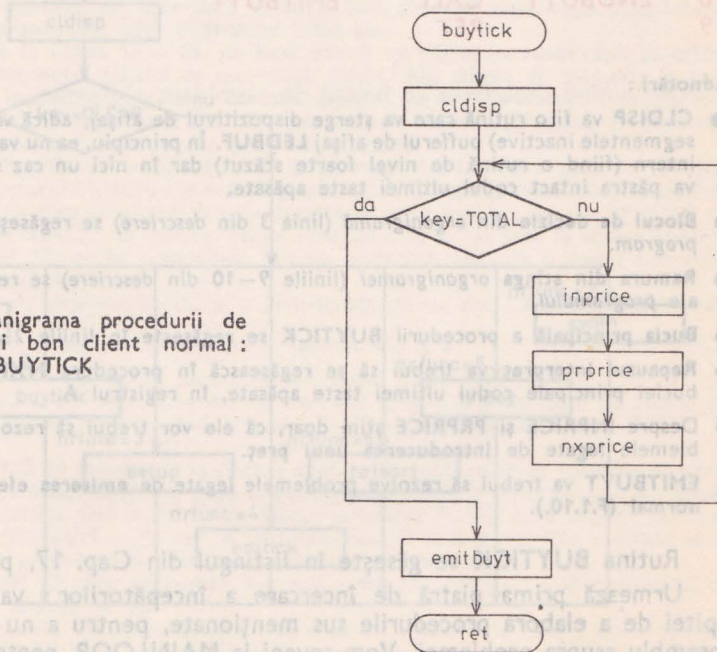


Fig. 14.2. Organigrama procedurii de emitere a unui bon client normal : **BUYTICK**

Redăm acțiunile din organigramă și într-o descriere în limbaj de nivel înalt.

```

procure buytick
1  begin
2      cldisp {ștergere afișaj}
3      while key ≠ TOTAL do
4          begin
5              inprice {se citește un preț de la tastatură}
6              prprice {se prelucrează prețul citit}
7              nxprice {începe citirea unui preț nou}
8          end
9      emitbuyt {se emite bonul client normal}
10 end

```

Rezultă programul scris în limbaj de asamblare :

```

1  BUYTICK : CALL    CLDISP
2  BTESTTOT : CP     TOTAL
3           JR      Z,ENDBUYT
4           CALL   INPRICE
5           CALL   PRPRICE
6           CALL   NXPRICE
7           JR     BTESTTOT
8  ENDBUYT : CALL    EMITBUYT
9           RET

```

Adnotări :

- **CLDISP** va fi o rutină care va șterge dispozitivul de afișaj, adică va umple cu FF_H (toate segmentele inactice) bufferul de afișaj LEDBUF. În principiu, ea nu va afecta nici un registru intern (fiind o rutină de nivel foarte scăzut) dar în nici un caz registrul A, în care se va păstra intact codul ultimei taste apășate.
- **Blocul de decizie** din organigramă (linia 3 din descriere) se regăsește în liniile 2 — 3 din program.
- **Ramura din stînga organigramei** (liniile 9—10 din descriere) se regăsesc în liniile 8 — 9 ale programului.
- **Bucula principală** a procedurii BUYTICK se regăsește în liniile 2—7 ale programului.
- **Repausul interpret** va trebui să se regăsească în procedura NXPRICE, care va restitui buclei principale codul ultimei taste apășate, în registrul A.
- Despre INPRICE și PRPRICE știm doar, că ele vor trebui să rezolve absolut toate problemele legate de **introducerea unui preț**.
- **EMITBUYT** va trebui să rezolve problemele legate de **emiterea efectivă a bonului client normal** (F.1.10.).

Rutina BUYTICK se găsește în listingul din Cap. 17, pag. 1—10.

Urmează prima piatră de încercare a începătorilor : va trebui să rezistăm ispitei de a elabora procedurile sus menționate, pentru a nu pierde viziunea de ansamblu asupra problemei. Vom reveni la MAINLOOP, pentru a-i trata cealaltă procedură : PROCFUNC.

14.1.2. Procesarea tastei FUNC : PROCFUNC

Tasta FUNC fiind multifuncțională, numărul de tastări succesive ale ei va trebui să declanșeze activități diferite ale casei de marcat. Aceste activități sînt impuse prin specificația funcțională (F.2.x. respectiv F.1.13. și F.1.15.).

■ Procedeele de a declanșa activități complet diferite prin tastarea repetată a uneia și aceleași taste, este o operație destul de exigentă pentru operator. De aceea nu se va uita că el poate greși, prevederea măsurilor, de ergonomie formulate în specificația funcțională (indicarea numărului de tastări succesive pe afișaj și prevederea posibilității de a anula o comandă înainte de a o executa efectiv, (F2.0.) fiind obligatorii.

În fig. 14.3. redăm organigrama procedurii PROCFUNC.

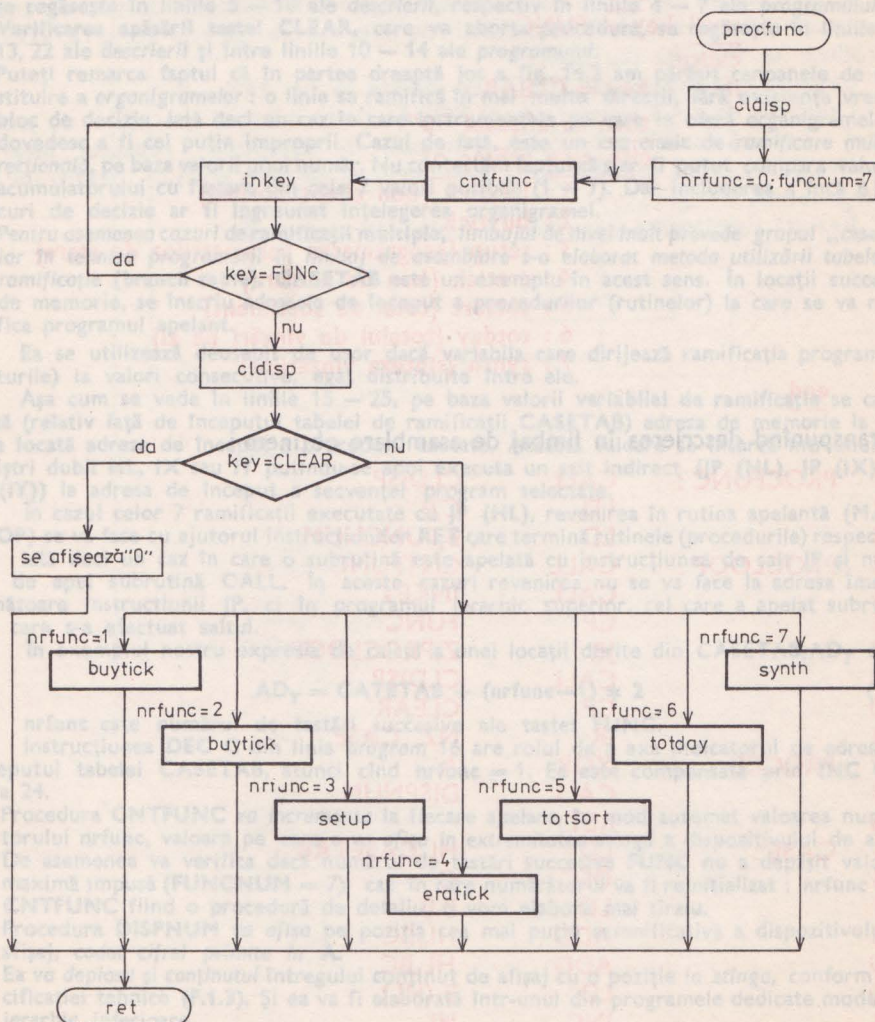


Fig. 14.3. Organigrama procedurii de prelucrare a tastărilor succesive FUNC : PROCFUNC

Elaborăm descrierea în limbaj de nivel înalt :

```

procedură procfunc
1  begin
2  cldisp {ștergere afișaj}
3  nrfunc := 0
4  funcnum := 7
5  repeat
6  begin
7  cntfunc {contorizare FUNC și afișare}
8  inkey
9  end
10 until key ≠ FUNC
11 cldisp
12 if
13 key = CLEAR
14 then se afișează "0"
15 else case nrfunc of
16 1 : buytick {bon client normal}
17 2 : buytick
18 3 : setup {programarea parametrilor de stare}
19 4 : eratick {bon de anulare}
20 5 : totsort {total de sortiment}
21 6 : totday {totalul de vânzări pe zi}
22 7 : synth {sinteza vânzărilor}
23 end

```

Transpunând descrierea în limbaj de asamblare obținem :

```

1  PROCFUNC :  CALL  CLDISP
2  LD  C,0
3  LD  B,FUNCNUM
4  PROCLOOP : CALL  CNTFUNC
5  CALL  INKEY
6  CP  FUNC
7  JR  Z,PROCLOOP
8  CALL CLDISP
10 CP  CLEAR
11 JR  NZ, CASE
12 PROCCL :  XOR  A
13 CALL DISPNUM
14 RET
15 CASE :    LD  HL,CASETAB
16 DEC  C
17 LD  B,0
18 ADD HL,BC
19 ADD HL,BC
20 LD  E,(HL)
21 INC  HL
22 LD  D,(HL)
23 EX  DE,HL

```

24
25
26
27
28
29
30
31
32

CASETAB :

INC	C
JP	(HL)
DW	BUYTICK
DW	BUYTICK
DW	SETUP
DW	ERATICK
DW	TOTSORT
DW	TODAY
DW	SYNTH

Adnotări :

- **Bucła de contorizare a tastărilor** (cea care include procedura cntfunc) din **organigramă**, se regăsește în liniile 5 — 10 ale **descrierii**, respectiv în liniile 4 — 7 ale **programului**.
- **Verificarea apăsării tastei CLEAR**, care va aborta procedura, se regăsește în liniile 12, 13, 22 ale **descrierii** și între liniile 10 — 14 ale **programului**.
- Puteți remarca faptul că în partea dreaptă jos a fig. 14.3 am părăsit canoanele de constituire a **organigramelor** : o linie se ramifică în mai multe direcții, fără existența vreunui bloc de decizie. Iată deci un caz în care instrumentele pe care le oferă organigramele se dovedesc a fi cel puțin impropii. Cazul de față, este un caz clasic de **ramificare multidirecțională**, pe baza valorii unui număr. Nu contestăm faptul că s-ar fi putut compara valoarea acumulatorului cu fiecare din cele 7 valori posibile (1 — 7). Dar includerea a încă 6 blocuri de decizie ar fi îngreunat înțelegerea organigramei.
Pentru asemenea cazuri de ramificații multiple, limbajul de nivel înalt prevede grupul „case-of” iar în tehnica programării în limbaj de asamblare s-a elaborat metoda utilizării tabeli de ramificație (branch table). **CASETAB** este un exemplu în acest sens. În locații succesive de memorie, se înscriu adresele de început a procedurilor (rutinelor) la care se va ramifica programul apelant.

Ea se utilizează deosebit de ușor dacă variabila care dirijează ramificația programului (salturile) ia valori consecutive, egal distribuite între ele.

Așa cum se vede în liniile 15 — 25, pe baza valorii **variabilei de ramificație** se calculează (relativ față de începutul tabeli de ramificații **CASETAB**) adresa de memorie la care este locată adresa de început a procedurii căutate. Această valoare se încarcă într-unul din regiștri dubli **HL**, **IX** sau **IY** putîndu-se apoi executa un salt indirect (**JP (HL)**, **JP (IX)** sau **JP (IY)**) la adresa de început a secvenței program selectate.

În cazul celor 7 ramificații executate cu **JP (HL)**, revenirea în rutina apelantă (**MAIN-LOOP**) se va face cu ajutorul instrucțiunilor **RET** care termină rutinele (procedurile) respective.

Iată deci un caz în care o subrutină este apelată cu instrucțiunea de salt **JP** și nu cu cea de apel subrutină **CALL**. În aceste cazuri revenirea nu se va face la adresa imediat următoare instrucțiunii **JP**, ci în programul ierarhic superior, cel care a apelat subrutina din care s-a efectuat saltul.

În exemplul nostru expresia de calcul a unei locații dorite din **CASETAB**, AD_T este :

$$AD_T = CATETAB + (nrfunc - 1) * 2 \quad (14.1)$$

nrfunc este numărul de tastări succesive ale tastei **FUNC**.

Instrucțiunea **DEC C** din linia **program 16** are rolul de a axa indicatorul de adresă pe începutul tabeli **CASETAB**, atunci cînd **nrfunc = 1**. Ea este compensată prin **INC C** în linia 24.

- Procedura **CNTFUNC** va incrementa la fiecare apelare, în mod automat valoarea numărătorului **nrfunc**, valoare pe care o va afișa în extremitatea stîngă a dispozitivului de afișaj. De asemenea va verifica dacă numărul de tastări succesive **FUNC** nu a depășit valoarea maximă impusă (**FUNCNUM = 7**), caz în care numărătorul va fi reinițializat : **nrfunc = 1**. **CNTFUNC** fiind o procedură de detaliu, o vom elabora mai tîrziu.
- Procedura **DISPNUM** va afișa pe poziția cea mai puțin semnificativă a dispozitivului de afișaj, codul cifrei primite în **A**. Ea va deplasa și conținutul întregului conținut de afișaj cu o poziție la stînga, conform specificației tehnice (F.1.3). Și ea va fi elaborată într-unul din programele dedicate modulelor ierarhic inferioare.
- Primele două cazuri (**nrfunc = 1** și **nrfunc = 2**) ne vor conduce la procedura deja prezentată **BUYTICK**. Este vorba de un preț introdus cu cod de sortiment (**nrfunc = 1**) sau un preț de ambalaj restituit (**nrfunc = 2**).

Listingul comentat al rutinei PROCFUNC se găsește în Cap. 17, pag. 1—11. Menținându-ne în continuare pe primul nivel ierarhic, vom elabora procedurile dedicate funcțiilor speciale ale casei de marcat.

14.1.3. Programarea sesiunii de lucru : SETUP

Modul de programare a parametrilor de stare ai sesiunii de lucru (*magazin, număr casă, număr casier, dată*) este impus prin specificația tehnică F.2.1.

■ Procedura **SETUP** va fi declanșată după trei tastări succesive ale tastei **FUNC**.

■ Pe parcursul programării cei 4 parametri vor fi separați prin apăsarea tastei **TOTAL**. În fiecare fază de introducere tasta **CLEAR** va fi activă permițând ștergerea unei date eronat introduse.

■ Prin natura lor, datele care urmează să fie introduse, ne scutesc de analiza corectitudinii. Singura măsură de protecție pe care o vom lua este aceea ca, codurile introduse să fie cifră sau punct.

Organigrama procedurii **SETUP** se găsește în fig. 14.4.

După cea de-a 4-a tastare a tastei **TOTAL** (la terminarea procesului de programare) urmează revenirea în bucla principală de așteptare : **repausul interbon (MAINLOOP)**. Vom semnaliza acest eveniment și pe cale acustică, pentru un plus de ergonomie.

Rezultă următoarea descriere în limbaj de nivel înalt :

```

1  procedure setup
2      begin
3          prog := 0
4          repeat
5              begin
6                  repeat
7                      begin
8                          while (key ≠ cifră)  $\wedge$  (key ≠ punct) do
9                              begin
10                                  inkey
11                                  if key = CLEAR then „reinițializare buffere”
12                                  end
13                              end
14                          stordisp {memorare și afișare cifră sau}
15                          {punct}
16                          inkey
17                          end
18                      until key = TOTAL
19                      transpar {parametrul introdus se transferă la locul lui}
20                      clbufdp {reinițializarea bufferelor }
21                      prog := prog + 1
22                  end
23              until prog = 4
24          se afișează "0"
25          sonerie
26      end

```

Procedura de programare a parametrilor de stare în cadrul sistemului de control este prezentată în figura 14.4. Aceasta este o procedură de nivel înalt care are ca scop introducerea unui set de parametri de stare în memoria de control. Procedura este prezentată în figura 14.4 și este o procedură de nivel înalt care are ca scop introducerea unui set de parametri de stare în memoria de control.

Listiugul comentat al rutinei de programare a parametrilor de stare este prezentat în Cap. 17 la pag. 144.

Emiterii unui bon de salutare și salutul funcțiilor de control în cazul de marcat, care nu se vor efectua în cazul în care nu este prezentă materializarea de control. În principiu procedura de control este prezentată în figura 14.4 și este o procedură de nivel înalt care are ca scop introducerea unui set de parametri de stare în memoria de control. Ea va conține în plus față de BUYTICK o secvență de control.

In fig. 14.5. redăm organizarea procedurii de control.

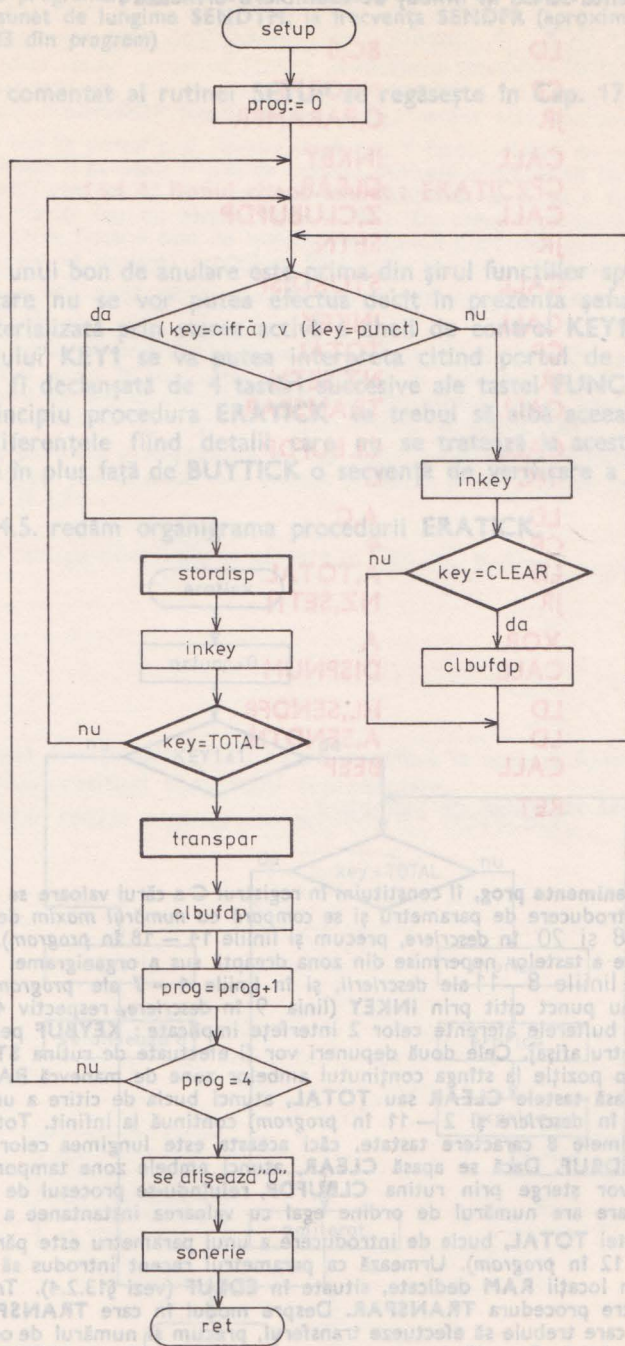


Fig. 14.4. Organigrama procedurii de programare a parametrilor de stare : SETUP

Rutina aferentă scrisă în limbaj de asamblare urmează :

1	SETUP :	LD	BC,0
2	SETN :	CP	NRORPT
3		JR	C,PARAMPR
4		CALL	INKEY
5		CP	CLEAR
6		CALL	Z,CLUBUFDP
7		JR	SETN
8	PARAMPR :	CALL	STORDISP
9		CALL	INKEY
10		CP	TOTAL
11		JR	NZ,SETN
12		CALL	TRANSPAR
13		CALL	CLBUFDP
14		INC	C
15		LD	A,C
16		CP	4
17		LD	A,TOTAL
18		JR	NZ,SETN
19		XOR	A
20		CALL	DISPNUM
21		LD	HL,SENDFR
22		LD	A,SENDTM
23		CALL	BEEP
24		RET	

Adnotări :

- Contorul de evenimente prog, îl constituim în registrul **C** a cărui valoare se incrementează după fiecare introducere de parametru și se compară cu numărul maxim de parametri 4. (vezi liniile 18 și 20 în descriere, precum și liniile 14 — 18 în program).
- Bucla de filtrare a tastelor nepermise din zona dreaptă sus a organigramei din fig. 14.4 se regăsește în liniile 8—11 ale descrierii, și în liniile 4—7 ale programului.
- Fiecare cifră sau punct citit prin **INKEY** (linia 9 în descriere, respectiv 4 în program) este depusă în bufferele aferente celor 2 interfețe implicate : **KEYBUF** pentru tastatură și **LEDBUF** pentru afișaj. Cele două depuneri vor fi efectuate de rutina **STORDISP** care va deplasa cu o poziție la stînga conținutul ambelor zone de manevră RAM.
- Dacă nu se apasă tastele **CLEAR** sau **TOTAL**, atunci bucla de citire a unui parametru (liniile 5—15 în descriere și 2—11 în program) continuă la infinit. Totdeauna vor fi disponibile ultimele 8 caractere tastate, căci aceasta este lungimea celor două buffere **KEYBUF** și **LEDBUF**. Dacă se apasă **CLEAR**, atunci ambele zone tampon (**KEYBUF** și **LEDBUF**) se vor șterge prin rutina **CLBUFDP**, reluîndu-se procesul de introducere a parametrului care are numărul de ordine egal cu valoarea instantanee a registrului **C**.
- La apăsarea tastei **TOTAL**, bucla de introducere a unui parametru este părăsită (linia 16 în descriere și 12 în program). Urmează ca parametrul recent introdus să fie transferat din **KEYBUF** în locații RAM dedicate, situate în **EDBUF** (vezi §13.2.4). Transferul va fi efectuat de către procedura **TRANSPAR**. Despre modul în care **TRANSPAR** va determina adresa la care trebuie să efectueze transferul, precum și numărul de octeți de transferat s-a mai vorbit în Cap. 13, la prezentarea tabelii de distribuție **SETUPTAB**, și se va mai vorbi la prezentarea procedurii însăși. Cert este că ea (**TRANSPAR**) va folosi registrul **C** ca identificator al parametrului de transferat.

- Procedura de programare a parametrilor de stare a casei de marcat se termină prin emisia unui sunet de lungime SENDTM, la frecvența SENDFR (aproximativ 3 kHz), (vezi liniile 21—23 din program)

Listingul comentat al rutinei SETUP se regăsește în Cap. 17 la pag. 1—14.

14.1.4. Bonul client anulat : ERATICK

Emiterea unui bon de anulare este prima din șirul funcțiilor speciale ale casei de marcat, care nu se vor putea efectua decât în prezența șefului de unitate, prezență materializată prin starea activă a cheii de control KEY1 (F.2.2.). Activarea semnalului KEY1 se va putea interpreta citind portul de intrare SYSIN. Procedura va fi declanșată de 4 tastări succesive ale tastei FUNC.

■ În principiu procedura ERATICK va trebui să aibă aceeași structură cu BUYTICK, diferențele fiind detalii care nu se tratează la acest nivel ierarhic. Ea va conține în plus față de BUYTICK o secvență de verificare a prezenței cheii de control.

În fig. 14.5. redăm organigrama procedurii ERATICK.

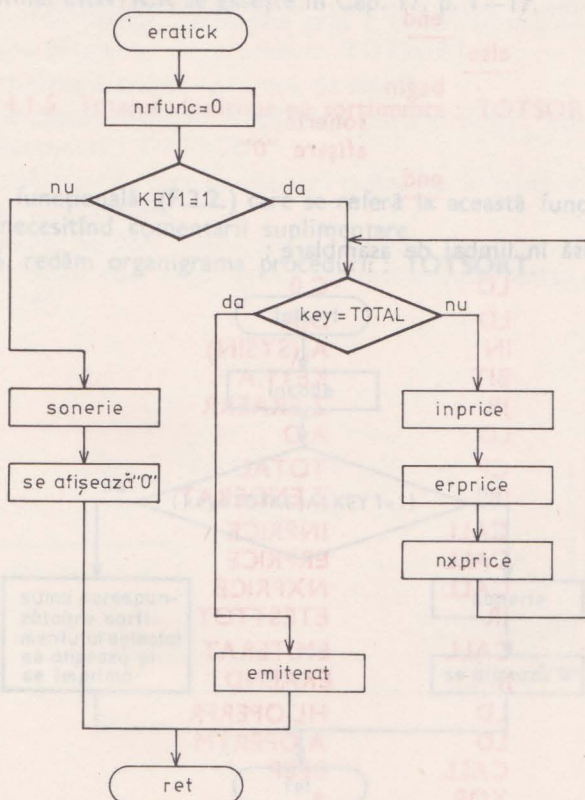


Fig. 14.5. Organigrama procedurii de emisie a unui bon de anulare : ERATICK

● În ramura din stînga a fig. 14.5. se vede că orice încercare de a declanșa procedura ERATICK, în absența cheii de control, este rejectată, emițîndu-se totodată un semnal sonor de avertizare.

```

procedură eratick
  begin
    2   nrfunc := 0
    3   if key 1=1
    4     then
    5     begin
    6     while key ≠ TOTAL do
    7       begin
    8         inprice {se citește un preț de la
    9           tastatură}
    10        erprice {se anulează}
    11        nxprice {începe citirea unui nou
    12          preț}
    13      end
    14      emiterat {se emite bonul de anulare}
    15    end
    16  else
    17    begin
    18      sonerie
    19      afișare "0"
    20    end
  end

```

Rutina scrisă în limbaj de asamblare :

```

1 ERATICK :   LD      C,0
2             LD      D,A
3             IN      A,(SYSIN)
4             BIT     KEY1,A
5             JR      Z,ERAERR
6             LD      A,D
7 ETESTTOT : CP      TOTAL
8             JR      Z,ENDERAT
9             CALL   INPRICE
10            CALL   ERPRICE
11            CALL   NXPRICE
12            JR      ETESTTOT
13 ENDERAT :  CALL   EMITERAT
14            JR      ERAEND
15 ERAERR :  LD      HL,OPERFR
16            LD      A,OPERTM
17            CALL   BEEP
18            XOR     A
19            CALL   DISPNUM
20 ERAEND :  RET

```

Adnotări :

- Datorită faptului că procedura **INPRICE** va citi eventual și prețuri precedate de codul de marfă (deci tastări succesive **FUNC**) numărătorul aferent — **nrfunc** — va trebui să fie inițializat întocmai ca și în cazul procedurii **BUYTICK**. (Reținem faptul că **BUYTICK** primește din **MAINLOOP** mereu **nrfunc = 0**); **nrfunc** este contorizat în registrul **C**.
- Constatăm aici că **primul preț introdus după cele 4 tastări FUNC nu poate fi precedat de cod de sortiment și nu poate fi preț de ambalaj**. Ambele ar trebui să fie precedate de noi tastări **FUNC**, ceea ce ar incrementa **nrfunc**, selectându-se astfel o altă funcție specială a casei de marcat (nu s-a părăsit **PROCFUNC**). De aceea recomandăm operatorului să înceapă emiterea fiecărui bon de anulare introducând după cele patru tastări **FUNC**, un preț „0”, care va declanșa procedura **ERATICK**, și nu va afecta sumele memorate.
- Ramura de evitare (lipsă **KEY1**) apare în descriere în liniile 14–18, iar în program între liniile 15 și 20.
- Remarcăm faptul că primul argument al instrucțiunii **BIT** din linia program 4 este un simbol. Astfel rutina își va păstra valabilitatea și în condițiile modificării hardware-ului, programatorul va atribui simbolului **KEY1** o altă valoare. La o nouă asamblare aceste valori se vor substitui în toate instrucțiunile care au folosit **KEY1**.
- Procedurile **ERPRICE** și **EMITERAT** nu vor diferi structural de **PRPRICE** și **EMITBUYT**. În esență ele vor inversa sensul unor operații aritmetice și vor imprima un mesaj special pe bonul emis (**F.2.2.**).

Listingul rutinei **ERATICK** se găsește în Cap. 17, p. 1–17.

14.1.5. Totalul vânzărilor pe sortimente : **TOTSORT**

Specificația funcțională (**F.2.2.**) care se referă la această funcție specială este clară, ea nemăinecșitînd comentarii suplimentare.

În fig. 14.6. redăm organigrama procedurii : **TOTSORT**.

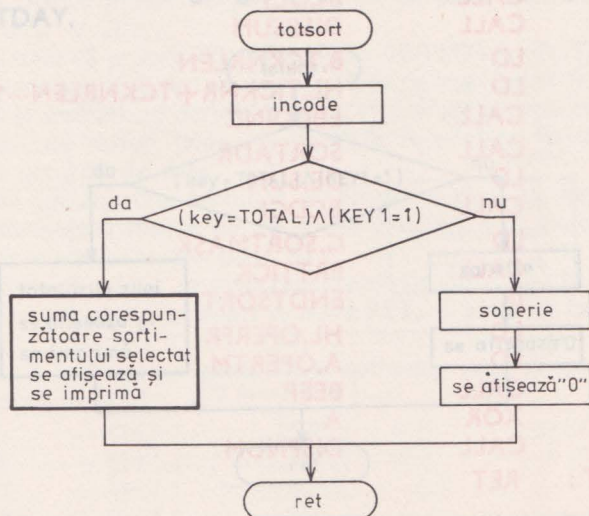


Fig. 14.6. Organigrama procedurii de emitere a unui bon cuprinzînd totalul unui sortiment : **TOTSORT**

Descrierea în limbaj de nivel înalt a fig. 14.6. este :

```

1 procedură totsort
1   begin
2     incod {se citește de la tastatură codul sortimentului dorit}
3     if (key = TOTAL)  $\wedge$  (KEY1=1)
4       then
5         begin
6           "suma corespunzătoare sortimentului selectat,
7             se imprimă și se afișează"
8         end
9       else
10        begin
11          sonerie
12          afișare „0”
13        end
14      end

```

Descrierea în limbaj de nivel înalt este destul de vagă, în speță în linia 6. Vom detalia această linie în programul scris în limbaj de asamblare.

```

1 TOTSORT :   CALL   INCODE
2             CP     TOTAL
3             JR     ERRTSORT
4             IN     A,(SYSIN)
5             BIT   KEY1,A
6             JR     Z,ERRTSORT
7             CALL  SORTADR
8             LD    DE,PRTFBUF+4
9             CALL  BCDCI
10            CALL  DISPSUM
11            LD    B,TCKNRLEN
12            LD    HL,TICKNR+TCKNRLEN-1
13            CALL  EBCDINC
14            CALL  SORTADR
15            LD    DE,SUM
16            CALL  BCDCI
17            LD    C,SORTMASK
18            CALL  PRTTICK
19            JR     ENDTSORT
20 ERRTSORT :  LD     HL,OPERFR
21            LD     A,OPERTM
22            CALL  BEEP
23            XOR   A
24            CALL  DISPNUM
25 ENDTSORT :  RET

```

Adnotări :

- Procedura **INCOD** va citi un cod de sortiment format din 2 cifre și îl va depune în **EDBUF** la intrarea **CODE** (vezi fig. 13.10). Pe parcursul citirii celor două cifre tasta **CLEAR** va fi activă (vezi specificația funcțională F.1.13 și F.1.15).

- Linia 6 din descriere s-a transpus în liniile 7 – 19 din program.
- Procedura SORTADR va determina adresa de început a registrului BCD în RAM (tabela SORTTOT) care conține totalul vânzărilor aferente codului de sortiment din CODE. SORTADR va restitui în HL adresa de început a registrului căutat. (structura tabelului SORTTOT se găsește în §13.3.3).
- Sumele și prețurile sînt memorate în cod BCD, ele vor trebui convertite în cod intern. Rutina BCDCI convertește valoarea BCD din registrul RAM pontat de HL și îl depune într-o zonă adresată de DE. (În cazul de față PRTBUF).
- Procedura DISPSUM va afișa un număr din PRTBUF. DISPSUM elimină și zerourile nesemnificative din stînga numărului.
- Folosind masca de selecție SORTMASK, PRTTICK va imprima conținutul lui EDBUF, selectînd liniile care au bitul de coincidență setat (vezi §13.2.4).
- Procedura TOTSORT emite același sunet (caracterizat prin durata OPERTM și frecvența OPERFR) ca și ERATICK în caz de operare greșită :
 - fie că nu este prezentă cheia de control (liniile 4–6 din program) ;
 - fie că nu s-a apăsat tasta TOTAL după introducerea codului de sortiment selectat.

Listingul comentat al rutinei TOTSORT se găsește în Cap. 17, pag. 1–18.

14.1.6. Totalul vânzărilor pe zi : TOTDAY

Procedura acestei funcții, declanșată prin 6 tastări succesive a tastei FUNC, diferă prin puține elemente de procedura TOTSORT :

- nu este necesară citirea unui cod de marfă ;
- nu trebuie căutat registrul RAM care conține suma ; cerută, căci adresa ei este cunoscută : DAYBCD ;
- masca de selecție cu care se apelează rutina de imprimare bon, PRTTICK, este alta : DAYMASK.

În aceste condiții nu vom insista asupra modului de elaborare a programului, publicînd în continuare doar organigrama și descrierea în limbaj de nivel înalt a procedurii TOTDAY.

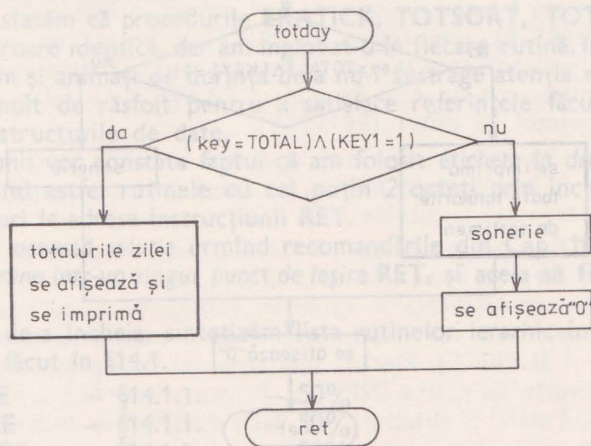


Fig. 14.7. Organigrama procedurii de emitere a unui bon cuprinzînd totalul vânzărilor pe zi : TOTDAY

Descrierea :

procedură totday

begin

if (key=TOTAL) \wedge (KEY1=1)

then

begin

„totalul zilei din DAYBCD se imprimă
și se afișează „”

end

else

begin

sonerie
afișează „0”

end

end

În Cap. 17, pag. 1—19 se găsește listingul comentat al rutinei TOTDAY.

14.1.7. Sinteza vânzărilor : SYNTH

Specificată funcțional (F.2.5.), această comandă se declanșează prin 7 taste succesive FUNC, urmată de tasta TOTAL.

■ Se vor lista în ordinea crescătoare a codurilor de sortiment toate sumele aferente.

Organigrama procedurii SYNTH se găsește în fig. 14.8.

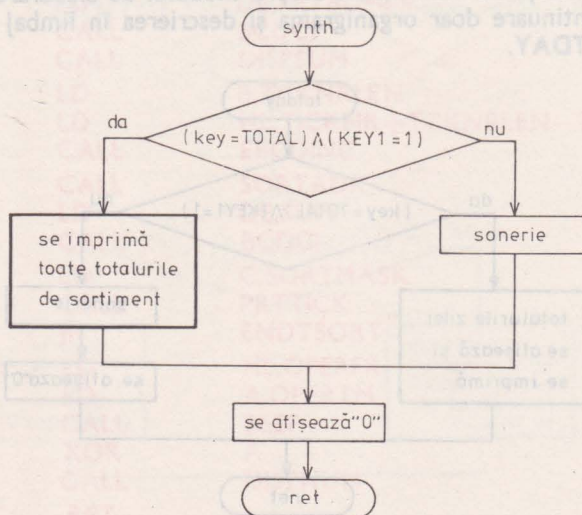


Fig. 14.8. Organigrama procedurii de listare a sintezei vânzărilor SYNTH

Descrierea procedurii în limbaj de nivel înalt este la fel de compactă :

```
procedure synth  
  begin  
    if (key=TOTAL) ^ (KEY1=1)  
      then "se imprimă sumele aferente sortimentelor"  
      else „se sună soneria”  
        „se afișează "0"”  
  end
```

Imprimarea celor 100 de linii de date o lăsăm pe seama rutinei **PRTTICK**, care primind masca de selecție **SYNTMASK**, va dispune de toate informațiile necesare pentru a imprima bonul de sinteză. Ea va prelua rînd pe rînd conținuturile regiștrilor **RAM** din **SORTTOTT** și le va imprima.

Listingul comentat al acestei proceduri se găsește în Cap. 17, pag. 1—20.

Prin elaborarea ultimei funcții majore considerăm **nivelul ierarhic zero** al **software-ului** casei de marcat epuizat.

Înainte de a aborda următorul nivel vom sintetiza activitatea desfășurată :

1. Procedurile **nivelului ierarhic zero** reprezintă o implementare fidelă a sarcinilor impuse prin specificația funcțională. Scopul principal urmărit n-a fost cel de-a etala tehnici elevate de programare în limbaj de asamblare, ci de a prezenta modul în care o specificație tehnică trebuie transpusă în program, și de a obișnui cititorul cu un stil de muncă : specificație, organigramă sau descriere în limbaj de nivel înalt, program.

2. Aceste proceduri nu rezolvă nimic concret, putînd fi plictisitoare, chiar monotone. Ele sînt totuși importante fiindcă crează cadrul întregului software, și impun trăsăturile specifice modulelor ierarhic inferioare.

3. Pe parcursul elaborării acestui nivel am încercat să ne conformăm la sugestia dată în Cap. 10, ca fiecare modul să fie cît se poate de autonom evitîndu-se salturile de la unul la altul.

Astfel constatăm că procedurile **ERATICK**, **TOTSORT**, **TODDAY** și **SYNTH** au ramura de eroare identică, dar am înglobat-o în fiecare rutină, în ideea enunțată mai sus, precum și animați de dorința de-a nu-i sustrage atenția cititorului, care a avut oricum mult de răsfoit pentru a satisface referințele făcute la specificația tehnică, și la structurile de date.

4. Peate unii vor constata faptul că am folosit etichete în dreptul instrucțiunilor **RET**, lungind astfel rutinele cu cel puțin 2 octeți prin includerea în corpul lor a unor salturi la adresa instrucțiunii **RET**.

Am ales și această soluție urmînd recomandările din Cap. 10, ca fiecare sub-rutină să se termine într-un singur punct de ieșire **RET**, și acela să fie ultima instrucțiune din listă.

5. Înainte de-a încheia, sintetizăm lista rutinelor ierarhic inferioare, a căror specificație s-a făcut în §14.1.

- | | | |
|-------------|------------|--------|
| a) INPRICE | — §14.1.1. | — 50% |
| b) PRPRICE | — §14.1.1. | — 50% |
| c) NXPRICE | — §14.1.1. | — 50% |
| d) EMITBUYT | — §14.1.1. | — 25% |
| e) CLDISP | — §14.1.1. | — 100% |
| f) CNTFUNC | — §14.1.2. | — 100% |

g)	STORDISP	— §14.1.3.	— 90%
h)	TRANSPAR	— §14.1.3.	— 100%
i)	CLBUFDP	— §14.1.3.	— 100%
j)	ERPRICE	— §14.1.4.	— 50%
k)	EMITERAT	— §14.1.4.	— 25%
l)	DISPNUM	— §14.1.2.	— 90%
m)	INCODE	— §14.1.5.	— 100%
n)	SORTADR	— §14.1.5.	— 100%
o)	BCDCI	— §14.1.5.	— 90%
p)	DISPSUM	— §14.1.5.	— 90%
q)	PRTTICK	— §14.1.7.	— 60%

14.2. Programe de prelucrare

Procedurile cuprinse în acest paragraf formează nivelul ierarhic 1 al software-ului casei de marcat. Spre deosebire de nivelul ierarhic zero, care stabilește cadrul (ambianța) de lucru și definește proceduri, *nivelul 1 va fi mai concret*, rezolvând chiar dacă nu pînă la ultimul detaliu, funcțiile impuse prin specificația tehnică, folosind căile inaugurate în nivelul ierarhic zero.

14.2.1. Citirea unui preț : INPRICE

Oricine se poate întreba de ce cele trei proceduri legate de pregătirea unui preț : citirea (INPRICE), prelucrarea (PRPRICE), și pregătirea următorului preț (NXPRICE) nu formează corpul unei singure proceduri ? Răspunsul este simplu : procedura de emisie a unui bon de anulare ERATICK va folosi prima și ultima procedură, dar miezul va diferi (ERPRICE).

A doua întrebare care se ridică se referă la rolul procedurii NXPRICE. De ce nu poate fi înglobată în INPRICE ? După analiza organigramelor capitolului precedent răspunsul se obține și în acest caz, relativ ușor.

■ În INPRICE se poate ajunge din două puncte de plecare : repaosul interbon și repaosul interpret. Între cele două situații apare deosebirea esențială că în repaosul interbon (MAINLOOP), tastările FUNC (una sau două) care preced un eventual cod de marfă, vor fi deja prelucrate în momentul apelării procedurii BUYTICK (deci și INPRICE). Plecînd din repaosul interpret, aceste operații ar trebui să fie efectuate de rutina INPRICE. Pentru a-i conferi o funcție clar definită, simplificîndu-i astfel și structura, am decis constituirea procedurii NXPRICE.

■ NXPRICE va funcționa ca un moderator : grație ei, procedura INPRICE va fi apelată totdeauna în momentul în care deja va fi fost tastat primul caracter din codul de sortiment, sau din valoarea prețului care urmează a fi introdus.

Plecînd de la aceste considerente rezultă descrierea de mai jos :

procedur **inprice**

```

1  begin
2  if nrfunc  $\neq$  0 then incode { se citește de la tastatură co-
3  else implcode { codul este implicit 99}
4  repeat
5  begin
6  stordisp {cifra sau punctul se memorează și se afișează}
7  repeat
8  begin
9  inkey
10 if key = CLEAR then clbufdp {reiniția-
    lizare}
11 if key = * then multop {pregătirea
    înmulțirii}
12 end
13 until (key = cifră)  $\vee$  (key = punct)  $\vee$  (key = +)
14 end
15 until (key = +)
16 end

```

Programul aferent va mai preciza cîteva elemente.

```

1  INPRICE : LD D,A
2  LD A,C
3  OR A
4  LD A,D
5  JR Z, CODELESS
6  CODED : CALL INCODE
7  AFTERCOD : CP NRORPT
8  JR C, PROCDIG
9  CALL INKEY
10 JR AFTERCOD
12 CODELESS : LD C,9
13 CALL IMPLCODE
14 LD HL, NRFUNC
15 LD (HL), 1
16 PROCDIG : CALL STORDISP
17 NEXTKEY : CALL INKEY
18 CP CLEAR
19 CALL Z, CLBUFD
20 CP ASTER
21 CALL Z, MULTOP
22 CP NRORPT
23 JR C, PROCDIG
24 CP PLUS
25 JR NZ, NEXTKEY
26 RET

```


Adnotări :

Evenimentele importante legate de introducerea unui preț se regăsesc în programul de sus. Iată-le:

- Înainte de toate, se verifică (pe baza valorii lui `nrfunc` (aici în registrul C) dacă caracterul conținut în A face parte din preț, sau dintr-un cod de sortiment care precede prețul. Dacă prețul s-a tastat fără cod de marfă (`C=0`) atunci se va genera în mod implicit codul 99 (conform specificației funcțiilor F.1.12). Secvența se regăsește în liniile program 12—13.
- dacă `nrfunc` diferit de zero înseamnă că prețul ce urmează a fi introdus, este precedat de cod de sortiment. Intregul cod va fi citit de procedura **INCODE** (linia 6).
- în afara misiunii de a citi cel de-al doilea caracter din cod, și eventual să permită folosirea tastei **CLEAR** pentru corectarea erorilor din cod, **INCODE** va trebui să genereze și un indicator, preferabil în memoria RAM, care va informa programele de prelucrare **PRPRICE** și **ERPRICE** asupra numărului de tastări **FUNC** care a precedat codul respectiv de sortiment (reamintim că o tastare **FUNC** (`nrfunc=1`) înseamnă vânzare, deci bani intrați în casă, iar 2 tastări **FUNC** (`nrfunc=2`) reprezintă răscumpărare de ambalaj, deci bani ieșiți din casă). **INCODE** va genera în RAM un indicator pe care-l vom numi **NRFUNC**. Această variabilă va conține numărul de tastări **FUNC**, care au precedat introducerea unui cod de sortiment. Și în cazul în care prețul s-a tastat fără cod de sortiment, **NRFUNC** trebuie actualizat, de astă dată de către **INPRICE** (vezi liniile program 14—15)
- La adresa **PROCDIG** (linia 16) se ajunge mereu cu caractere valide (cifră sau punct) ale prețului. Ele se depun în **KEYBUF** și se afișează (**LEDBUF**).
- Secvența program cuprinsă între liniile 17—26 citește următoarele caractere ale prețului, așteptând sosirea unui terminator:
 - dacă se tastează **CLEAR**, se șterge afișajul și bufferul de intrare **KEYBUF**, după care se va relua introducerea prețului, (deja fără cod) linia 19, **CLBUFD**;
 - dacă se tastează semnul înmulțirii "*", se vor lua măsuri pregătitoare prin procedura **MULTOP** (linia 21). **MULTOP** va trebui să preia prețul, să efectueze asupra lui toate verificările și să-l convertească în format **BCD** și să-l depună într-unul din regiștri **BCD** (**TMPBCD** sau **DATBCD**). Totodată, **MULTOP** va înscrie un indicator, indicator care va fi verificat la apariția unei taste "+". Dacă indicatorul de înmulțire va fi activ atunci, după tastarea semnelui "+" va trebui efectuată o înmulțire.
- ieșirea din rutina **INPRICE** se va face doar la apariția tastei "+" vezi liniile 24—26 din program.

14.2.2. Prelucrarea unui preț : PRPRICE

La ieșirea din rutina **INPRICE**, în **KEYBUF** se află un număr cu max. 8 cifre semnificative, care ar trebui să reprezinte un preț de produs.

■ Înainte de a aduna valoarea lui la totalul clientului, și de a-l imprima, va trebui să se verifice corectitudinea sa sintactică, pentru ca apoi să fie normalizat (2 cifre zecimale), iar doar după aceea convertit în format **BCD**, pentru a putea fi manipulat (adunat, scăzut, înmulțit). Abia după ce s-au efectuat aceste manevre pregătitoare, se vor putea întreprinde activitățile specifice de casă de marcat.

Iată algoritmul propus :

```
procedure prprice
1 begin
2   syntan {se analizează prețul introdus}
3   if "corect" then
4     begin
5     prelproc {prețul se transformă în BCD și dacă}
               {este indicat, se execută înmulțirea}
```



```

6      addsort { se adună la totalul de sortiment }
           { corespunzător codului }
7      opforbuy { se execută operațiile legate de bonul clien-
           tului : prețul se adună la totalul clien-
           tului, sau se scade din totalul clientului, noul
           total se afișează, iar prețul se imprimă }
8      end
9      end

```

Rutina aferentă va fi :

```

1  PRPRICE :   CALL   SYNTAN
2              JR     NZ,ENDPRPR
3              CALL   PRELPROC
4              CALL   ADDSORT
5              LD     HL,PRTBUF+15
6              LD     (HL),PLUS
7              CALL   OPFORBUY
8  ENDPRPR :   RET

```

Adnotări :

- Procedura **SYNTAN** va verifica corectitudinea șirului de cifre și puncte, din **KEYBUF**, și dacă îl, va găsi corect, îl va depune în **EBCD** (vezi §13.3.1). În caz de eroare, ea va returna rutinei apelante flagul **Z** resetat (**Z=0**). În acest caz, **PRPRICE** va fi ocolit complet. (Ca eroare posibilă amintim un număr cu mai multe puncte zecimale).
- Procedura **PRELPROC** va converti numărul din format **BCD extins** în format **BCD**, și-l va depune în **DATBCD**. Dacă indicatorul de înmulțire va fi activ, va efectua înmulțirea celor două numere (**DATBCD** și **TMPBCD**).
- Procedura **ADDSORT** adună la valoarea curentă a registrului **BCD** afectat sortimentului, care are codul specificat în **CODE**, numărul **DATBCD**.
- Procedura **OPFORBUY** va completa totalul clientului (registru **GUESTBCD**) și va imprima un rând de preț, marcând semnul "+", în dreptul cifrei celei mai puțin semnificative (liniile 5-7); se afișează suma curentă a clientului, pe dispozitivul de afișaj. Ea va fi aceea, care va folosi variabila **RAM NRFUNC**, specificată la prezentarea rutinei **INPRICE** pentru a decide sensul de manevrare a banilor intrați în casă sau ieșiți din casă.

14.2.3. Anularea unui preț : ERPRICE

Procedura **ERPRICE** apelată în procedura de emitere a unui bon de anulare (**ERATICK**) va avea aceeași structură ca și **PRPRICE**, diferențele constă în :

- Linia de preț imprimată va fi marcată cu "A" în loc de "+".
- Suma introdusă se va scade din totalul de sortiment (**SUBSORT**).
- Va trebui să fie prevăzută cu un test suplimentar care interzice operația de anulare, dacă suma totală aferentă oricărui sortiment implicat devine negativă.

Listingul comentat al acestei proceduri se regăsește în Cap. 17, pag. 1-23.

14.2.4. Pregătirea următorului preț : NXPRICE

Funcțiile acestei proceduri au fost specificate la descrierea rutinei INPRICE (§14.2.1) vom trece direct la elaborarea algoritmului aferent.

procedura nxprice

```

1  begin
2      nrfunc := 0
3      maxfunc := 2
4      repeat
5          begin
6              inkey
7              if key = FUNC then cntfunc {contorizare FUNC}
8                                     {și afișare}
9          until (key=cifră) V (key=punct) V (key=TOTAL)
10         clbufdp
11     end

```

Rutina în limbaj de asamblare rezultă :

```

1  NXPRICE :   LD      C,0
2              LD      B,MAXFUNC
3  BEGPR :     CALL   INKEY
4              CP      FUNC
5              CALL   Z,CNTFUNC
6              CP      NRRPT
7              JR      C,ENDNXPR
8              CP      TOTAL
9              JR      NZ,BEGPR
10 ENDNXPR :   CALL   CLBUFDP
11              RET

```

Adnotări :

- În secvența cuprinsă între liniile 3—9 identificăm *repaosul interpret*.
- Din această secvență nu se va putea ieși decât prin apăsarea unei cifre sau punct (liniile 6, 7, 10, 11), considerat a fi primul caracter al unui cod de produs sau al unui preț, sau apăsând tasta TOTAL, care va provoca imprimarea efectivă a bonului client normal, sau de anulare (liniile 8—11).
- NXPRICE va limita tastările succesive FUNC, afișând o numărare cu secvența : 1, 2, 1, 2, ... impusă prin valoarea maximă admisă de tastări succesive : MAXFUNC=2. (Reamintim că din repaosul interpret (conform specificației funcționale) nu se vor putea lansa funcții speciale ale casei de marcat (cele care ar necesita mai mult decât 2 tastări succesive FUNC).

14.2.5. Citirea codului de sortiment : INCODE

■ Procedura de citire a unui cod de sortiment, format obligatoriu din 2 cifre, INCODE, a fost specificată funcțional la prezentarea procedurii de citire a unui preț INPRICE (vezi §14.2.1)

Structura ei fiind foarte simplă, ne permitem să elaborăm direct programul în limbaj de asamblare :

```

1  INCODE :   LD           HL,NRFUNC
2            LD           (HL),C
3            LD           B,SCLEN
4  CODDIG :   CP           POINT
5            JR           C,PRCODDIG
6            CALL        INKEY
7            JR           CODDIG
8  PRCODDIG : CALL        STORDISP
9            CALL        INKEY
10           CP           CLEAR
11           CALL        Z,CLEARCOD
12           DJNZ        CODDIG
13           LD           HL,KEYBUF+BUFLEN-SCLEN
14           LD           DE,CODE
15           LD           BC,SCLEN
16           LDIR
17           CALL        CLBU FDP
18           RET
19  CLEARCOD : CALL        CLBU FDP
20           LD           B,SCLEN+1
21           RET

```

Adnotări :

- Foarte importantă acțiunea din liniile 1—2: **contorul din RAM** al tastărilor succesive **FUNC**, este *actualizat* cu valoarea cuprinsă în **C**.
- **SCLEN** (Sort Code **LE**ngth) precizează *lungimea codului de sortiment* (2 în cazul nostru)
- În *prima buclă* (liniile 4—7) se *filtrează* toate tastele care *diferă de cifre*; știind că prima tastă după **FUNC**, va trebui să fie obligatoriu cifră, (nu se va admite nici **CLEAR**).
- În *secvența principală* (liniile 8—11) codul cifrei tastate se vizualizează și se depune în **KEYBUF** (**STORDISP**).
- În cazul în care se tastează **CLEAR**, se *apelează rutina CLEARCOD*: care va șterge cele 2 buffere **KEYBUF** și **LEDBUF**, și va reinițializa contorul de evenimente din **B**.
- Citirea cifrelor de cod continuă pînă cînd **B=0** (linia 12).
- **INCODE** se termină printr-o secvență care *transpune codul tastat, din KEYBUF, în locul de destinație CODE* (din **EDBUF**).
- **BUFLEN** este lungimea bufferului **KEYBUF**.

14.2.6. Generarea implicită a codului de sortiment : IMPLCODE

■ Misiunea acestei proceduri este deosebit de simplă. Ea va trebui să înscrie în zona dedicată codului de sortiment (**CODE** din **EDBUF**) codul 99, rezervat pentru mărfurile a căror preț se introduce fără specificarea codului de sortiment.

Listingul comentat al rutinei **IMPLCODE** se găsește în Cap. 17, pag. 1—26.

14.2.7. Numărător pentru tastări succesive FUNC : CNTFUNC

Această procedură va fi apelată la detectarea apăsării tastei FUNC

■ Ea va incrementa numărătorul **nrfunc** (din registrul C), și îi va afișa valoarea în extremitatea din stînga a dispozitivului de afișaj.

■ Dacă registrul C depășește o valoare impusă la apelare (**funcnum** sau **maxfunc** în B), atunci **CNTFUNC** va reinițializa contorul : **C = 1**.

```

procedură cntfunc
1   begin
2       nrfunc := nrfunc+1
3       if nrfunc < funcnum then nrfunc = 1
4       cdisp
5       begin
6           „citește codul de comandă segmente al nrfunc din LEDGEN”
7           „depune codul obținut în LEDBUF+0”
8       end
9       sonerie
10  end
    
```

Implementarea acestei proceduri nu ridică probleme. O redăm totuși, pentru exemplificarea accesului la generatorul de coduri pentru afișaj LEDGEN.

```

1  CNTFUNC :  PUSH    AF
2              INC     C
3              LD      A,B
4              CP      C
5              JR      NC,DISPNR
6              LD      C,1
7  DISPNR :  CALL    CLDISP
8              LD      B,0
9              LD      HL,LEDGEN
10             ADD     HL,BC
11             LD      B,A
12             LD      A,(HL)
13             LD      DE,LEDBUF+0
14             LD      (DE),A
15             LD      HL,FCSDFRE
16             LD      A,FCSDTIME
17             CALL   BEEP
18             POP     AF
19
    
```

Adnotări :

- Incrementarea **nrfunc** se face în linia 2.
- Testarea valorii limitei impuse, și o eventuală reinițializare a numărătorului **nrfunc** se fac în liniile 3–6.
- În secvența 8–10 se calculează adresa din generatorul de coduri comandă segmente, la care se află locat codul cifrei egale cu **nrfunc**.
- Codul de comandă se depune în **LEDBUF** pe poziția extremă stîngă, în liniile 13–14.
- Ultima secvență (15–17) emite un sunet scurt pentru a marca fiecare tastare **FUNC**.

14.2.8. Operații pentru client : OPFORBUY

■ Procedura **OPFORBUY** se apelează în momentul în care ultimul preț introdus se află, validat, în **DATBCD**, iar **NRFUNC** indică tipul operației de efectuat. Ea va efectua strict operații pentru client (totalul de sortiment a fost deja actualizat în **PRPRICE** sau **ERPRICE**, prin apelul uneia din rutinele **ADDSORT** sau **SUBSORT**).

```
procedura opforbuy
1   begin
2       if nrfunc = 1
3           then if code > 9
4               then „se adună prețul introdus la totalul clientului”
5               else if code ≤ 9
6                   then
7                       begin
8                           „se schimbă în bufferul de tipărire + cu
9                           „se scade prețul introdus din totalul clien-
10                          tului”
11                          end
12                          if „nu-s-a detectat eroare”
13                              then
14                                  begin
15                                      „se afișează noul total al clientului”
16                                      „se tipărește prețul introdus pe bon”
17                                  end
18                              else
19                                  begin
20                                      „se scade prețul introdus din totalul de
21                                      sortiment corespunzător”
22                                      „se afișează vechiul total al clientului”
23                                  end
24                          end
25   end
```

Adnotări :

- Dacă **NRFUNC** = 1 atunci codul de sortiment va trebui să fie cuprins în gama de valori 10—99. In caz contrar, operatorul a greșit, caz în care operațiile deja efectuate trebuie anulate și comanda va fi rejectată (liniile 18—22 din descriere).
- Dacă **NRFUNC** = 2, atunci codul de sortiment va trebui să fie, cuprins între 0—9. In caz contrar se va semnala aceeași eroare operator.
- Esența procedurii este cuprinsă în liniile 4, 9, 14, 15 ale descrierii.
- Adunarea și scăderea vor apela rutinele de aritmetică **BCD**.
- Vizualizarea se va face cu rutina **DISPSUM**.
- Imprimarea liniei de preț se va face cu **PRTLINE**.

Listingul comentat al rutinei **OPFORBUY** se găsește în Cap. 17 pag. 1—27, 1—28.

14.2.9. Emiterea bonului client normal : EMITBUYTICK

■ Această procedură va fi declanșată de apăsarea tastei **TOTAL** în *repaosul interbon (MAINLOOP)* sau în *repaosul interpret (NXPRICE)*.

■ Apelul ei se face din procedura **BUYTICK**.

Activitățile pe care le întreprinde sînt redată în descrierea formală care urmează.

```
1 begin
2     „se incrementează numărătorul bonurilor”
3     „se adună totalul clientului la totalul zilei”
4     „totalul clientului se afișează”
5     „se imprimă bonul clientului”
6     „totalul clientului se reinițializează cu 0”
7 end
```

Notăm faptul că imprimarea va fi efectuată cu rutina **PRTTICK** : după listarea mesajului "*** VA MULȚUMIM ***", se va imprima un rînd gol, urmat de antetul bonului următor (**UNIT.NR.**, **CASA NR.**), secvență care rezultă din structura tabelii **EDBUF** (vezi fig. 13.10)

Listingul comentat al procedurii se regăsește în Cap. 17. pag. 1—31.

14.2.10. Emiterea bonului client normal : EMITERATICK

■ Cu toate diferențele de esență pe care operația de emitere a unui bon de anulare le comportă, structura acestei proceduri va fi identică cu cea a procedurii **EMITBUYTICK**.

procedură emiterat

```
1 begin
2     „se incrementează numărătorul bonurilor”
3     „se scade totalul clientului din totalul zilei”
4     „totalul clientului se afișează”
5     „se imprimă bonul de anulare”
6     „totalul clientului se reinițializează cu 0”
7 end
```

Listingul comentat al acestei rutine se găsește în Cap. 17, pag. 1—32.

14.2.11. Imprimare din EDBUF : PRTTICK

Procedura **PRTTICK** specificată funcțional în §14.1.5. și §14.1.7. va trebui să imprime conținutul anumitor linii din **EDBUF**.

■ Liniile de imprimat se vor selecta pe baza unei măști de selecție (vezi §13.2.4.).

■ Imprimarea efectivă se va face prin **PRTLIN**, care va imprima pe cele două fișii ale casei de marcat, mesajul care fusese în prealabil transpus din **EDBUF**, prin procedura **TRANLINE**.

■ Un caz aparte îl constituie PRTTICK, lansat din TOTSORT, când linia a 6-a din EDBUF ($ID_0 = 86H$) va trebui să fie imprimată de 100 de ori, după ce în prealabil zona afectată codului de sortiment fusese încărcată cu numărul de cod crescător, iar în zona de preț (vezi fig. 13.10) s-a transferat totalul aferent. Această iterație va fi rezolvată în procedura SYNTTOTS.

procedura prttick

```

1  begin
2      contor := numărul liniilor din tabela EDBUF
3      repeat
4          begin
5              if „linia curentă face parte din bonul în curs de tipărire”
6                  then if (este linia totalului)  $\wedge$  (bon de tip sin-
7                      teză)
8                      then
9                          synttots {se imprimă cele 100 de}
10                             {totaluri de sortiment}
11                          else
12                              tranline
13                              prtline {se imprimă linia curentă}
14                              „se trece la linia următoare”
15                              contor := contor - 1
16                          end
17                      until contor = 0
18          end

```

```

1  PRTTICK : LD B,NREDLNS
2           LD IX,EDBUF+2
3           LD DE,EDLLEN
4  EDLINE : LD A,(IX-1)
5           AND C
6           JR Z,NEXTEDLN
7           LD A,C
8           CP SYNTMASK
9           JR NZ,NORMLINE
10          LD A,(IX-2)
11          CP 86H
12          JR NZ,NORMLINE
13          CALL SYNTTOTS
14          JR NEXTEDLN
15  NORMLINE : CALL TRANLINE
16           CALL PRTLINE
17  NEXTEDLN : ADD IX,DE
18           DJNZ EDLINE
19           LD HL,EDBFRE
20           LD A,EDBTIME
21           CALL BEEP
22          RET

```


Adnotări :

- Decizia de imprimare sau neimprimare a unei linii din EDBUF, formulată în descriere în liniile 6, 7, 9 se regăsește în program între liniile 7-12.
- Simbolurile utilizate au următoarea semnificație :
NRDLNS — (Number of ED LiNeS) — numărul total al liniilor din EDBUF — în registrul B ($0E_H$).
EDLLEN — (EDLine LENght) — lungimea unei linii în EDBUF — în registrul DE (0011_H)
SYNTMASK — *masca de selecție* pentru liniile care vor fi imprimate în procedura SYNTHi
- Registrul index IX va indica mereu începutul util (imprimabil) al unei linii din EDBUF (vezi liniile program 2 și 17).
- IX - 1 indică ID_1 , iar IX - 2 poartă pe ID_0 (vezi fig. 13.10).
- Procedura **TRANLINE** transferă linia curentă (partea utilă) din EDBUF în PRTBUF după cum urmează :

```

TRANLINE :      PUSH      BC
                PUSH      DE
                PUSH      IX
                POP       HL
                LD         DE,PRTBUF+1
                LD         BC,EDLLEN-2
                LDIR
                POP       DE
                POP       BC
                RET
    
```

- Dacă imprimarea liniilor din EDBUF a ajuns la linia marcată prin $ID_0 = 86_H$ și masca de selecție indică SYNTH, atunci se va declanșa procedura SYNTTOTS (liniile program 13-14).

procedura synttots

```

1  begin
2      code := 0
3      cnt := 100
4      repeat
5          begin
6              „totalul de sortiment corespunzător codului curent se
              transferă în bufferul de tipărire”.
7              „se imprimă”
8              code := code + 1
9              cnt := cnt - 1
10         end
11     until cnt = 0
12 end
    
```

```

1  SYNTTOTS :      PUSH      DE
2                  PUSH      BC
3                  LD         C,0
4                  CALL      IMPLCODE
5                  LD         B,100
6  NSORTT :        PUSH      BC
7                  CALL      SORTADR
8                  LD         DE,PRTBUF+4
    
```

9	CALL	BCDCI
10	LD	HL, CODE
11	LD	DE, PRTBUF+1
12	LD	BC, SLEN
13	LDIR	
14	LD	A, ASTER
15	LD	(PRTBUF+15), A
16	CALL	PRTLINE
17	LD	B, SLEN
18	LD	HL, CODE+SLEN-1
19	CALL	EBCDINC
20	POP	BC
21	DJNZ	NSORTT
22	POP	BC
23	POP	DE
24	RET	

Anotări :

- Inițializarea celor două variabile **code** și **cnt** din liniile 2, 3 ale descrierii, se regăsesc în liniile 3—5 ale programului.
- **Suma (BCD)** corespunzătoare sortimentului curent, avînd codul **code**, se transformă în cod intern și se depune în **PRTBUF** (liniile program 7, 8, 9)
- Valoarea curentă a codului de sortiment **code**, se transferă în **PRTBUF**, la începutul liniei (liniile program 10—13).
- Linia imprimată se marchează cu "x" în dreptul cifrei celei mai puțin semnificative.
- Valoarea codului de sortiment fiind exprimată în cod intern, se incrementează prin secvența program cuprinsă între liniile 17—19.
- Procedura **EBCDINC** incrementează un număr scris în format **EBCD** indicat prin **HL**, avînd lungimea specificată în **B**.
- Acțiunea buclei principale (liniile 6—21) se reia pînă cînd **B=0** (100 de ori).

Listingul comentat al rutinelor din acest paragraf se regăsește în Cap. 17, p. 1—33—1—35.

4.2.12. Localizarea registrului de sortiment : **SORTADR**

■ Procedura **SORTADR** localizează un registru în tabela **SORTTOT** pe baza valorii din **CODE**.

Redăm descrierea formală a procedurii :

procedure sortadr

begin

„numărul **ebcd** din zona **code** se transformă în binar”

„se localizează registrul căutat, aplicînd relația 13.2., adresa de început în **hl**”

end

În Cap. 17, pag. 1—29 se regăsește listingul comentat al procedurii.

14.2.13. Actualizarea totalului de sortiment (+/-): ADDSORT/SUBSORT

■ Cele două proceduri adună (respectiv scad) prețul reprezentat în format BCD, aflat în DATBCD, la suma curentă din SORTTOT, pe baza codului de sortiment solicitat, din CODE.

Iată structura posibilă a acestor rutine :

ADDSORT :	CALL	SORTADR ; (sau SUBSORT)
	PUSH	HL
	CALL	MOVTMP
	CALL	ADDBCD ; (sau SUBBCD)
	POP	DE
	CALL	TMPMOV
	RET	

● Rutinele MOVTMP respectiv TMPMOV mută o valoare BCD (de 5 byte) "în" și "din" TMPBCD. ADDBCD și SUBBCD efectuează operațiile aritmetice "+" sau "-" asupra regiștrilor BCD, TMPBCD și DATBCD.

Aici se termină prezentarea nivelului ierarhic 1 al software-ului casei de marcat. Mai mult sau mai puțin, fiecare din rutinele prezentate au fost particulare proiectului considerat. Odată cu acest paragraf se termină și referințele făcute la specificația tehnică. Modulele care se vor prezenta în paragrafele următoare sînt fie cu caracter general cum ar fi aritmetică BCD, conversii de coduri, sau reprezintă detalii (intimități) de programare.

14.3. Aritmetica BCD cu virgulă fixă

Așa cum am prezentat în Cap. 13, la elaborarea structurilor de date, ne propunem să scriem un pachet de programe aritmetice, care să trateze numere BCD compacte (2 cifre/octet), avînd lungimea de 5 octeți, dintre care cel de-al 5-lea octet este partea zecimală. Numerele reprezentate vor fi fără semn.

Rutinele de adunare, scădere și înmulțire vor folosi regiștri TMPBCD și DATBCD, rezultatul generîndu-se în TMPBCD.

14.3.1. Adunarea a două numere : ADDBCD

Acțiunea procedurii este :

$$\text{TMPBCD} := \text{TMPBCD} + \text{DATBCD} \quad (14.2.)$$

■ Locarea octeților în regiștri BCD respectă structura :

REGBCD + 0 : octetul cel mai semnificativ

REGBCD + 4 : octetul cel mai puțin semnificativ

■ În cadrul unui octet, D_7-D_4 conține cifra cea mai semnificativă.

Dacă HL pointează pe octetul cel mai puțin semnificativ din DATBCD, iar DE pe același octet în TMPBCD și B conține lungimea numărului BCD (exprimat în octeți), atunci secvența care urmează va efectua *adunarea* celor două numere, generând rezultatul în TMPBCD.

- | | | |
|---|------------------|------------------|
| 1 | XOR | A |
| 2 | ADDBYTE : | LD A,(DE) |
| 3 | ADC | A,(HL) |
| 4 | DAA | |
| 5 | LD | (DE),A |
| 6 | DEC | DE |
| 7 | DEC | HL |
| 8 | DJNZ | ADDBYTE |

Pentru ilustrarea acțiunii acestei secvențe vom elabora o *schemă sinoptică* (fig. 14.9.), pe care vom parcurge de 2 ori bucla program cuprinsă între liniile 2—8, ilustrând evoluția regiștrilor BCD în cazul adunării numerelor **1484,25** și **347,50**.

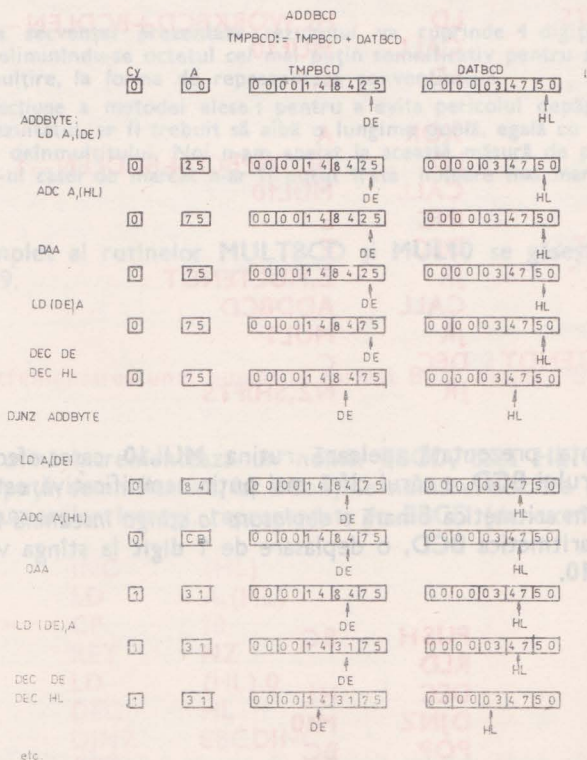


Fig. 14.9. Schema sinoptică de acțiune a rutinei ADDBCD

14.3.2. Scăderea a două numere : SUBBCD

■ Acțiunea acestei rutine va decurge în aceleași condiții ca și cea a rutinei **ADDBCD**, diferența constînd în operația aritmetică de bază executată: **SBC A,(HL)** în loc de **ADC A,(HL)**.

■ În cazul unui rezultat negativ : $C_Y = 1$

În Cap. 17, pag. 1—37 se regăsește listingul comentat al acestei rutine.

14.3.3. Înmulțirea a două numere : MULTBCD

■ Presupunem că în momentul apelului, **TMPBCD** conține înmulțitorul, iar **DATBCD** deînmulțitul. Pentru a genera rezultatul în **TMPBCD**, vom disloca înmulțitorul din **TMPBCD** într-un registru de lucru nou creat : să-l numim **WORKBCD**. După ștergerea registrului **TMPBCD** inițializăm numărătorii :

B — lungimea unui număr **BCD** (5)

C — numărul de cifre al numărului **BCD** ($2 \times 5 = 10$)

Secvența redată în continuare va efectua înmulțirea numerelor :

```
1  SHIFTS : LD HL,WORKBCD+BCDLEN-1
2          CALL MUL10
3          LD E,A
4
5          XOR A
6          LD HL,TMPBCD+BCDLEN-1
7          CALL MUL10
8          INC E
9  MULT :  DEC E
10         JR Z,MULTENDT
11         CALL ADDBCD
12         JR MULT
13  MULTENDT : DEC C
14         JR NZ,SHIFTS
```

■ Secvența prezentată apelează rutina **MUL10** care efectuează înmulțirea cu 10 a numărului **BCD**, a cărui cifră mai puțin semnificativă este pontată de **HL**.

Așa cum în aritmetica binară o deplasare la stînga înseamnă înmulțirea cu baza 2, tot așa în aritmetica **BCD**, o deplasare de 1 digit la stînga va însemna înmulțire cu baza 10.

```
MUL10 : PUSH BC
M10 :   RLD
        DEC HL
        DJNZ M10
        POP BC
        RET
```

Acesta este un exemplu clasic de utilizare a puternicei instrucțiuni Z80, RLD.

Acțiunea buclei din M10 o ilustrăm în fig. 14.10.

Revenind la rutina MULTBCD, constatăm că înmulțirea de înmulțitului cu fiecare cifră a înmulțitorului se face prin adunări succesive (liniile 9–12).

Înmulțirea nu se efectuează dacă digitul curent al înmulțitorului, care pe urma deplasării lui WORKBCD (liniile 1–2) apare în A, este 0.

La fiecare parcurgere a buclei principale, conținutul registrului rezultat TMPBCD, este deplasat cu un digit la stînga.

Pentru ilustrarea acțiunii secvenței de înmulțire am realizat schema sinoptică din fig. 14.11.

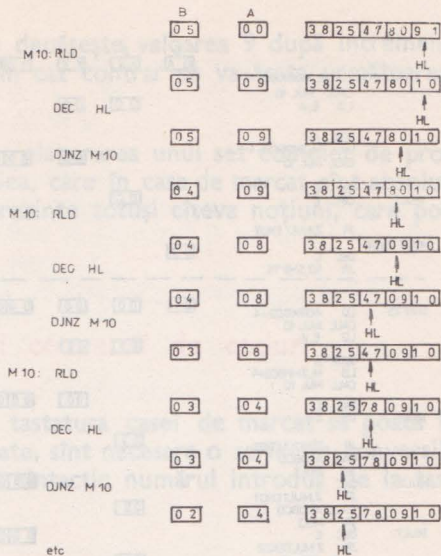


Fig. 14.10. Schema sinoptică de acțiune a rutinei MUL10

Adnotări :

- După efectuarea secvenței prezentate rezultatul va cuprinde 4 digiți zecimali. El va trebui trunchiat, eliminându-se octetul cel mai puțin semnificativ pentru a aduce numărul rezultat din înmulțire, la forma de reprezentare convențională.
- Rezultă o imperfecțiune a metodei alese : pentru a evita pericolul depășirii, registrul în care se crează rezultatul, ar fi trebuit să aibă o lungime dublă, egală cu suma lungimilor înmulțitorului și de înmulțitului. Noi n-am apelat la această măsură de protecție, fiindcă oricum software-ul casei de marcat n-ar fi putut trata numere mai mari decât formatul impus.

Listingul complet al rutinelor MULTBCD și MUL10 se găsește în Cap. 17, pag. 1–38, 1–39.

14.3.4. Incrementarea unui număr în format BCD extins : EBCDINC

■ Această rutină incrementează un număr EBCD, dacă HL specifică octetul (cifra) cea mai puțin semnificativă, iar B conține numărul de cifre ale numărului. Ea se va aplica numerelor întregi, reprezentate în EBCD (de exemplu : CODE).

```

1  EBCDINC :   INC      (HL)
2              LD      A,(HL)
3              CP      10
4              RET     NZ
5              LD      (HL),0
6              DEC     HL
7              DJNZ   EBCDINC
8              RET

```

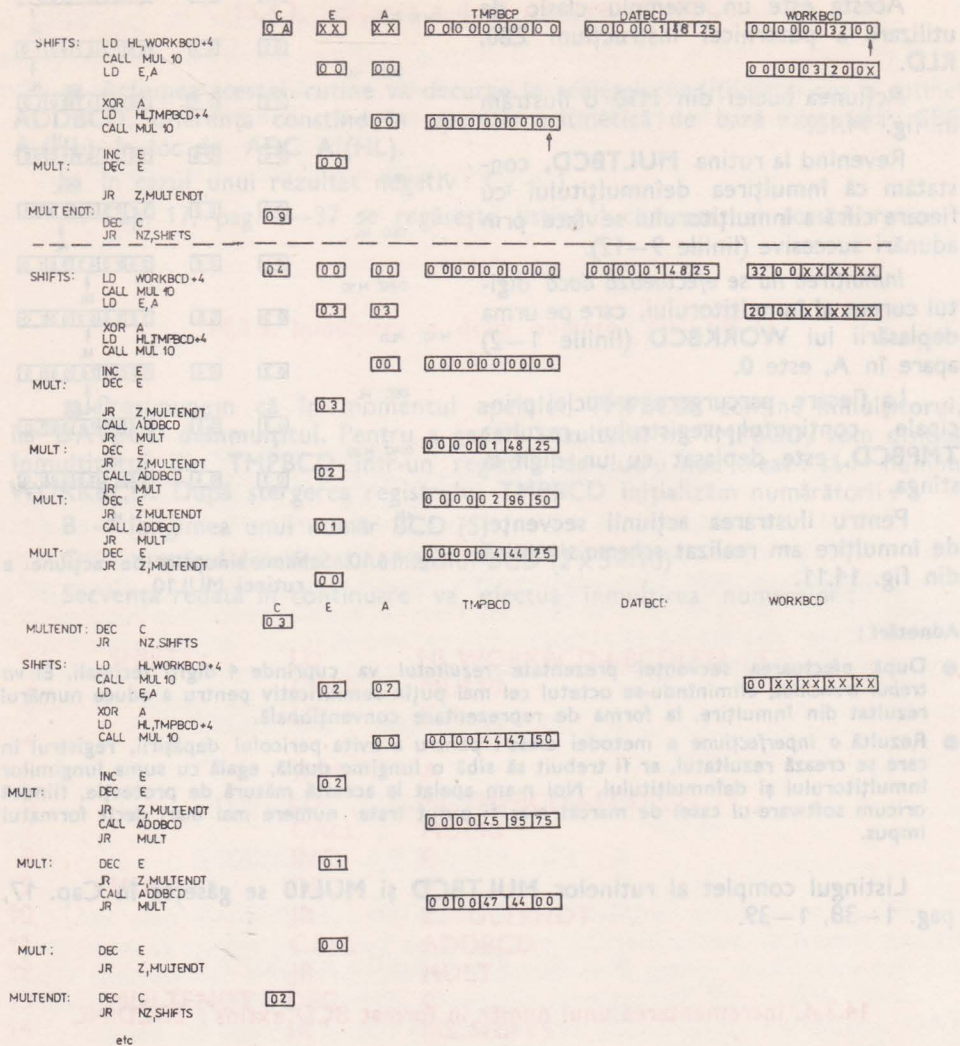



Fig. 14.11. Schema sinoptică de acțiune a rutinei MULTBCD

Dacă cifra mai puțin semnificativă nu depășește valoarea 9 după incrementare, atunci rutina se termină în linia 4. În caz contrar se va trata următoarea cifră semnificativă a numărului.

În prezentul paragraf nu ne-am propus elaborarea unui set complet de programe aritmetice BCD. Le-am tratat pe acelea, care în casa de marcat sînt absolut necesare. Sperăm ca ele să fie reușit să prezinte totuși cîteva noțiuni, care pot fi utile.

14.4. Analiză sintactică și conversii de coduri

Pentru ca prețurile introduse de la tastatura casei de marcat să poată fi prelucrate și apoi afișate, respectiv imprimate, sînt necesare o serie de conversii.

Prima operație este aceea de a valida sintactic numărul introdus de la tastatură.

14.4.1. Analiza și conversia numerelor introduse de la tastatură : SYNTAN

■ Numerele tastate nu vor putea conține mai mult decît un punct zecimal. Numărul introdus trebuie normalizat : partea zecimală va avea obligatoriu 2 cifre, indiferent că ele au fost tastate sau nu.

■ Procedura SYNTAN va efectua aceste operații asupra numărului primit în KEYBUF, depunînd rezultatul, (din care se filtrează și punctul zecimal) în zona EBCD. (Vezi definiția structurii în §13.3.1. fig. 13.12.).

Pentru exemplificarea modului în care SYNTAN normalizează numerele din KEYBUF am recurs la o schiță (vezi fig. 14.12). Se poate remarca tratarea distinctă a părții întregi și a celei zecimale.

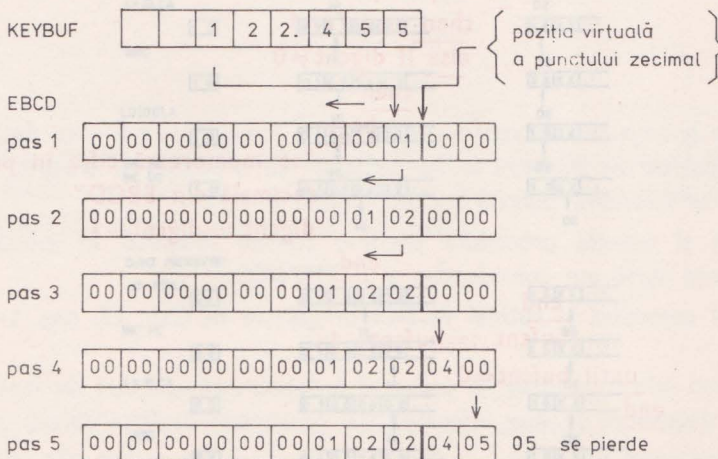


Fig. 14.12. Schema sinoptică de acțiune a rutinei SYNTAN

Prezentăm descrierea în limbaj de nivel înalt a procedurii SYNTAN.

procedură syntan

begin

„se inițializează EBCD cu "0"”

bufcnt := lungimea bufferului KEYBUF

errorf := 0

„se caută primul octet din KEYBUF diferit de spațiu”

if este cifră

then

begin

„se memorează cifra în partea întreagă din EBCD”

while (bufcnt ≠ 0) ∧ „octetul curent din KEYBUF nu conține do
punct”

begin

„se ia octetul următor din KEYBUF”

bufcnt := bufcnt - 1

„se memorează cifra în partea întreagă din EBCD”

end

end

if bufcnt ≠ 0

then

begin

digcnt := 2

repeat

begin

„se ia octetul următor din KEYBUF”

if conține punct

then errorf := 1

else if digcnt ≠ 0

then

begin

„se memorează cifra în partea

zecimale din EBCD”

digcnt := digcnt - 1

end

end

bufcnt := bufcnt - 1

until bufcnt = 0

end

end

În Cap. 17, pag. 1-42, 1-43 se găsește listingul comentat al procedurii SYNTAN.

14.4.2. Impachetarea numerelor EBCD în format BCD : EBCDBCD

■ Misiunea acestei rutine este cea de a împacheta un număr reprezentat în format **BCD extins** (o cifră/octet), situat în **EBCD**, în format **BCD compact** (2 cifre/octet) și al depune în registrul **DATBCD**.

■ Vom folosi 2 indicatori :

HL — pointează pe adresa de început a bufferului sursă : **EBCD**

DE — pointează pe adresa de început a bufferului destinație : **DATBCD**

■ Contorizarea evenimentelor se va face în registrul **B** pe care-l inițializăm la lungimea unui număr **BCD compact** (5 byte).

În aceste condiții secvența ce urmează, va efectua conversia propusă :

```

1  PACKBYTE:  LD  A,(HL)
2              INC  HL
3              RLCA
4              RLCA
5              RLCA
6              RLCA
7              RRD
8              LD  (DE),A
9              INC  HL
10             INC  DE
11             DJNZ PACKBYTE
12             RET
    
```

În centrul atenției se află instrucțiunea **RRD** (Rotate Right Digit), prezența căreia simplifică considerabil problema.

În fig. 14.13 prezentăm dinamica secvenței **PACKBYTE** din **EBCDBCD**.

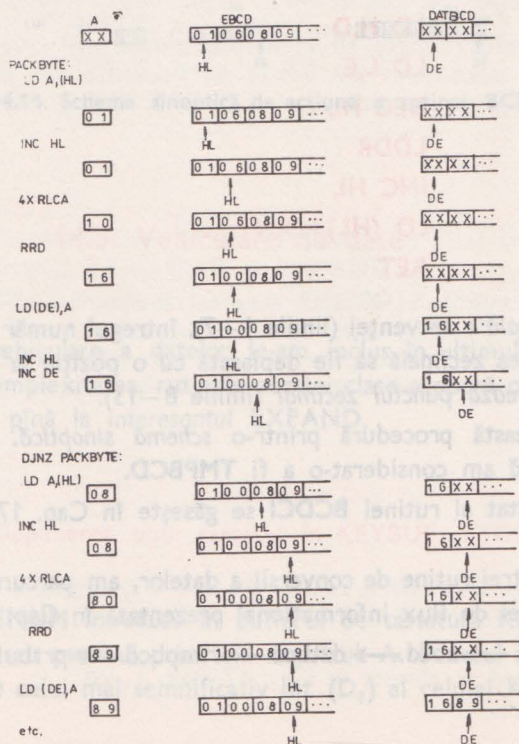


Fig. 14.13. Schema sinoptică de acțiune a rutinei **PACKBYTE**

14.4.3. Conversia inversă a numerelor : BCDCl

■ Această rutină despachetează numărul de la adresa indicată de HL în cod intern, și îl depune într-o zonă începînd cu adresa DE.

■ Ea generează și punctul zecimal, substituind zerourile ne semnificative cu spații.

Implementarea ultimei cerințe nefiind o problemă, ne vom concentra atenția asupra secvenței de conversie BCD în cod intern.

Iată secvența propusă :

(B — conține inițial lungimea unui număr BCD, adică 5)

```
1  BCDCl :      RLD
2              LD (DE),A
3              INC DE
4              XOR A
5              RRD
6              LDI
7              DJNZ BCDCl
8              LD BC,2
9              LD H,D
10             LD L,E
11             DEC HL
12             LDDR
13             INC HL
14             LD (HL),POINT
15             RET
```

În bucla principală a secvenței (liniile 1—7), întregul număr este despachetat pentru ca apoi partea zecimală să fie deplasată cu o poziție la dreapta. În golul astfel creat se inserează punctul zecimal (liniile 8—13).

Ilustrăm și această procedură printr-o schemă sinoptică. În exemplul din fig. 14.14 zona sursă am considerat-o a fi TMPBCD.

Listingul comentat al rutinei BCDCl se găsește în Cap. 17, pag. 1—46.

Prezentînd cele trei rutine de conversii a datelor, am parcurs de fapt ramura principală a diagramei de flux informațional prezentată în Cap. 13.

tastatură → keybuf → ebcd → datbcd → tmpbcd → prtbuf → imprimantă
→ afișaj

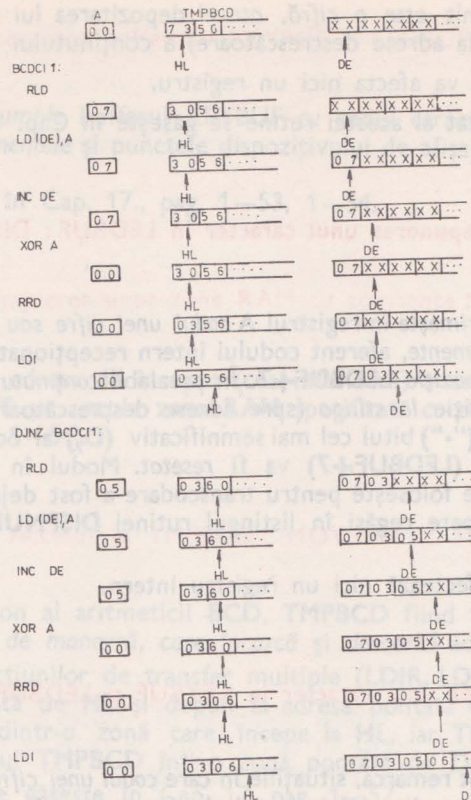


Fig. 14.14. Schema sinoptică de acțiune a rutinei BCDCI

14.5. Vehiculare de date

Rutinele de vehiculare a datelor le-am inclus în ultimul nivel ierarhic. În ceea ce privește complexitatea, rutinele acestei clase acoperă o plajă largă, de la banalul **FILLBYTES**, pînă la interesantul **EXPAND**.

14.5.1. Depunerea unui caracter în KEYBUF: STORENUM

Rutina **STORENUM** introduce în bufferul de tastatură **KEYBUF**, pe poziția **KEYBUF+7**, cifra sau punctul primit în registrul **A**. Punctul zecimal se va concretiza prin setarea celui mai semnificativ bit (**D₇**) al celei **KEYBUF+7**.

Dacă caracterul primit este o cifră, atunci depozitarea lui va fi precedată de o deplasare la stînga (la adrese descrescătoare) a conținutului întregului buffer.

STORENUM nu va afecta nici un registru:

Listingul comentat al acestei rutine se găsește în Cap. 17, pag. 1—48.

14.5.2. Depunerea unui caracter în LEDBUF : DISPNUM

Această rutină primește în registrul **A** codul unei cifre sau punct. Ea va depune codul de comandă segmente, aferent codului intern recepționat în bufferul dispozitivului de afișaj, pe poziția **LEDBUF+7**. În prealabil conținutul întregului buffer va fi deplasat cu o poziție la stînga (spre adrese descrescătoare). La recepționarea caracterului **POINT** (".") bitul cel mai semnificativ (**D₇**) al octetului din extrema dreaptă a afișajului (**LEDBUF+7**) va fi resetat. Modul în care generatorul de caractere **LEDGEN** se folosește pentru transcodare a fost deja ilustrat în §14.2.7. **CNTFUNC**, și se poate regăsi în listingul rutinei **DISPNUM** din Cap. 17, pag. 1—49.

DISPNUM nu afectează nici un registru intern.

14.5.3. Depunerea unui caracter în KEYBUF și LEDBUF : STORDISP

Așa cum ați putut remarca, situațiile în care codul unei cifre sau punct, trebuie afișat și memorat concomitent sînt destul de frecvente. De aceea am constituit rutina **STORDISP**, care apelează pe rînd **STORENUM** și **DISPNUM**.

14.5.4. Normalizarea numerelor introduse : STOREINT

Ca structură, rutina **STOREINT** se aseamănă mult cu **STORNUM**. Ea este apelată din **SYNTAN**, pentru a normaliza un număr oarecare din **KEYBUF**, aliniindu-i partea întregă în dreptul punctului zecimal virtual din **EBCD**. Listingul comentat al acestei rutine se găsește în Cap. 17 pag. 1—50.

14.5.5. Afișarea unui rezultat din PRTBUF : DISPSUM

Folosind rutina **DISPNUM**, rutina pe care o prezentăm transferă un număr din bufferul de imprimantă în **LEDBUF**. Ea va fi folosită pentru vizualizarea totalurilor clientului.

Programul sursă se găsește în Cap. 17, pag. 1—51.

14.5.6. Ștergerea bufferelor KEYBUF și LEDBUF : CLBUFDP

CLBUFDP va umple bufferul KEYBUF cu codul caracterului spațiu (20_H) și va șterge toate segmentele și punctele dispozitivului de afișaj umplând LEDBUF cu FF_H.

Ea se găsește în Cap. 17., pag. 1—53, 1—54.

14.5.7. Umplerea unor zone RAM cu constante : FILLBYTES

Primind în HL adresa de început (cea inferioară) a unei zone iar în B lungimea ei, FILLBYTES va umple zona RAM specificată cu octetul din registrul C. (vezi Cap. 17, pag. 1—54).

14.5.8. Accesul la TMPBCD : MOVTMP și TMPMOV

Registru tampon al aritmeticii BCD, TMPBCD fiind frecvent utilizat, s-au creat două module de manevră, care încarcă și descarcă acest registru RAM.

Aidoma instrucțiunilor de transfer multiple (LDIR, LDDR, etc.) care preiau de la adresa indicată de HL și depun la adresa pontată de DE, MOVTMP va încărca TMPBCD dintr-o zonă care începe la HL, iar TMPMOV va transfera conținutul registrului TMPBCD într-o zonă pontată de DE.

Listingul lor se găsește în Cap. 17, pag. 1—52.

14.5.9. Distribuirea parametrilor de stare : TRANSPAR

Nedorind să încheiem Cap. 14 cu banalități, am lăsat la urmă două rutine de manevră interesante : TRANSPAR și EXPAND.

Rutina TRANSPAR transferă câte un parametru de stare, citit prin procedura SETUP, la locul lui de destinație din EDBUF. În momentul apelării, parametrul de transferat se află în KEYBUF, lungimea cuvântului de transferat, precum și adresa de destinație regăsindu-se în tabela de distribuire SETUPTAB.

Iată rutina :

```
1  TRANSPAR :   LD   HL,SETUPTAB
2              ADD  HL,BC
3              ADD  HL,BC
4              ADD  HL,BC
5              LD   E,(HL)
6              INC  HL
7              LD   D,(HL)
8              INC  HL
9              LD   B,(HL)
```


10		EX DE,HL
11		LD DE,KEYBUF+BUFLN-1
12		
12	TAKEBYTE :	LD A,(DE)
13		DEC DE
14		BIT 7,A
15		JR Z,NOPOINT
16		LD (HL),POINT
17		DEC HL
18		RES 7,A
19		DEC B
20		JR Z,ENDTRAN
21	NOPOINT :	LD (HL),A
22		DEC HL
23		DJNZ TAKEBYTE
24	ENDTRAN :	RET
	SETUPTAB :	DW SHOPN
		DB SHOPL
		DW DESKN
		.
		.
		DW CLRKN
		DB CLRKL

Pe baza numărului de ordine primit în **BC** (valori între 0 și 3) liniile 1—4 determină adresa intrării dorite în **SETUPTAB**, de unde se citește adresa de destinație și lungimea câmpului util (liniile 6—10).

Bucla de transfer (liniile 12—23) verifică fiecare octet preluat din **KEYBUF** și dacă el conține și un punct zecimal, atunci acesta va fi inserat explicit (cod pe 1 octet) în câmpul de destinație (liniile 16—20).

14.5.10. Transferul mesajelor din EPROM în RAM : EXPAND

Sintetizăm acțiunea acestei proceduri printr-o descriere grosieră :

EXPAND

procedură expand

begin

nr1 := „numărul liniilor de expandat”

repeat

begin

„transfer din EPROM în RAM, până când se ajunge a începutul rîndului următor”

„completare cu blankuri”

nr1 :=nr1-1

end

until nr1=0

end

Aceasta fiind ultima descriere în limbaj de nivel înalt cuprinsă în cartea noastră, o vom detalia. **EXPAND**, pretîndu-se la formalizare, noua ei descriere va reflecta cît se poate de fidel structura programului rezultat în limbaj de asamblare.

Listingul sursă al rutinei **EXPAND** se găsește în Cap. 17, pag. 1—55.

procedură expand

begin

nrl := 'numărul liniilor de expandat'

"Inițializare rampointer"

"Inițializare rompointer"

char := ROM (rompointer)

repeat

begin

counter := 'lungimea unei linii'

repeat

begin

RAM (rampointer) := char

rompointer := rompointer+1

rampointer := rampointer+1

counter := counter-1

char := ROM (rompointer)

end

until char = 'inceputul rîndului următor'

repeat

begin

RAM (rampointer) := spațiu

rampointer := rampointer+1

counter := counter-1

end

until counter = 0

nrl := nrl-1

end

until nrl = 0

end

Organigrama unei posibile rutine de test, o redăm în fig. 15.1.

Implementarea algoritmului din fig. 15.1 în limbaj de asamblare nu poate constitui o problemă.

15.1. Test de EPROM

Software-ul rezident al casei de marcat este stocat într-un circuit de memorie de tip EPROM. Altfel spus, se doborîc un dvar, o oricăru bit din cel și kbyte-al capului poate avea efecte imprvizibile. Există posibilitatea ca altfel să blocheze total echipamentul, sau să-l provoace un comportament evident

MODULE SOFTWARE DEDICATE PENTRU TESTARE

Viața oricărui echipament electronic depinde în primul rând de fiabilitatea lui, iar în al doilea rând de ușurința cu care erorile de funcționare pot fi detectate și remediate. Acest din urmă deziderat nu poate fi atins dacă în faza de elaborare a proiectului, aspectului de service-abilitate nu i s-a acordat atenția cuvenită.

Prezența unui microprocesor într-un echipament poate complica activitatea de service, ea putând solicita în ultimă instanță utilizarea unor echipamente de test complexe cum ar fi, analizorul de stări logice, analizorul de semnături, sau testorul specializat pentru acel produs. Pe teren, inginerului de service, i se oferă rareori posibilitatea utilizării unor astfel de echipamente, fapt care cauzează lungirea timpului mort, cauzat de eventualele defecțiuni ale echipamentului de depanat.

Rememorăm însă faptul că același microprocesor poate veni în ajutorul personalului de exploatare și a celui de service, datorită faptului că el este capabil să execute autoteste, cu condiția ca structura hardware a echipamentului să o permită, și ca modulele program dedicate să fie elaborate și implementate.

Pornind de la considerentele de sus nu vom finaliza proiectul casei de marcat înainte de a fi elaborat module de test și/sau recomandări pentru testarea bunei funcționări a echipamentului. Prezența unui LED indicator de „prezență tensiune”, acționat pe cale software (vezi § 11.3) este un prim pas în acest sens.

„Aprinderea” acestui LED este o indicație certă nu numai a faptului că există tensiunile de alimentare cerute ci confirmă și faptul că microprocesorul „lucrează”.

Măsura în care bunele noastre intenții se vor concretiza, va reieși din materialul prezentului capitol.

15.1. Test de EPROM

Software-ul rezident al casei de marcat este stocat într-un circuit de memorie de tip EPROM. Alterarea în timp, sau datorită unor avarii, a oricărui bit din cei 4 kbyte-ai capsulei poate avea efecte inprevizibile. Există posibilitatea ca alterarea să blocheze total echipamentul, sau să-i provoace un comportament evident

anormal. Caz fericit. Dar ce ne facem dacă eroarea apare în zona rutinelor de aritmetică, și cauzează în mod aleator erori de calcul? Comentariile sînt de prisos.

Experiența demonstrează că *alterarea conținutului sau a performanțelor electrice statice sau dinamice ale unui EPROM, nu este exclusă*. De aceea ne vom lua măsuri de precauție, propunîndu-ne să testăm integritatea informației din EPROM, la fiecare inițializare a echipamentului. *In cazul constatării unei erori, casa de marcat nu va trece în starea operațională, tocmai pentru a evita malfuncționările ascunse.*

Pentru semnalizarea unei erori detectate în EPROM, am prevăzut (§11.3) un indicator luminos care se va putea activa prin bitul D_4 al portului SYSOUT.

Ne mai rămîne să stabilim metoda de testare a integrității informației din EPROM. Ideea de bază este de a genera un cuvînt de 1,2 sau 3 octeți care se calculează din conținutul EPROM-ului, după un algoritm prestabilit, și de a înscrie și această informație în celele dedicate din EPROM.

Testarea integrității conținutului EPROM se poate face executînd o rutină care va recalcula cuvîntul de control pe baza informației actuale și îl va compara cu valoarea preînregistrată în locațiile dedicate. Inegalitatea celor două cuvinte semnălează alterarea informației conținute în EPROM.

Probabilitatea ca apariția unei sau a mai multor alterări de conținut să nu modifice cuvîntul de control, și deci eroarea să se poată strecura neobservată, depinde de algoritmul ales pentru constituirea cuvîntului de control și de lungimea acestuia. Lungirea cuvîntului de control scade probabilitatea nedetecării erorilor, dar poate complica rutina de test, care la rîndul ei trebuie să rezide în EPROM scăzînd astfel gradul de eficiență al utilizării spațiului EPROM (destul de prețios de altfel). Acest conflict se rezolvă după diverse criterii, în funcție de gradul de securitate impus echipamentului.

În ceea ce privește algoritmi utilizați, numărul lor este foarte mare, oricine putîndu-și imagina o procedură în acest sens. Există cîteva tehnici consacrate din care amintim două :

- constituirea unei sume de control pe bază de adunări succesive ;
- constituirea unui cuvînt de control prin înmulțirea informației cu un polinom de control (CRC).

Această din urmă tehnică este des folosită în validarea transferului de date între calculator și periferice. Ținînd cont de faptul că echipamentul nostru este unul ipotetic, vom alege algoritmul cel mai simplu.

■ Vom constitui o sumă de control pe un octet adunînd toți octeții din EPROM (exceptînd ultima locație) și neglijînd transportul de la D_7 în sus. Vom calcula complementul față de 2 al acestui număr vuloare pe care o vom înscrie în ultima locație a memoriei EPROM. Recalculînd suma întregului EPROM, aceasta va trebui să fie 0, atîta timp cît informația cuprinsă rămîne nealterată.

Organigrama unei posibile rutine de test, o redăm în fig. 15.1.

Implementarea algoritmului din fig. 15.1 în limbaj de asamblare nu poate constitui o problemă.

EPROMT :	LD	HL,EPROMB	:inițializări
	LD	BC,EPROML	
	XOR	A	
EPROMADD :	ADD	A,(HL)	:constitulea sumei
	INC	HL	

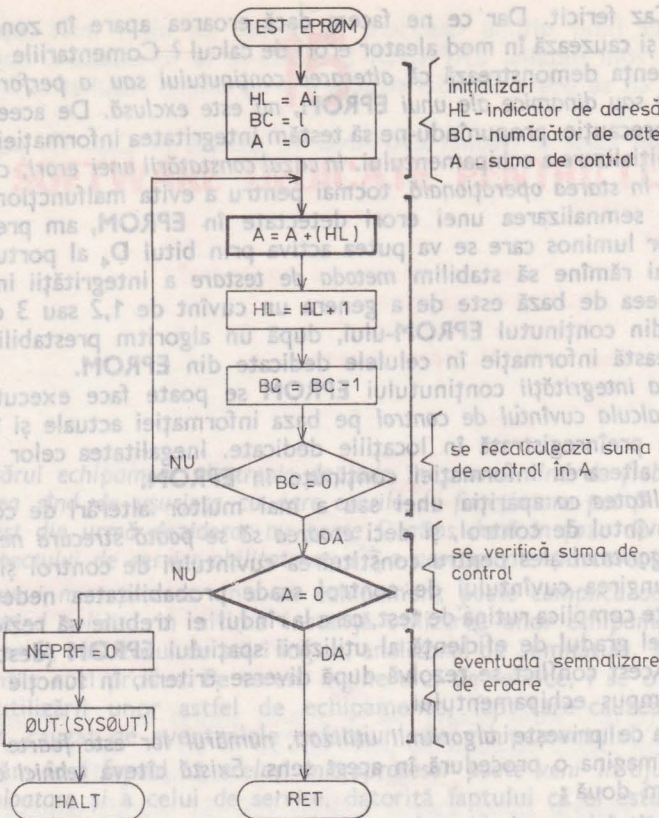


Fig. 15.1. Organigrama rutinei de test conținut EPROM

DEC	BC	
LD	D,A	;test pentru sfârșit
LD	A,B	
OR	C	
LD	A,D	
JR	NZ,EPROMADD	
OR	A	;verificarea sumei de control
JR	Z,OK	;rol
LD	A,EPERMES	;eventuală semnalizare de eroare
OUT	(SYSOUT),A	
HALT		

OK : RET

În locul instrucțiunii JR Z,OK s-ar fi putut scrie RET Z economisind astfel 2 octeți, dar s-a ales varianta de sus pentru o mai bună corelare cu organigrama din fig. 15.1.

15.2. Test de RAM

Integritatea celulelor de memorie RAM este la fel de importantă ca și cea a celulelor de EPROM, erorile cauzate de alterarea informației stocate în ele, fiind cel puțin atât de nefaste ca și cele datorate EPROM-urilor. Memoria RAM este cea structură a unui calculator, care se pretează cel mai bine la testarea „via” program. Chiar și în module de memorie constituite din sute de circuite integrate, programe de test adecvate vor putea localiza exact capsula avariată. Această facilitate se datorează faptului că informația se poate scrie și apoi reciti din celulele individuale de memorie RAM.

Literatura algoritmilor de test pentru memorie RAM este deosebit de bogată, algoritmi elaborați grupându-se în jurul a trei deziderate :

- detectarea celulelor de memorie avariate ;
- detectarea erorilor în circuitele de selecție și cele de adresare ;
- sesizarea unor erori aleatoare, rare.

În toate cele trei cazuri viteza de execuție, care în special în cazul urmării erorilor rare depinde de algoritmul utilizat, poate fi critică. Dacă spațiul de memorie de testat ajunge la sute de kocteți sau depășesc 1 Mbyte, parametrul de timp de testare devine deosebit de important.

Noi vom implementa în casa de marcat 2 teste : un test nedistructiv, pentru identificarea unor celule de memorie avariate și altul distructiv pentru verificarea corectitudinii circuitelor de adresare și selecție. În ambele cazuri distrugerea se referă la informația stocată în prealabil în RAM și nu la circuitul fizic.

15.2.1. Test de RAM nedistructiv (de conținut)

■ În testul pe care-l propunem, vom verifica și conținutul fiecărei celule de memorie în parte salvând în prealabil și restaurând după testare, conținutul original al celulei testate.

Organigrama unei astfel de rutine o redăm în fig. 15.2.

Vom transpune în limba de asamblare algoritmul prezentat în fig. 15.2.

```

RAMT1: LD      HL,RAMBOT      ; inițializări
        LD      BC,RAMLEN
RAMBYTE1: LD      A,(HL)      ; salvarea conținutului
        CPL                      ; inițial al celulei
        LD      (HL),A        ; înscrierea altei valori
        CP      (HL)          ; test de corectitudine
        JR      NZ,RAMERR
        CPL                      ; restaurarea conținutului
        LD      (HL),A        ; inițial al celulei
        INC      HL
        DEC      BC
    
```

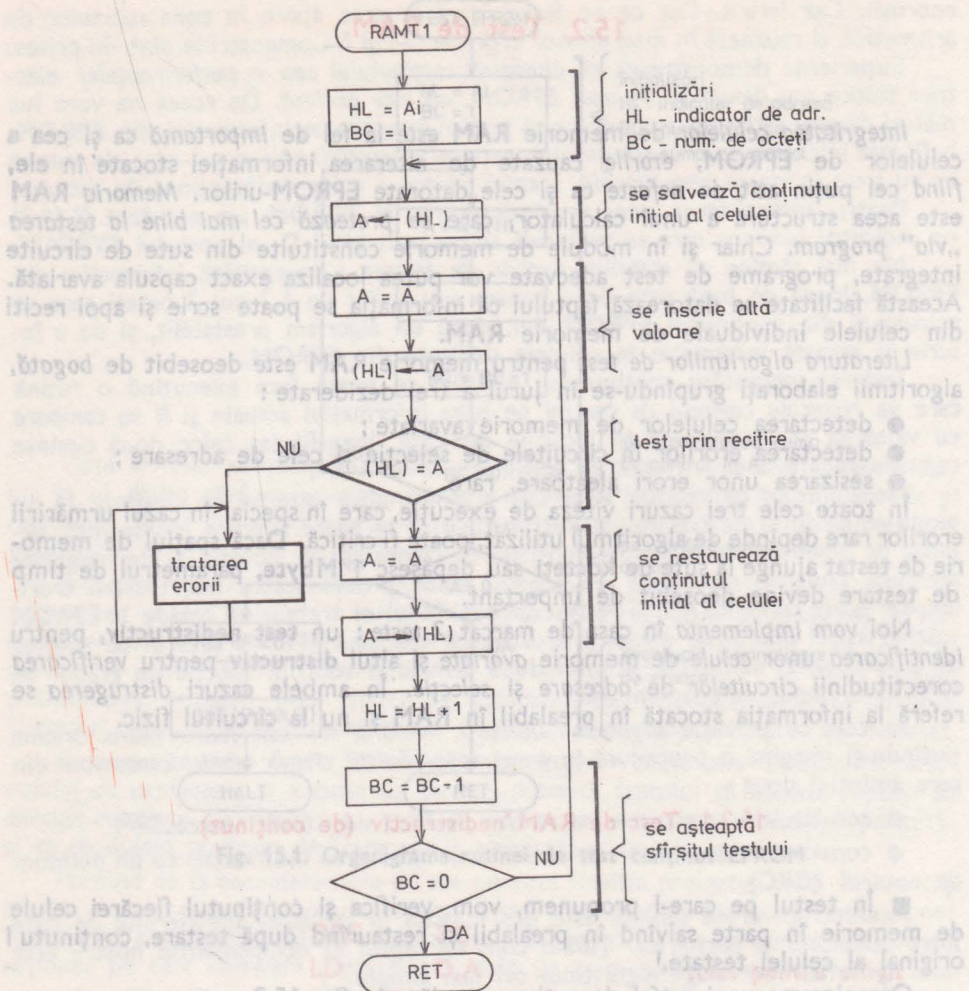



Fig. 15.2. Organigrama testului nedistructiv pentru memoria RAM

- LD ; se așteaptă terminarea tes-
- OR ; tului
- JR
- RET
- A,B
- C
- NZ,RAMBYTE1

Testul prezentat detectează erorile de conținut, în schimb nu poate constata eventualele erori de adresare. Ex. pentru două adrese diferite se selectează aceeași celulă. El are însă avantajul nedistructivității, putînd fi lansat oricînd fără a periclita integritatea datelor conținute în memoria RAM. Pentru a verifica și circuitele de adresare vom elabora cel de-al doilea test. Ați remarcat faptul că ramura de eroare (RAMERR) este deocamdată netratată. O vom face după ce vom fi elaborat cel de-al doilea test.

15.2.2. Test de RAM distructiv (de adresare)

■ Pentru a verifica o eventuală suprapunere a două sau mai multe celule, testul de RAM trebuie să decurgă în doi pași. În primul pas se înscrie întreaga memorie RAM disponibilă cu un conținut cunoscut. Valorile înscrise trebuie să difere între ele.

■ În cel de-al doilea pas se recitește întreaga zonă RAM, comparând fiecare octet cu valoarea înscrisă în primul pas. Dacă apar imperfecțiuni în circuitul de adresare, atunci ele vor putea fi detectate în cel de-al doilea pas.

■ Rămâne de stabilit algoritmul care să genereze cele n valori distincte, care urmează să fie înscrise în memorie pe durata primului pas. Noi vom folosi în acest scop însăși numărătorul de adresă, înscriind în grupuri de câte doi octeți tocmai adresa primului octet din grup.

Organigrama rutinei RAMT2 este redată în fig. 15.3.

Codificînd algoritmul, obținem următorul program

```

RAMT2 : LD      HL,RAMBOT      ; inițializări
        LD      BC,RAMLEN
RAMWR : LD      (HL),L       ; PAS 1
        INC     HL
        LD      (HL),H
        INC     HL
        DEC     BC
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,RAMWR
        LD      HL,RAMBOT    ; reinițializări
        LD      BC,RAMLEN
RAMRD : LD      A,(HL)       ; PAS 2
        CP      L
        JR      NZ,RAMERR
        INC     HL
        LD      A,(HL)
        CP      H
        JR      NZ,RAMERR
        DEC     BC
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,RAMRD
        RET
    
```

● Specificăm faptul că pentru corecta funcțiune a rutinei RAMT2, numărul de octeți de tratat (RAMLEN) trebuie să fie un număr par. Din fericire toate circuitele integrate de memorie conțin un număr par de celule.

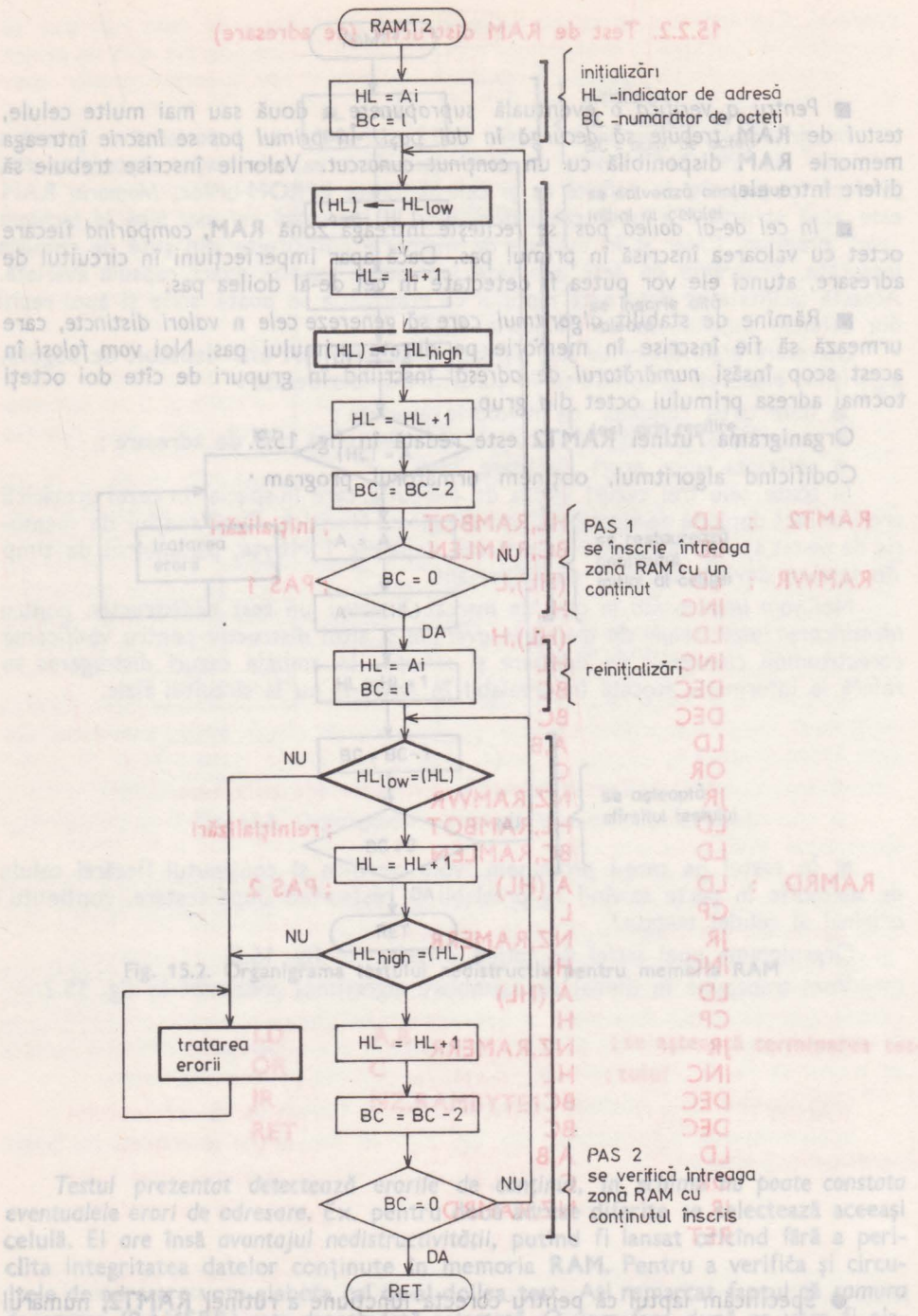


Fig. 15.3. Organigrama testului distructiv pentru memoria RAM

Tratarea erorii RAM

Ca și în cazul testului de EPROM, vom semnala evenimentul activînd un LED. În paragraful 11.3 am amintit dioda luminiscentă, care se poate activa prin bitul D_3 al portului de ieșire SYSOUT (NRAMF). *Tratarea în continuare a erorilor se va face în mod distinct de cazul precedent, în care oprirea microprocesorului (HALT) s-a dovedit a fi cea mai bună soluție, căci EPROM-ul este în cazul nostru un singur circuit, și el montat de obicei pe soclu. Personalul de service va putea înlocui cu ușurință circuitul integrat EPROM cu altul bun. Circuitele de memorie RAM sînt două și sînt și lipite. De aceea va trebui să venim în ajutorul celui care depanează, comunicîndu-i informații suplimentare pentru localizarea erorii. O soluție ar fi să afișăm pe dispozitivul de afișaj sau pe imprimantă numărul capsulei și adresa celulei defecte. Dar în cazul echipamentului studiat, ambele interfețe sînt controlate aproape în totalitate prin software, care presupune funcționarea ireproșabilă a memoriei RAM. Încercînd să afișăm sau să imprimăm ceva, riscăm ca nici una din activități să nu aibă efect, sau să genereze mesaje inegale, care vor putea crea confuzii. Soluția care ni se oferă este de a*

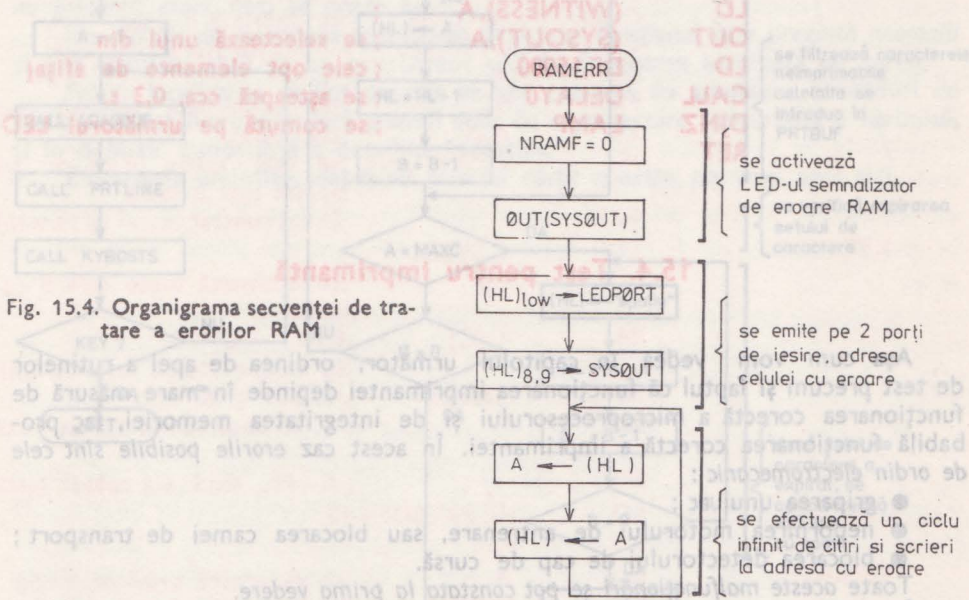


Fig. 15.4. Organigrama secvenței de tratare a erorilor RAM

emite pe cei 2 porți de ieșire adresa la care s-a constatat eroarea, iar apoi să intrăm într-un ciclu infinit de citire și scriere a acelei celule. Astfel creăm posibilitatea ca inginerul de service, să detecteze cu ajutorul unui osciloscop cauzele erorii. Organigrama acestei secvențe de tratare a erorii o redăm în fig. 15.4.

Programul scris în limbajul de asamblare, se regăsește, completat cu un semnal sonor de avertizare, în listingul din capitolul 17 pag. 1—4, 1—5.

15.3. Test pentru dispozitivul de afișaj

■ Testarea dispozitivului de afișaj nu se va putea realiza decât cu participarea operatorului, care cunoscînd ce anume va trebui să se întîmple, va verifica apariția evenimentelor preconizate. În esență testul de bună funcționare a dispozitivului de afișaj constă în a activa toate segmentele și punctele zecimale ale dispozitivului, pentru ca operatorul să le poată verifica „aprinderea”. Dacă oricare segment sau punct zecimal este lezat, riscăm să avem indicații eronate pe dispozitivul de afișaj.

lata o secvență de test posibilă :

```

LAMP : LD      B,7
        XOR     A
        OUT    (LEDPORT),A ; se activează toate segmentele
                                și punctul zecimal
LAMP : LD      A,(WITNESS)
        AND    MASK2
        OR     B
        LD     (WITNESS),A ;
        OUT   (SYSOUT),A ; se selectează unul din
        LD    DE,15000 ; cele opt elemente de afișaj
        CALL DELAY0 ; se așteaptă cca. 0,3 s.
        DJNZ LAMP ; se comută pe următorul LED
        RET
    
```

15.4. Test pentru imprimantă

Așa cum vom vedea în capitolul următor, ordinea de apel a rutinelor de test precum și faptul că funcționarea imprimantei depinde în mare măsură de funcționarea corectă a microprocesorului și de integritatea memoriei, fac probabilă funcționarea corectă a imprimantei. În acest caz erorile posibile sînt cele de ordin electromecanic :

- griparea unui ac ;
- nepornirea motorului de antrenare, sau blocarea camei de transport ;
- blocarea detectorului de cap de cursă.

Toate aceste malfunctionări se pot constata la prima vedere.

În aceste condiții prevederea unui test dedicat imprimantei poate fi redundantă. De aceea nici nu o vom include în proiectul final. Respectînd prevederile specificației funcționale F.1.0., operatorul va avea siguranța bunei funcționări a imprimantei înainte de a emite primul bon client în sesiunea respectivă de lucru.

Ca fapt divers amintim totuși că majoritatea imprimantelor de sine stătătoare sînt prevăzute cu programe de test, care listează rînduri complete conținînd setul de caractere imprimabile.

În fig. 15.5. redăm organigrama unui astfel de test pentru casa de marcat considerată.

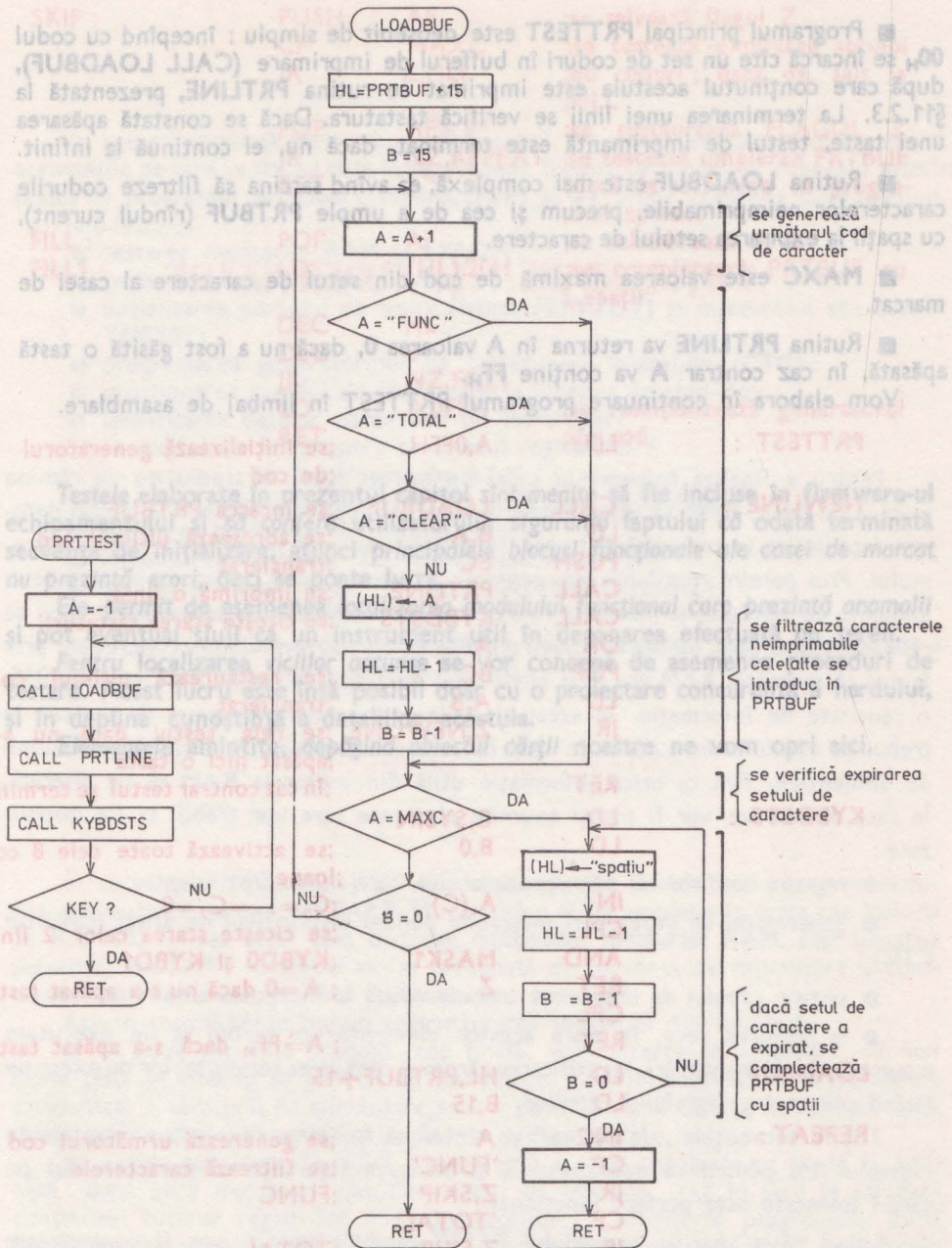


Fig. 15.5. Test pentru imprimantă : PRTTEST

■ Programul principal PRTEST este deosebit de simplu : începînd cu codul 00_H se încarcă cîte un set de coduri în bufferul de imprimare (CALL LOADBUF), după care conținutul acestuia este imprimat cu rutina PRTLINE, prezentată la §11.2.3. La terminarea unei linii se verifică tastatura. Dacă se constată apăsarea unei taste, testul de imprimantă este terminat, dacă nu, el continuă la infinit.

■ Rutina LOADBUF este mai complexă, ea avînd sarcina să filtreze codurile caracterelor neimprimabile, precum și cea de a umple PRTBUF (rîndul curent), cu spații la expirarea setului de caractere.

■ MAXC este valoarea maximă de cod din setul de caractere al casei de marcat.

■ Rutina PRTLINE va returna în A valoarea 0, dacă nu a fost găsită o tastă apăsată, în caz contrar A va conține FF_H.

Vom elabora în continuare programul PRTEST în limbaj de asamblare.

```

PRTEST :      LD      A,OFFH      ;se inițializează generatorul
                                ;de cod
NEWLINE :      CALL   LOADBUF     ;se încarcă PRTBUF
              LD      B,A        ;se adresează ultimul cod
              PUSH   BC          ;transferat
              CALL   PRTLINE     ;se imprimă o linie
              CALL   KYBDSTS     ;se citește starea tastaturii
              OR     A           ;
              POP    BC          ;se restaurează ultimul cod
              LD     A,B         ;transferat
              JR     Z,NEWLINE   ;se reia testul dacă nu s-a
                                ;apăsat nici o tastă
                                ;în caz contrar testul se termină
              RET

KYBDSTS :      LD      C,SYSIN    ;
              LD      B,0        ;se activează toate cele 8 co-
                                ;loane
              IN     A,(C)       ;C0=...=C7=0
              CPL    A           ;se citește starea celor 2 linii
              AND   MASK1       ;KYBD0 și KYBD1
              RET   Z           ;A=0 dacă nu s-a apăsat tastă
              RET   A           ;A=FFH dacă s-a apăsat tastă

LOADBUF :      LD      HL,PRTBUF+15
              LD      B,15
REPEAT :      INC     A          ;se generează următorul cod
              CP     'FUNC'     ;se filtrează caracterele
              JR     Z,SKIP     ;FUNC
              CP     'TOTAL'    ;
              JR     Z,SKIP     ;TOTAL și
              CP     'CLEAR'    ;
              JR     Z,SKIP     ;CLEAR
              LD     HL,A       ;
              DEC   HL          ;
              DEC   B           ;
  
```

SKIP :	PUSH	AF	;se salvează flagul Z
	CP	MAXC	;se testează expirarea setului
	JR	Z,FILL	;de coduri ; dacă da, salt la
			;FILL
	POP	AF	;se restaurează FLAGUL Z
	JR	NZ,REPEAT	;se testează umplerea PRTBUF
	RET		;dacă da se revine în progra-
			;mul apelant
FILL :	POP	AF	;se echilibrează stiva
FILL1 :	LD	(HL),20H	;se completează PRTBUF cu
			„spații”
	DEC	HL	
	DEC	B	
	JR	NZ,FILL1	
	LD	A,OFFH	;se reinițializează generatorul
	RET		;de cod

Testele elaborate în prezentul capitol sînt menite să fie incluse în firmware-ul echipamentului și să confere utilizatorului siguranța faptului că odată terminată secvența de inițializare, atunci principalele blocuri funcționale ale casei de marcat nu prezintă erori, deci se poate lucra.

Ele permit de asemenea localizarea modului funcțional care prezintă anomalii și pot eventual sluji ca un instrument util în depanarea efectuată pe teren.

Pentru localizarea viciilor ascunse se vor concepe de asemenea proceduri de testare. Acest lucru este însă posibil doar cu o proiectare concurentă a hardului, și în deplina cunoștință a detaliilor acestuia.

Elementele amintite, depășind obiectul cărții noastre ne vom opri aici.

INIȚIALIZARE ȘI SUPERVIZARE

Programul principal PRTTEST este conceput de simplu: începând cu codul 00, se încarcă câte un set de coduri în bufferul de imprimare (CALL LOADBUF), după care conținutul acestuia este imprimat cu rutina PRTLINE, prezentată la §11.2.3. La terminarea unei linii se verifică tastatura. Dacă se constată apăsarea unei taste, testul de imprimare este oprit, dacă nu, el continuă la infinit.

Rutina LOADBUF este mai complexă, ea având sarcina să filtreze codurile caracterelor neimprimabile și cele care umplu PRTBUF (rîndul curent), cu spații la expirarea celuiului de caractere.

Rutina PRTLINE va returna în A valoarea 0, dacă nu a fost găsită o tastă apăsată, în caz contrar A va conține valoarea...

Vom elabora în continuare programul PRTTEST în limbaj de asamblare.

Pentru a finaliza software-ul casei de marcat luată în studiu nu ne rămîne altceva de făcut, decît să elaborăm secvențele de inițializare. Vorbim la plural datorită faptului că va trebui să distingem între pornirea rece și pornirea caldă a sistemului. Prin pornire rece înțelegem cuplarea la rețea a casei, moment în care se va alimenta cu tensiune și memoria RAM. Pornirea caldă se va efectua atunci cînd se reia activitatea casei de marcat, după o pauză de curent, eveniment concretizat prin faptul că memoria RAM era deja alimentată cu tensiune, în ea fiind înscrisă deja o cantitate de informație. În acest al doilea caz, activitatea casei de marcat va trebui să continue în punctul în care ea fusese abandonată la dispariția tensiunii de alimentare, fără ca orice informație utilă din memoria RAM să fie alterată. În casa de marcat, vor fi totuși anumite elemente care vor trebui să fie inițializate:

- registri hardware ai microprocesorului, care nu au fost salvați;
- generatorul de întreruperi mascabile pentru dispozitivul de afișaj (circuitul CTC);
- poziția capului de imprimare trebuie adusă la începutul unui rînd nou;
- la pornirea rece, în afara acestor elemente vor trebui să fie executate o sumedenie de inițializări, identificarea cărora și stabilirea secvenței lor de execuție făcînd obiectul paragrafului următor.

Tot în secvențele „de trezire” va trebui să includem și testele prezentate în capitolul 15, pentru ca operatorul să poată avea siguranța că echipamentul pe care-l folosește este perfect funcțional.

Știînd că la apariția semnalului RESET (ambele porniri vor fi concretizate prin apariția acestui semnal) execuția programului începe la adresa 0, secvențele de inițializare vor trebui dispuse la începutul EPROM-ului. Cele 2 secvențe de trezire avînd astfel în mod obligatoriu o parte comună, va trebui să decidem în mod judicios momentul în care ele se vor despărți, urmîndu-și calea pentru a deveni evenimente distincte: pornire (start) sau repornire (restart).

16.1. Pornirea rece : CSTART

În această secvență pornim cu „*tabula rasa*”. Ne propunem să trecem în revistă activitățile pe care va trebui să le întreprindem, înainte de a intra în repaosul interbon (adică în programul principal : **MAINLOOP**) și a-i preda comanda operatorului. Iată-le :

- testarea memoriei **RAM** și **EPROM** ;
- testarea dispozitivului de afișaj ;
- inițializarea portului de ieșire sistem (**SYSOUT**) și martorului său (**WITNESS**) ;
- programarea generatorului de întreruperi mascabile (**CTC**) ;
- poziționarea capului de imprimare la început de rând nou ;
- specificarea datelor sistemului de întreruperi
 - modul de întrerupere și octetul registrului I
 - vectorul de întreruperi în **RAM** ;
- transferul mesajelor din **EPROM** în **EDBUF (EXPAND)** ;
- ștergerea bufferelor de intrare/ieșire ;
- inițializarea celulelor ce memorează vânzări ;
- înscrierea unei cifre „0” pe dispozitivul de afișaj ;
- validarea sistemului de întreruperi.

Astfel se constituie organigrama din fig. 16.1 care este o primă aproximație a procedurii de pornire rece a casei de marcat.

16.2. Pornirea caldă : WSTART

În secvența din Fig. 16.1 n-am marcat punctul de decizie în care se va ramifica secvența de pornire **WSTART**. El va trebui să fie amplasat în orice caz înainte de **RAMT2**, fiindcă acest test distruge conținutul memoriei **RAM**. Dar lansarea secvenței **WSTART** va trebui să fie precedată de secvența de inițializare a componentelor hardware căci ele au rămas fără tensiune.

Aceste considerente impun restructurarea secvenței din fig. 16.1.

■ În noua secvență propusă, cea finală, vom încerca să amplasăm *cît mai multe teste în amonte de punctul de ramificație*, datorită faptului că orice cădere accidentală a tensiunii de alimentare este o posibilă sursă de avarii. La repornirea casei, principalele blocuri funcționale vor trebui testate.

■ Aceste considerente ridică o problemă a cărei rezolvare nu poate fi amînată : *dacă între momentul apariției căderii de tensiune, caz în care se salvează conținutul tuturor regiștrilor microprocesorului, și relansarea programului după restaurarea regiștrilor WSTART, se vor executa alte programe, atunci integritatea memoriei RAM va fi periclitată de însași apelurile de subrutine sau salvările de regiștri, care vor scrie pe stivă, în secvența de trezire care precede punctul de ramificație CSTART, WSTART.*

■ Pentru a fi siguri că aceste manevre pe stivă nu vor afecta cu nimic conținutul **RAM**, vom rezerva o zonă de stivă dedicată amplasată în capătul memoriei, zonă care va fi folosită în exclusivitate de secvența comună de trezire. După punctul

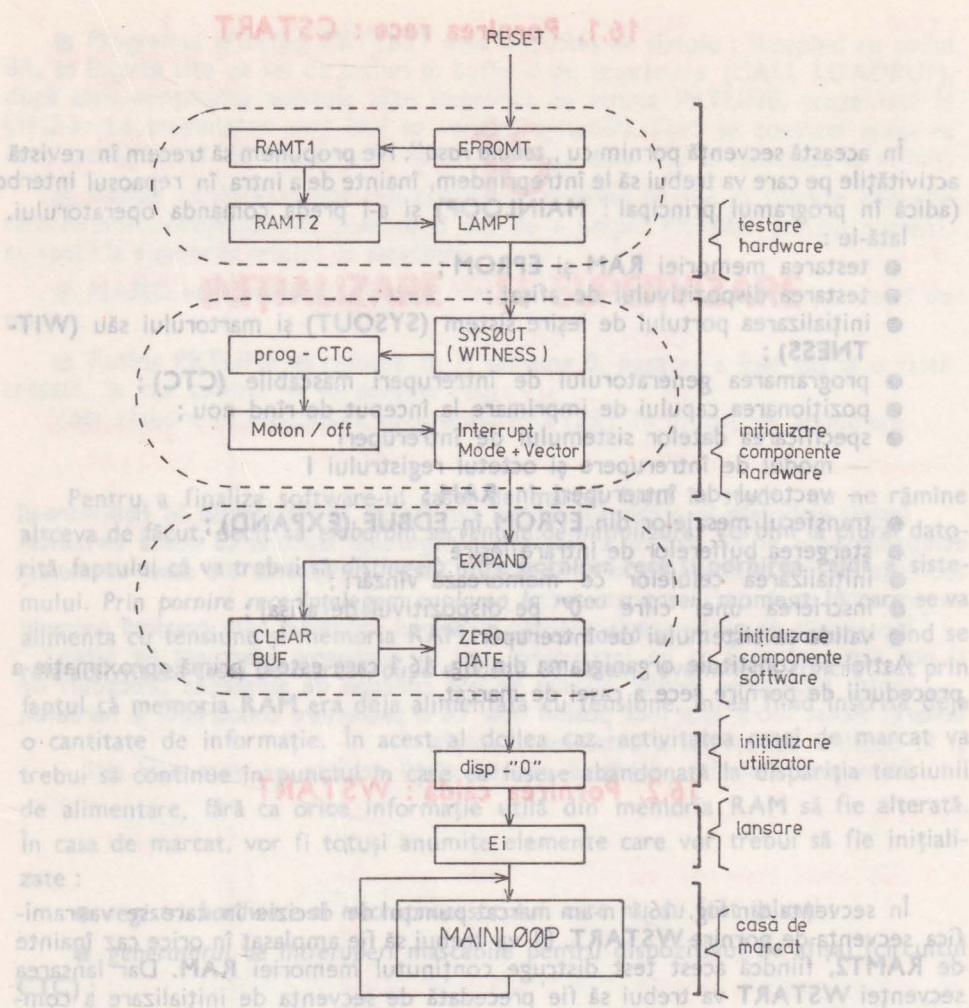


Fig. 16.1. Secvența pornirii reci (o primă aproximatie)

de ramificație, **CSTART** își va inițializa indicatorul de stivă **SP** la o adresă inferioară zonei rezervate pentru secvența comună, iar **WSTART** va prelua noua valoare pentru **SP** din celulele **RAM** în care ea fusese salvată la căderea tensiunii (a se vedea rutina **DROPOUT**). Lungimea zonei rezervate o vom stabili după definitivarea secvențelor de trezire, urmată de o analiză minuțioasă a modului în care secvența care precede punctul de ramificație va utiliza stiva.

În fig. 16.2. redăm organigrama finală a secvențelor de pornire : (a se vedea specificația funcțională **F.0.19.** și **F.1.0.**)

■ **Ramificația CSTART, WSTART** se va face pe baza comparării primei linii (9 octeți) din **EDBUF** cu primii 9 octeți ai tablei de mesaje (**MESSAGE**) din **EPROM**, așa cum am amintit deja în capitolul 13. Știm că **EDBUF** se generează executând rutina **EXPAND**, la trezirea rece a casei de marcat. Dacă pe urma com-

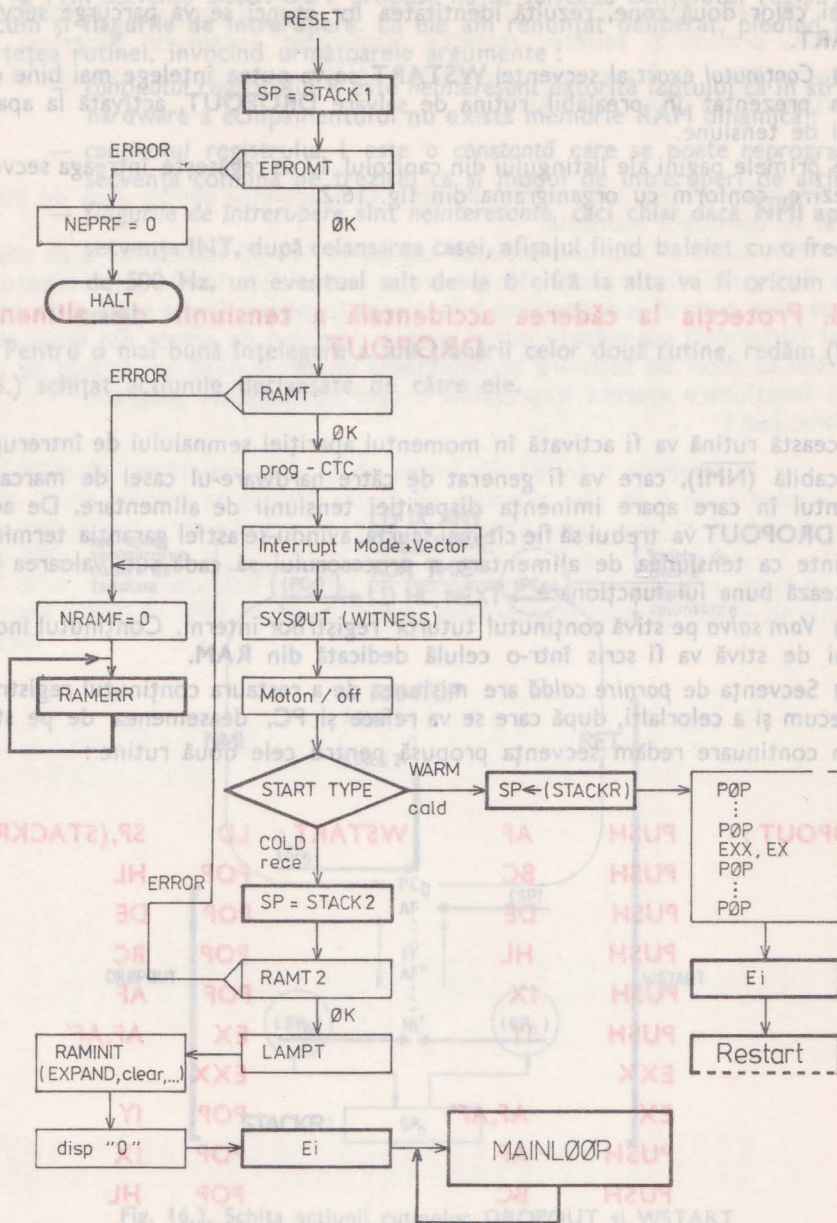


Fig. 16.2. Organigrama secvenelor de pornire (aproximația finală)

parației celor două zone, rezultă identitatea lor atunci se va parcurge secvența **WSTART**.

■ *Conținutul exact al secvenței WSTART se va putea înțelege mai bine după ce s-a prezentat în prealabil rutina de salvare DROPOUT, activată la apariția căderii de tensiune.*

Pe primele pagini ale listingului din capitolul 17 se regăsește întreaga secvență de trezire, conform cu organigrama din fig. 16.2.

16.3. Protecția la căderea accidentală a tensiunii de alimentare : DROPOUT

Această rutină va fi activată în momentul apariției semnalului de întrerupere nemascabilă (**NMI**), care va fi generat de către hardware-ul casei de marcat în momentul în care apare iminența dispariției tensiunii de alimentare. De aceea rutina **DROPOUT** va trebui să fie cât mai scurtă, avîndu-se astfel garanția terminării ei înainte ca tensiunea de alimentare a procesorului să cadă sub valoarea care garantează buna lui funcționare.

■ *Vom salva pe stivă conținutul tuturor regiștrilor interni. Conținutul indicatorului de stivă va fi scris într-o celulă dedicată din RAM.*

■ *Secvența de pornire caldă are misiunea de a restaura conținutul registrului SP precum și a celorlalți, după care se va reface și PC, deasemenea de pe stivă.*

În continuare redăm secvența propusă pentru cele două rutine :

DROPOUT :	PUSH	AF	WSTART :	LD	SP,(STACKR)
	PUSH	BC		POP	HL
	PUSH	DE		POP	DE
	PUSH	HL		POP	BC
	PUSH	IX		POP	AF
	PUSH	IY		EX	AF,AF'
	EXX			EXX	
	EX	AF,AF'		POP	IY
	PUSH	AF		POP	IX
	PUSH	BC		POP	HL
	PUSH	DE		POP	DE
	PUSH	DE		POP	BC
	PUSH	HL		POP	AF
				EI	
	LD	(STACKR),SP		RET	
	HALT				

● Prin secvența **DROPOUT** propusă nu se salvează conținutul regiștrilor **R**, precum și **flagurile de întrerupere**. La ele am renunțat deliberat, pledînd pentru scurtetea rutinei, invocînd următoarele argumente :

- conținutul regiștrului **R** este neinteresant datorită faptului că în structura hardware a echipamentului nu există memorie **RAM** dinamică ;
- conținutul regiștrului **I** este o constantă care se poate reprograma în secvența comună de trezire, ca și modul de întrerupere de altfel ;
- **flagurile de întrerupere** sînt neinteresante, căci chiar dacă **NMI** apare în secvența **INT**, după relansarea casei, afișajul fiind baleiat cu o frecvență de 500 Hz, un eventual salt de la o cifră la alta va fi oricum inesizibil.

Pentru o mai bună înțelegere a funcționării celor două rutine, redăm (în fig. 16.3.) schițat acțiunile declanșate de către ele.

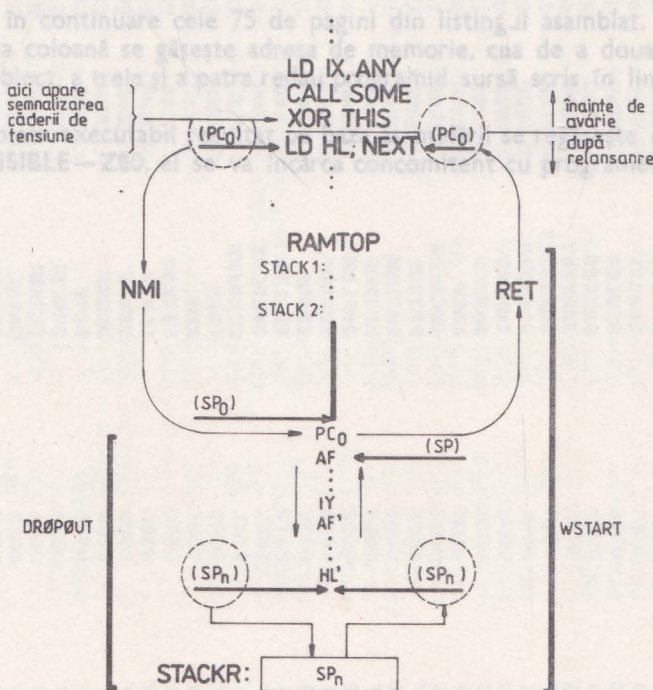


Fig. 16.3. Schița acțiunii, rutinelor **DROPOUT** și **WSTART**

● Din reprezentarea grafică sperăm să rezulte cât se poate de clar *modul în care se abortează secvența considerată ca exemplificare (după execuția instrucțiunii XOR THIS) și cum se reia „firul pierdut” începînd cu instrucțiunea LD HL, NEXT.*

LISTA COMENTATĂ A PROGRAMULUI

Redăm în continuare cele 75 de pagini din listing-ul asamblat.

În prima coloană se găsește adresa de memorie, cea de a doua coloană conține codul obiect, a treia și a patra redau programul sursă scris în limbaj de asamblare Z80.

Codul binar executabil rezultat pe baza asamblării se regăsește octet cu octet pe caseta **VISIBLE—Z80**, el se va încărca concomitent cu programele de simulare **VISZ80**.

0000	0000	MOV	R0, #0
0001	0001	MOV	R1, #0
0002	0002	MOV	R2, #0
0003	0003	MOV	R3, #0
0004	0004	MOV	R4, #0
0005	0005	MOV	R5, #0
0006	0006	MOV	R6, #0
0007	0007	MOV	R7, #0
0008	0008	MOV	R8, #0
0009	0009	MOV	R9, #0
0010	0010	MOV	R10, #0
0011	0011	MOV	R11, #0
0012	0012	MOV	R12, #0
0013	0013	MOV	R13, #0
0014	0014	MOV	R14, #0
0015	0015	MOV	R15, #0
0016	0016	MOV	R16, #0
0017	0017	MOV	R17, #0
0018	0018	MOV	R18, #0
0019	0019	MOV	R19, #0
0020	0020	MOV	R20, #0
0021	0021	MOV	R21, #0
0022	0022	MOV	R22, #0
0023	0023	MOV	R23, #0
0024	0024	MOV	R24, #0
0025	0025	MOV	R25, #0
0026	0026	MOV	R26, #0
0027	0027	MOV	R27, #0
0028	0028	MOV	R28, #0
0029	0029	MOV	R29, #0
0030	0030	MOV	R30, #0
0031	0031	MOV	R31, #0
0032	0032	MOV	R32, #0
0033	0033	MOV	R33, #0
0034	0034	MOV	R34, #0
0035	0035	MOV	R35, #0
0036	0036	MOV	R36, #0
0037	0037	MOV	R37, #0
0038	0038	MOV	R38, #0
0039	0039	MOV	R39, #0
0040	0040	MOV	R40, #0
0041	0041	MOV	R41, #0
0042	0042	MOV	R42, #0
0043	0043	MOV	R43, #0
0044	0044	MOV	R44, #0
0045	0045	MOV	R45, #0
0046	0046	MOV	R46, #0
0047	0047	MOV	R47, #0
0048	0048	MOV	R48, #0
0049	0049	MOV	R49, #0
0050	0050	MOV	R50, #0
0051	0051	MOV	R51, #0
0052	0052	MOV	R52, #0
0053	0053	MOV	R53, #0
0054	0054	MOV	R54, #0
0055	0055	MOV	R55, #0
0056	0056	MOV	R56, #0
0057	0057	MOV	R57, #0
0058	0058	MOV	R58, #0
0059	0059	MOV	R59, #0
0060	0060	MOV	R60, #0
0061	0061	MOV	R61, #0
0062	0062	MOV	R62, #0
0063	0063	MOV	R63, #0
0064	0064	MOV	R64, #0
0065	0065	MOV	R65, #0
0066	0066	MOV	R66, #0
0067	0067	MOV	R67, #0
0068	0068	MOV	R68, #0
0069	0069	MOV	R69, #0
0070	0070	MOV	R70, #0
0071	0071	MOV	R71, #0
0072	0072	MOV	R72, #0
0073	0073	MOV	R73, #0
0074	0074	MOV	R74, #0
0075	0075	MOV	R75, #0

; Initializarea variabilelor utilizate

```

6C00 RAMBOT EQU 6C00H
0400 RAMLEN EQU 400H
0005 BCDLEN EQU 5
6C00 DAYBCD EQU RAMBOT
6DF9 SORTTOT EQU DAYBCD+BCDLEN
6DFE GUESTBCD EQU SORTTOT+BCDLEN*100
6E03 WORKBCD EQU GUESTBCD+BCDLEN
6E08 TMPBCD EQU WORKBCD+BCDLEN
6E0D DATBCD EQU TMPBCD+BCDLEN
000A EBCDLEN EQU 10
6E17 PRIBUF EQU EBCD+EBCDLEN
0012 PRILEN EQU 18
6E29 KEYBUF EQU PRIBUF+PRILEN
0008 BUFLLEN EQU 8
6E31 LEDBUF EQU KEYBUF+BUFLLEN
6E39 EDBUF EQU LEDBUF+BUFLLEN
0011 EDLLEN EQU 17
000E NREDLNS EQU 14
6F27 WITNESS EQU EDBUF+EDLLEN*NREDLNS
6E57 CODE EQU EDBUF+EDLLEN*2-4
6E93 SUM EQU EDBUF+EDLLEN*5+5
6EB3 TICKNR EQU EDBUF+EDLLEN*7+3
6EBD CLRKN EQU TICKNR+10
6F01 SHOPN EQU EDBUF+EDLLEN*11-4
6F0A DATE EQU EDBUF+EDLLEN*12-4
0003 CLRNML EQU EDBUF+EDLLEN*12+5
0008 DATEL EQU 3
0002 DESKML EQU 8
0009 CONTRLEN EQU 2
6F28 STACKR EQU CONTRLEN
6FFB STACK1 EQU WITNESS+1
6FF0 STACK2 EQU 6FF8H
6FF8 INTTAB EQU STACK1-8

```



```

700F B1
7010 7A
7011 20 F7
7013 B7
7014 28 05
7016
7018 3E AF
7018 D3 90
701A 76

; se reface totalul ( indicatorul Z neafectat )
; daca BC nu este 0, salt inapoi
; se pozitioneaza indicatorul Z
; daca A = 0, salt la testul de RAM nr. 1
; daca nu,
; se trimite mesaj de eroare la portul SYSOUT
; ( NEPRF = 0 )

;
;
; Test de RAM nr. 1
;
RAMT1:
LD 21 6C00
LD 01 0400

RAMBYTE1:
LD 7E
LD 2F
CPL 77
LD 7024 BE
LD 20 0A
LD 7027 2F
LD 7028 77
LD 7029 23
LD 702A 0B
LD 702B 78
LD 702C B1
LD 20 F2
LD 702F 18 2E
LD 7031

RAMER:
LD 3E B7
LD 03 90
LD 7035 54
LD 7036 5D
LD 7037

RAMESD:
LD 21 2710
LD 3E C8
LD 703C CD 77C3
LD 703F 21 03E8
LD 7042 3E C8

C
LD A,D
NZ, EPROMADD
OR A
JR Z, RAMT1

EPMERR:
LD A, EPRERMES
OUT (SYSOUT), A
HALT

;
;
; HL, RAMBOT
; BC, RAMLEN
; A, (HL)
; se completeaza
; se rescrie si
; se testeaza daca s-a rescris corect
; daca nu, salt la eroare RAM
; HL, A
; HL, A
; NZ, RAMERR
; (HL), A
; HL
; BC
; A, B
; NZ, RAMBYTE1
; CTCPR0G

; A, RAMES
; (SYSOUT), A
; D, H
; E, L

; HL, RAMERHFR
; A, RAMERTM
; BEEP
; HL, RAMERLFR
; A, RAMERTM

```



```

MACRO-80 5.80      15-Jun-86      1-9
PAGE
EXX
EX      AF,AF
POP     IX
POP     YX
POP     HL
POP     DE
POP     BC
POP     AF
EI
RET
;
;
; Bucla principala a programului
;
;
; MAINLOOP:
CALL    INKEY
CP      WRORPT
JR      C,SIMPBUY
CP      TOTAL
JR      NZ,TESTFUNC
LD      C,O
CALL    BUYTICK
JR      MAINLOOP
TESTFUNC:
CP      FUNC
CP      Z,PROCFUNC
CALL    MAINLOOP
JR      PAGE

```

; se valideaza sistemul de intreruperi
; si se revine la adresa la care s-a intrerupt
; executia programului
; Bucla principala a programului
; se citeste o tasta de la tastatura si
; si se testeaza daca este cifra sau punct
; daca da,salt la lansarea secventei de emitere
; a unui bon client
; daca nu,se testeaza daca este TOTAL
; daca nu,urmeaza sa testam daca este FUNC
; daca da,incepe emiterea unui bon client
; C = 0,pentru a indica rutinei BUYTICK ca
; primul pret este fara cod
; dupa terminarea bonului,se revine la inceput
; se testeaza daca s-a actionat FUNC
; daca da,incepe prelucrarea ei
; dupa terminarea operatiei incepute se revine
; la inceput

Rutina PROCFUNC

```

; Este apelata din programul principal in cazul actionarii tastei FUNC.
; Numara actionarile tastei FUNC in registrul C. Aceasta numarare se termina
; la actionarea unei taste diferite de FUNC ( exceptie CLEAR ) si
; se transfera controlul rutinei selectate prin valoarea din C.
;

```

```

715B      CALL      C,DISP      ; se sterge buferul de afisaj
715C      LD        C,O         ; se initializeaza numarul actionarilor
715E      LD        B,FUNCNUM   ; si numarul maxim admis de actionari

PROCFUNC: CALL      C,DISP      ; se prelucreaza tasta FUNC
          LD        C,O         ; si se citeste o noua tasta
          LD        B,FUNCNUM   ; se testeaza daca este FUNC
          CALL      INKEY       ; daca da, se revine pentru prelucrearea ei
          CP        Z,PROCLC   ; daca nu, se sterge afisajul si
          CP        CLEAR      ; se testeaza daca este CLEAR
          JR        NZ,CASE     ; daca nu, salt la ramificare
          CALL      A           ; daca a fost CLEAR,
          CALL      DISPNUM    ; atunci se afiseaza 0 si
          RET              ; se revine in programul principal

CASE:    HL,CASETAB          ; in HL adresa tabelului cu adresele rutinelor
          DEC        C         ; se decrementeaza numarul pentru ca valo-
          LD        B,O        ; rile sa inceapa de la 0 ( pe 16 biti in BC )
          ADD       HL,BC      ; se determina adresa la care se gaseste
          ADD       HL,BC      ; adresa rutinei de apelat
          LD        E,(HL)     ; se transfera in E si D adresa rutinei,
          INC        HL        ;
          LD        D,(HL)     ;
          EX        DE,HL     ; iar din DE in HL
          INC        C         ; se reface valoarea initiala a numaratorului
          INC        C         ; si salt la rutina selectata
          JP        (HL)

```

```

11B4 MACRO-80 5.80 15-Jun-86
11B5
11B6 CB 11C3
11B7 SE 14
11B8 ST 08B3
11B9 IS
11BA SE
11BB
11BC
11BD
11BE
11BF ST 1005A0
11C0
11C1
11C2
11C3
11C4
11C5
11C6
11C7
11C8
11C9
11CA
11CB
11CC
11CD
11CE
11CF
11D0
11D1
11D2
11D3
11D4
11D5
11D6
11D7
11D8
11D9
11DA
11DB
11DC
11DD
11DE
11DF
11E0
11E1
11E2
11E3
11E4
11E5
11E6
11E7
11E8
11E9
11EA
11EB
11EC
11ED
11EE
11EF
11F0
11F1
11F2
11F3
11F4
11F5
11F6
11F7
11F8
11F9
11FA
11FB
11FC
11FD
11FE
11FF
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

```


Rutina CNTFUNC

; ;
 ; ; Prelucreeaza o actionare a tastei FUNC (incrementare numarator, afisare,
 ; ; sonerie).
 ; ; Distruge: C,D,E,H,L
 ; ; Registrii utilizati:
 ; ; C - numaratorul actionarilor tastei FUNC
 ; ; DE - pentru adresarea buferului de afisaj LEDBUF din C.
 ; ; HL - pentru adresarea generatorului de cifre pe 7 segmente
 ; ; B - numarul maxim admis de actionari

```

CNTFUNC:  PUSH  AF      ; se salveaza A
          INC   C       ; se incrementeaza numaratorul actionarilor
          LD   A,B     ; se transfera numarul maxim de actionari
          CP   C       ; admis in A
          JR   NC,DISPNR ; se testeaza daca numaratorul a depasit
          LD   C,I     ; aceasta valoare
          CALL CLDISP  ; daca nu, salt la afisarea numaratorului
          LD   B,0     ; daca da, se reinitalizeaza numaratorul
          LD   DE,LEDBUF+0 ; se sterge buferul de afisaj
          LD   HL,LEDGEN ; se determina adresa codului corespunzator
          ADD  HL,BC   ; valorii numaratorului
          LD   B,A     ; se reface in B valoarea maxima admisa
          LD   A,(HL)  ; se reface in A valoarea maxima admisa
          LD   DE,A    ; codul selectat se transfera
          LD   HL,FCSDFRE ; in buferul de afisaj
          LD   A,FCSDFRE ; si se suna soneria
          CALL BEEP   ; se reface A
          POP  AF
          RET
    
```

```

3505 5R
3506 11
3507
3508 MACRO-80 5.80
3509 15-Jun-86
3510 PAGE 1-14 ENDDIWR
3511 DEC B
3512 SEC 3'V
3513 DEC MF
3514
3515 Rutina SETUP
3516
3517 ;
3518 ;
3519 ;
3520 ; Este rutina prin care se introduc in EDBUF informatii despre casa de marcat
3521 ; ( numarul unitatii comerciale,numarul casei,data si numarul casierului ).
3522 ;
3523 ;
3524 ;
3525 ;
3526 ;
3527 ;
3528 ;
3529 ;
3530 ;
3531 ;
3532 ;
3533 ;
3534 ;
3535 ;
3536 ;
3537 ;
3538 ;
3539 ;
3540 ;
3541 ;
3542 ;
3543 ;
3544 ;
3545 ;
3546 ;
3547 ;
3548 ;
3549 ;
3550 ;
3551 ;
3552 ;
3553 ;
3554 ;
3555 ;
3556 ;
3557 ;
3558 ;
3559 ;
3560 ;
3561 ;
3562 ;
3563 ;
3564 ;
3565 ;
3566 ;
3567 ;
3568 ;
3569 ;
3570 ;
3571 ;
3572 ;
3573 ;
3574 ;
3575 ;
3576 ;
3577 ;
3578 ;
3579 ;
3580 ;
3581 ;
3582 ;
3583 ;
3584 ;
3585 ;
3586 ;
3587 ;
3588 ;
3589 ;
3590 ;
3591 ;
3592 ;
3593 ;
3594 ;
3595 ;
3596 ;
3597 ;
3598 ;
3599 ;
3600 ;
3601 ;
3602 ;
3603 ;
3604 ;
3605 ;
3606 ;
3607 ;
3608 ;
3609 ;
3610 ;
3611 ;
3612 ;
3613 ;
3614 ;
3615 ;
3616 ;
3617 ;
3618 ;
3619 ;
3620 ;
3621 ;
3622 ;
3623 ;
3624 ;
3625 ;
3626 ;
3627 ;
3628 ;
3629 ;
3630 ;
3631 ;
3632 ;
3633 ;
3634 ;
3635 ;
3636 ;
3637 ;
3638 ;
3639 ;
3640 ;
3641 ;
3642 ;
3643 ;
3644 ;
3645 ;
3646 ;
3647 ;
3648 ;
3649 ;
3650 ;
3651 ;
3652 ;
3653 ;
3654 ;
3655 ;
3656 ;
3657 ;
3658 ;
3659 ;
3660 ;
3661 ;
3662 ;
3663 ;
3664 ;
3665 ;
3666 ;
3667 ;
3668 ;
3669 ;
3670 ;
3671 ;
3672 ;
3673 ;
3674 ;
3675 ;
3676 ;
3677 ;
3678 ;
3679 ;
3680 ;
3681 ;
3682 ;
3683 ;
3684 ;
3685 ;
3686 ;
3687 ;
3688 ;
3689 ;
3690 ;
3691 ;
3692 ;
3693 ;
3694 ;
3695 ;
3696 ;
3697 ;
3698 ;
3699 ;
3700 ;
3701 ;
3702 ;
3703 ;
3704 ;
3705 ;
3706 ;
3707 ;
3708 ;
3709 ;
3710 ;
3711 ;
3712 ;
3713 ;
3714 ;
3715 ;
3716 ;
3717 ;
3718 ;
3719 ;
3720 ;
3721 ;
3722 ;
3723 ;
3724 ;
3725 ;
3726 ;
3727 ;
3728 ;
3729 ;
3730 ;
3731 ;
3732 ;
3733 ;
3734 ;
3735 ;
3736 ;
3737 ;
3738 ;
3739 ;
3740 ;
3741 ;
3742 ;
3743 ;
3744 ;
3745 ;
3746 ;
3747 ;
3748 ;
3749 ;
3750 ;
3751 ;
3752 ;
3753 ;
3754 ;
3755 ;
3756 ;
3757 ;
3758 ;
3759 ;
3760 ;
3761 ;
3762 ;
3763 ;
3764 ;
3765 ;
3766 ;
3767 ;
3768 ;
3769 ;
3770 ;
3771 ;
3772 ;
3773 ;
3774 ;
3775 ;
3776 ;
3777 ;
3778 ;
3779 ;
3780 ;
3781 ;
3782 ;
3783 ;
3784 ;
3785 ;
3786 ;
3787 ;
3788 ;
3789 ;
3790 ;
3791 ;
3792 ;
3793 ;
3794 ;
3795 ;
3796 ;
3797 ;
3798 ;
3799 ;
3800 ;
3801 ;
3802 ;
3803 ;
3804 ;
3805 ;
3806 ;
3807 ;
3808 ;
3809 ;
3810 ;
3811 ;
3812 ;
3813 ;
3814 ;
3815 ;
3816 ;
3817 ;
3818 ;
3819 ;
3820 ;
3821 ;
3822 ;
3823 ;
3824 ;
3825 ;
3826 ;
3827 ;
3828 ;
3829 ;
3830 ;
3831 ;
3832 ;
3833 ;
3834 ;
3835 ;
3836 ;
3837 ;
3838 ;
3839 ;
3840 ;
3841 ;
3842 ;
3843 ;
3844 ;
3845 ;
3846 ;
3847 ;
3848 ;
3849 ;
3850 ;
3851 ;
3852 ;
3853 ;
3854 ;
3855 ;
3856 ;
3857 ;
3858 ;
3859 ;
3860 ;
3861 ;
3862 ;
3863 ;
3864 ;
3865 ;
3866 ;
3867 ;
3868 ;
3869 ;
3870 ;
3871 ;
3872 ;
3873 ;
3874 ;
3875 ;
3876 ;
3877 ;
3878 ;
3879 ;
3880 ;
3881 ;
3882 ;
3883 ;
3884 ;
3885 ;
3886 ;
3887 ;
3888 ;
3889 ;
3890 ;
3891 ;
3892 ;
3893 ;
3894 ;
3895 ;
3896 ;
3897 ;
3898 ;
3899 ;
3900 ;
3901 ;
3902 ;
3903 ;
3904 ;
3905 ;
3906 ;
3907 ;
3908 ;
3909 ;
3910 ;
3911 ;
3912 ;
3913 ;
3914 ;
3915 ;
3916 ;
3917 ;
3918 ;
3919 ;
3920 ;
3921 ;
3922 ;
3923 ;
3924 ;
3925 ;
3926 ;
3927 ;
3928 ;
3929 ;
3930 ;
3931 ;
3932 ;
3933 ;
3934 ;
3935 ;
3936 ;
3937 ;
3938 ;
3939 ;
3940 ;
3941 ;
3942 ;
3943 ;
3944 ;
3945 ;
3946 ;
3947 ;
3948 ;
3949 ;
3950 ;
3951 ;
3952 ;
3953 ;
3954 ;
3955 ;
3956 ;
3957 ;
3958 ;
3959 ;
3960 ;
3961 ;
3962 ;
3963 ;
3964 ;
3965 ;
3966 ;
3967 ;
3968 ;
3969 ;
3970 ;
3971 ;
3972 ;
3973 ;
3974 ;
3975 ;
3976 ;
3977 ;
3978 ;
3979 ;
3980 ;
3981 ;
3982 ;
3983 ;
3984 ;
3985 ;
3986 ;
3987 ;
3988 ;
3989 ;
3990 ;
3991 ;
3992 ;
3993 ;
3994 ;
3995 ;
3996 ;
3997 ;
3998 ;
3999 ;
4000 ;

```

17. PROGRAM COMENTAT

MACRO-80 5.80 15-Jun-86

PAGE 1-15

```

;
; Rutina TRANSPAR
;
; Apelata din SETUP, transfera parametrul introdusi la adresele lor, adrese
; determinate de valoarea numaratorului de programare.
; Transferul incepe cu ultima cifra introdusa, deci tabela SETUPTAB contine
; adresele de sfirsit ale zonelor.
; Distruge: A,D,E,H,L ( B = 0 si la inceput, si la sfirsit )
; Registrul utilizati:
; HL- pentru adresarea tabelii SETUPTAB si a zonei parametrului
; DE- la formarea adresei zonei parametrului si pentru adresarea
; buferului KEYBUF
; B - la calcularea adresei din SETUPTAB si ca numarator al octetilor
; parametrului
; A - la diferite operatii
;
; TRANSPAR:
LD HL, SETUPTAB ; in HL adresa tabelii de adrese si lungimi
ADD HL, BC ; se calculeaza adresa corespunzatoare
ADD HL, BC ; valorii numaratorului
LD E, (HL) ; se transfera adresa parametrului din
INC HL ; tabela SETUPTAB in DE
INC HL D, (HL) ; iar lungimea in octeti a parametrului
INC HL B, (HL) ; in B
LD EX, DE, HL ; adresa din DE in HL
LD LD, DE, KEYBUF+BUFLen-1 ; in DE adresa buferului KEYBUF
;
; TAKEBYTE:
LD A, (DE) ; se ia octetul curent din KEYBUF
DEC DE ; si se testeaza daca contine punct
BIT 7, A ; daca nu, salt
JR Z, NOPPOINT ; daca da, se introduce punctul
LD (HL), POINT
DEC HL ; se reseteaza bitul indicator de punct
DEC B ; se decrementeaza numaratorul de octeti
DEC B ; daca s-a ajuns la 0, salt la sfirsit
JR Z, ENDTRAN ; daca nu, atunci
; se transfera cifra
LD (HL), A
DEC HL
NOPPOINT:

```


Rutina TOTSORT

; Imprima si afiseaza totalul corespunzator sortimentului selectat.Nu se
 ; executa numai in prezenta cheii de control.La apel in A codul ultimei taste
 ; actionate,KEYBUF si LEDBUF sterse.Daca se opereaza incorect,se suna soneria
 ; si se afiseaza 0.

```

TOTSORT:
    CD 7370      ; se citește codul si inca o tasta
    FE 0E      ; este aceasta tasta TOTAL ?
    20 2A      ; daca nu,eroare
    DB 90      ; se citește portul intrarilor
    CB 4F      ; se testeaza prezenta cheii de control
    28 24      ; daca nu este prezenta,eroare
    CD 740F      ; se formeaza in HL adresa totalului cautat,
    11 6E1B      ; totalul se despacheteaza si se introduce
    CD 7666      ; in buferul PRIBUF,
    CD 76F3      ; de unde se afiseaza
    06 04      ; se incrementeaza numarul de bon
    21 6EB6      HL,TICKNR+TCKNRLEN-1
    CD 75E7      ; totalul se introduce in SUM
    CD 740F      ;
    11 6E93      ;
    CD 7666      ;
    OE 04      ; in C masca de selectie a liniilor din EDBUF
    CD 74E6      ; si se imprima bonul
    18 0C      ;
    ERRTSORT:
    LD 03EB      ; in caz de eroare,se suna soneria
    3E 14      ;
    CALL 77C3      ;
    XOR A        ;
    CALL 76B3      ; si se afiseaza 0
    ENDSORT:
    RET
    PAGE
    
```


MACR0-80 5.80 15-Jun-86 PAGE 1-19
 CS
 1581
 1581 CD 1603
 1582 VL
 1583 CD 1AC3
 1584 3E 14
 1585 S1 03EB
 1586 08 0C

FE OE 0605
 20 2A 04
 2084
 2085
 2086 DB 90 000
 2087 CB 4F 0A3
 2088 28 24 00E
 2089 21 6C00
 2090 11 6E1B
 2091 CD 7666
 2092 CD 76F3
 2093 06 04
 2094 21 6EB6
 2095 CD 75E7
 2096 21 6C00
 2097 11 6E93
 2098 CD 7666
 2099 OE 02
 2100 CD 74E6
 2101 18 0C
 2102
 2103 21 03E8
 2104 3E 14
 2105 CD 77C3
 2106 AF
 2107 CD 76B3
 2108
 2109 CD 7683
 2110 C9

EBD120N1: CVET DIZBDM
 Rutina TOTDAY
 ;
 ;
 ; Imprima si afiseaza totalul zilei. Nu se executa decit in prezenta cheii de
 ; control. La apeli in A codul ultimei taste actionate, iar KEYBUF si LEDBUF
 ; sterse. In caz de operare gresita se suna soneria si 0 pe afisaj.
 TOTDAY:
 CP VIT ; dupa ultimul FUNC s-a actionat TOTAL ?
 JR D ; daca nu, eroare
 IN VIT A, (SYSIN) ; se citeste portul de intrari
 BIT KEY1, A ; si se testeaza prezenta cheii de control
 JR VIT Z, ERRTDAY ; daca nu este prezenta, eroare
 LD HL, DAYBCD ; daca da, se despacheteaza totalul zilei si se
 DE, PRIBUF+4 ; introduce in buferul de tiparire PRIBUF,
 CALL BCDCI ;
 CALL DISPSUM ; de unde se afiseaza
 LD B, TCKNRLEN ; se incrementeaza numarul bonului
 LD HL, TCKNR+TCKNRLEN-1 ;
 CALL EBCDINC ;
 LD HL, DAYBCD ; se introduce totalul in EDBUF
 LD DE, SUM ;
 CALL BCDCI ;
 LD C, DAYMASK ; in C masca de selectie
 CALL PRITICK ; se tipareste bonul
 JR ENTDAY ;
 ERRTDAY:
 LD HL, OPERFR ; in caz de eroare de operare, se suna soneria
 LD HL, A, OPERTM ;
 CALL BEEP ;
 LD HL, A, OPERFR ; si se afiseaza 0
 CALL DISPNUM ;
 ENTDAY: RET
 PAGE


```

;
;
;
; Rutina INPRICE
;
; Citeste un pret ( cu sau fara cod ) de la tastatura.Primeste de la rutina
; apelanta codul ultimei taste actionate in A si numarul de actionari ale
; tastei FUNC in C.
; Introducerea pretului se termina la actionarea tastei "+".
; La terminarea rutinei pretul introdus se gaseste in buferele
; KEYBUF si LEDBUF,iar codul ( 99 in cazul preturilor fara cod explicit )
; in buferul CODE.
INPRICE:
LD D,A
LD A,C
OR A,C
LD A,D
JR Z,CODELESS
; se salveaza codul ultimei taste actionate
; si se testeaza daca inaintea ei s-a apasat
; FUNC ( se pozitioneaza indicatorul Z )
; se reface codul ultimei taste ( operatie ce
; nu afecteaza indicatorul Z )
; daca FUNC nu a fost actionat,atunci salt la
; pret fara cod ( implicit 99 )
; daca FUNC a fost actionat,atunci este pret cu
; cod,deci se citeste codul si inca o tasta
; dupa ea si se testeaza daca aceasta tasta
; este numar sau punct
; daca da,salt la prelucrarea ei
; in caz contrar se citeste o noua tasta si
; salt inapoi la testare
CODED: CALL INCODE
AFTERCOD: CP MRORPT
; KEABUF JR C,PROCDIG
; KEALBUF JR INKEY
; KEALBUF JR AFTERCOD
CODELESS:
LD C,9
CALL IMPLCODE
LD HL,NRFUNC
LD (HL),1
PROCDIG: CALL STORDISP
NEXTKEY: CALL INKEY
CP
CALL Z,CLBUFDP
CP
FE 0B
FE 0F
CD 7771
18 F7
OE 09
CD 739B
21 6F2A
36 01
CD 76D9
72FF
7302
7305
FE 0F
CC 771F
FE 0D

```


MACRO-80 3.80 15-Jun-86 PAGE 1-21
 MACRO-80 5.80 15-Jun-86 PAGE 1-23

```

1357      CD
1358      CD 13VB
1359      CR OC
1360      SI VE3P
1361      CD 13V
1362      CD 1343
1363      CD 75F2
1364      20 26
1365      CD 7643
1366      21 6E26
1367      36 10
1368      CD 7449
1369
1370      38 05
1371      CD 73AB
1372      18 14
1373      21 6DF9
1374      11 6E1B
1375      CD 7666
1376      CD 76F3
1377      21 07D0
1378      3E 14
1379      CD 77C3
1380      C9
1381      777A
1382      YE OF
1383      CC 771F
1384      FE ON
1385
1386      CD 13VB
1387      CR OC
1388      SI VE3P
1389      CD 13V
1390      CD 1343
1391      CD 75F2
1392      20 26
1393      CD 7643
1394      21 6E26
1395      36 10
1396      CD 7449
1397
1398      38 05
1399      CD 73AB
1400      18 14
1401      21 6DF9
1402      11 6E1B
1403      CD 7666
1404      CD 76F3
1405      21 07D0
1406      3E 14
1407      CD 77C3
1408      C9
1409      777A
1410      YE OF
1411      CC 771F
1412      FE ON
1413
1414      CD 13VB
1415      CR OC
1416      SI VE3P
1417      CD 13V
1418      CD 1343
1419      CD 75F2
1420      20 26
1421      CD 7643
1422      21 6E26
1423      36 10
1424      CD 7449
1425
1426      38 05
1427      CD 73AB
1428      18 14
1429      21 6DF9
1430      11 6E1B
1431      CD 7666
1432      CD 76F3
1433      21 07D0
1434      3E 14
1435      CD 77C3
1436      C9
1437      777A
1438      YE OF
1439      CC 771F
1440      FE ON
1441
1442      CD 13VB
1443      CR OC
1444      SI VE3P
1445      CD 13V
1446      CD 1343
1447      CD 75F2
1448      20 26
1449      CD 7643
1450      21 6E26
1451      36 10
1452      CD 7449
1453
1454      38 05
1455      CD 73AB
1456      18 14
1457      21 6DF9
1458      11 6E1B
1459      CD 7666
1460      CD 76F3
1461      21 07D0
1462      3E 14
1463      CD 77C3
1464      C9
1465      777A
1466      YE OF
1467      CC 771F
1468      FE ON
1469
1470      CD 13VB
1471      CR OC
1472      SI VE3P
1473      CD 13V
1474      CD 1343
1475      CD 75F2
1476      20 26
1477      CD 7643
1478      21 6E26
1479      36 10
1480      CD 7449
1481
1482      38 05
1483      CD 73AB
1484      18 14
1485      21 6DF9
1486      11 6E1B
1487      CD 7666
1488      CD 76F3
1489      21 07D0
1490      3E 14
1491      CD 77C3
1492      C9
1493      777A
1494      YE OF
1495      CC 771F
1496      FE ON
1497
1498      CD 13VB
1499      CR OC
1500      SI VE3P
1501      CD 13V
1502      CD 1343
1503      CD 75F2
1504      20 26
1505      CD 7643
1506      21 6E26
1507      36 10
1508      CD 7449
1509
1510      38 05
1511      CD 73AB
1512      18 14
1513      21 6DF9
1514      11 6E1B
1515      CD 7666
1516      CD 76F3
1517      21 07D0
1518      3E 14
1519      CD 77C3
1520      C9
1521      777A
1522      YE OF
1523      CC 771F
1524      FE ON
1525
1526      CD 13VB
1527      CR OC
1528      SI VE3P
1529      CD 13V
1530      CD 1343
1531      CD 75F2
1532      20 26
1533      CD 7643
1534      21 6E26
1535      36 10
1536      CD 7449
1537
1538      38 05
1539      CD 73AB
1540      18 14
1541      21 6DF9
1542      11 6E1B
1543      CD 7666
1544      CD 76F3
1545      21 07D0
1546      3E 14
1547      CD 77C3
1548      C9
1549      777A
1550      YE OF
1551      CC 771F
1552      FE ON
1553
1554      CD 13VB
1555      CR OC
1556      SI VE3P
1557      CD 13V
1558      CD 1343
1559      CD 75F2
1560      20 26
1561      CD 7643
1562      21 6E26
1563      36 10
1564      CD 7449
1565
1566      38 05
1567      CD 73AB
1568      18 14
1569      21 6DF9
1570      11 6E1B
1571      CD 7666
1572      CD 76F3
1573      21 07D0
1574      3E 14
1575      CD 77C3
1576      C9
1577      777A
1578      YE OF
1579      CC 771F
1580      FE ON
1581
1582      CD 13VB
1583      CR OC
1584      SI VE3P
1585      CD 13V
1586      CD 1343
1587      CD 75F2
1588      20 26
1589      CD 7643
1590      21 6E26
1591      36 10
1592      CD 7449
1593
1594      38 05
1595      CD 73AB
1596      18 14
1597      21 6DF9
1598      11 6E1B
1599      CD 7666
1600      CD 76F3
1601      21 07D0
1602      3E 14
1603      CD 77C3
1604      C9
1605      777A
1606      YE OF
1607      CC 771F
1608      FE ON
1609
1610      CD 13VB
1611      CR OC
1612      SI VE3P
1613      CD 13V
1614      CD 1343
1615      CD 75F2
1616      20 26
1617      CD 7643
1618      21 6E26
1619      36 10
1620      CD 7449
1621
1622      38 05
1623      CD 73AB
1624      18 14
1625      21 6DF9
1626      11 6E1B
1627      CD 7666
1628      CD 76F3
1629      21 07D0
1630      3E 14
1631      CD 77C3
1632      C9
1633      777A
1634      YE OF
1635      CC 771F
1636      FE ON
1637
1638      CD 13VB
1639      CR OC
1640      SI VE3P
1641      CD 13V
1642      CD 1343
1643      CD 75F2
1644      20 26
1645      CD 7643
1646      21 6E26
1647      36 10
1648      CD 7449
1649
1650      38 05
1651      CD 73AB
1652      18 14
1653      21 6DF9
1654      11 6E1B
1655      CD 7666
1656      CD 76F3
1657      21 07D0
1658      3E 14
1659      CD 77C3
1660      C9
1661      777A
1662      YE OF
1663      CC 771F
1664      FE ON
1665
1666      CD 13VB
1667      CR OC
1668      SI VE3P
1669      CD 13V
1670      CD 1343
1671      CD 75F2
1672      20 26
1673      CD 7643
1674      21 6E26
1675      36 10
1676      CD 7449
1677
1678      38 05
1679      CD 73AB
1680      18 14
1681      21 6DF9
1682      11 6E1B
1683      CD 7666
1684      CD 76F3
1685      21 07D0
1686      3E 14
1687      CD 77C3
1688      C9
1689      777A
1690      YE OF
1691      CC 771F
1692      FE ON
1693
1694      CD 13VB
1695      CR OC
1696      SI VE3P
1697      CD 13V
1698      CD 1343
1699      CD 75F2
1700      20 26
1701      CD 7643
1702      21 6E26
1703      36 10
1704      CD 7449
1705
1706      38 05
1707      CD 73AB
1708      18 14
1709      21 6DF9
1710      11 6E1B
1711      CD 7666
1712      CD 76F3
1713      21 07D0
1714      3E 14
1715      CD 77C3
1716      C9
1717      777A
1718      YE OF
1719      CC 771F
1720      FE ON
1721
1722      CD 13VB
1723      CR OC
1724      SI VE3P
1725      CD 13V
1726      CD 1343
1727      CD 75F2
1728      20 26
1729      CD 7643
1730      21 6E26
1731      36 10
1732      CD 7449
1733
1734      38 05
1735      CD 73AB
1736      18 14
1737      21 6DF9
1738      11 6E1B
1739      CD 7666
1740      CD 76F3
1741      21 07D0
1742      3E 14
1743      CD 77C3
1744      C9
1745      777A
1746      YE OF
1747      CC 771F
1748      FE ON
1749
1750      CD 13VB
1751      CR OC
1752      SI VE3P
1753      CD 13V
1754      CD 1343
1755      CD 75F2
1756      20 26
1757      CD 7643
1758      21 6E26
1759      36 10
1760      CD 7449
1761
1762      38 05
1763      CD 73AB
1764      18 14
1765      21 6DF9
1766      11 6E1B
1767      CD 7666
1768      CD 76F3
1769      21 07D0
1770      3E 14
1771      CD 77C3
1772      C9
1773      777A
1774      YE OF
1775      CC 771F
1776      FE ON
1777
1778      CD 13VB
1779      CR OC
1780      SI VE3P
1781      CD 13V
1782      CD 1343
1783      CD 75F2
1784      20 26
1785      CD 7643
1786      21 6E26
1787      36 10
1788      CD 7449
1789
1790      38 05
1791      CD 73AB
1792      18 14
1793      21 6DF9
1794      11 6E1B
1795      CD 7666
1796      CD 76F3
1797      21 07D0
1798      3E 14
1799      CD 77C3
1800      C9
1801      777A
1802      YE OF
1803      CC 771F
1804      FE ON
1805
1806      CD 13VB
1807      CR OC
1808      SI VE3P
1809      CD 13V
1810      CD 1343
1811      CD 75F2
1812      20 26
1813      CD 7643
1814      21 6E26
1815      36 10
1816      CD 7449
1817
1818      38 05
1819      CD 73AB
1820      18 14
1821      21 6DF9
1822      11 6E1B
1823      CD 7666
1824      CD 76F3
1825      21 07D0
1826      3E 14
1827      CD 77C3
1828      C9
1829      777A
1830      YE OF
1831      CC 771F
1832      FE ON
1833
1834      CD 13VB
1835      CR OC
1836      SI VE3P
1837      CD 13V
1838      CD 1343
1839      CD 75F2
1840      20 26
1841      CD 7643
1842      21 6E26
1843      36 10
1844      CD 7449
1845
1846      38 05
1847      CD 73AB
1848      18 14
1849      21 6DF9
1850      11 6E1B
1851      CD 7666
1852      CD 76F3
1853      21 07D0
1854      3E 14
1855      CD 77C3
1856      C9
1857      777A
1858      YE OF
1859      CC 771F
1860      FE ON
1861
1862      CD 13VB
1863      CR OC
1864      SI VE3P
1865      CD 13V
1866      CD 1343
1867      CD 75F2
1868      20 26
1869      CD 7643
1870      21 6E26
1871      36 10
1872      CD 7449
1873
1874      38 05
1875      CD 73AB
1876      18 14
1877      21 6DF9
1878      11 6E1B
1879      CD 7666
1880      CD 76F3
1881      21 07D0
1882      3E 14
1883      CD 77C3
1884      C9
1885      777A
1886      YE OF
1887      CC 771F
1888      FE ON
1889
1890      CD 13VB
1891      CR OC
1892      SI VE3P
1893      CD 13V
1894      CD 1343
1895      CD 75F2
1896      20 26
1897      CD 7643
1898      21 6E26
1899      36 10
1900      CD 7449
1901
1902      38 05
1903      CD 73AB
1904      18 14
1905      21 6DF9
1906      11 6E1B
1907      CD 7666
1908      CD 76F3
1909      21 07D0
1910      3E 14
1911      CD 77C3
1912      C9
1913      777A
1914      YE OF
1915      CC 771F
1916      FE ON
1917
1918      CD 13VB
1919      CR OC
1920      SI VE3P
1921      CD 13V
1922      CD 1343
1923      CD 75F2
1924      20 26
1925      CD 7643
1926      21 6E26
1927      36 10
1928      CD 7449
1929
1930      38 05
1931      CD 73AB
1932      18 14
1933      21 6DF9
1934      11 6E1B
1935      CD 7666
1936      CD 76F3
1937      21 07D0
1938      3E 14
1939      CD 77C3
1940      C9
1941      777A
1942      YE OF
1943      CC 771F
1944      FE ON
1945
1946      CD 13VB
1947      CR OC
1948      SI VE3P
1949      CD 13V
1950      CD 1343
1951      CD 75F2
1952      20 26
1953      CD 7643
1954      21 6E26
1955      36 10
1956      CD 7449
1957
1958      38 05
1959      CD 73AB
1960      18 14
1961      21 6DF9
1962      11 6E1B
1963      CD 7666
1964      CD 76F3
1965      21 07D0
1966      3E 14
1967      CD 77C3
1968      C9
1969      777A
1970      YE OF
1971      CC 771F
1972      FE ON
1973
1974      CD 13VB
1975      CR OC
1976      SI VE3P
1977      CD 13V
1978      CD 1343
1979      CD 75F2
1980      20 26
1981      CD 7643
1982      21 6E26
1983      36 10
1984      CD 7449
1985
1986      38 05
1987      CD 73AB
1988      18 14
1989      21 6DF9
1990      11 6E1B
1991      CD 7666
1992      CD 76F3
1993      21 07D0
1994      3E 14
1995      CD 77C3
1996      C9
1997      777A
1998      YE OF
1999      CC 771F
2000      FE ON

```

; se analizeaza pretul din buferul KEYBUF
 ; daca il gaseste incorect, nu se mai executa
 ; nici o prelucrare, ci salt la eroare
 ; daca pretul este corect, il impacheteaza si
 ; daca este indicat, executa inmultirea
 ; se introduce "A"
 ; se scade pretul din totalul de sortiment
 ; corespunzator codului din CODE
 ; se testeaza corectitudinea rezultatului
 ; daca eronat, salt la eroare
 ; daca rezultatul este corect,
 ; se executa operatiile necesare emiterii
 ; bonului
 ; daca s-a detectat o eroare,
 ; vechiul total (de anulare) al clientului
 ; se despacheteaza si se transfera in PRIBUF
 ; de unde
 ; se afiseaza
 ; si se suna soneria


```

;
; Rutina INCODE
;
; Citeste de la tastatura un cod de produs sau de ambalaj.Primeste de la
; rutina apelanta in A codul tastei apasate dupa FUNC si in C numarul de
; actionari ale tastei FUNC.
; Transmite rutinei apelante prin A codul tastei apasate dupa ultima cifra
; a codului.Distrugte: A,B,C,D,E,H,L
; Registrii utilizatii:
; A - pentru comunicatii intre rutine
; B - numarator al cifrelor codului la introducerea lor si la
; transferul lor din KEYBUF in CODE ( aici impreuna cu C )
; DE- pentru adresarea zonei CODE la transfer
; HL- pentru adresarea variabilei MRFUNC si a buferului KEYBUF
INCODE: LD HL,MRFUNC ; numarul de actionari ale tastei FUNC se
LD (HL),C ; memoreaza in variabila MRFUNC din RAM,
; pentru rutina OPFORBUY.
LD B,SCLEN ; in B numarul de cifre ale codului
CODDIG: CP POINT ; se testeaza daca ultima tasta actionata a
; fost cifra
JR C,PRCODDIG ; daca da,salt la prelucrarea ei
CALL INKEY ; daca nu,se citeste o noua tasta si
JR CODDIG ; se revine pentru testarea ei
PRCODDIG: CALL STORDISP ; cifra curenta se memoreaza si se afiseaza si
CALL INKEY ; se citeste o noua tasta
CALL CLEAR ; se testeaza daca este CLEAR si daca da,atunci
CP Z,CLEARCOD ; se sterg buferele de tastatura si de afisaj
CALL Z,CLEARCOD ; si se reinitializeaza numarul de cod
DWNZ CODDIG ; se decrementeaza numarul cifrelor codului
LD HL,KEYBUF+6 ; si daca codul nu este complet,salt inapoi
LD DE,CODE ; codul introdus se transfera din buferul de
LD BC,SCLEN ; tastatura KEYBUF in zona CODE si
LDIR CLBUFDP ; se sterg buferele KEYBUF si LEDBUF
CALL RET

```



```

MACRO-80 5.80 15-Jun-86 1-28
PAGE 1-28
CORRECT:
LD DE,GUESTBCD
CALL TMPMOV
LD HL,GUESTBCD
LD DE,PRIBUF+4
CALL BCDCI
CALL DISPSUM
LD HL,DATBCD
LD DE,PRIBUF+4
CALL BCDCI
LD HL,CODE
LD DE,PRIBUF+1
LD BC,SCLN
LDIR
CALL PRTLINE
JR ENDP
OPERROR:
CALL SUBSORT
LD HL,GUESTBCD
LD DE,PRIBUF+4
CALL BCDCI
CALL DISPSUM
ENDOP:
REI
PAGE

```

17. PROGRAM COMENTAT


```

CO
MACRO-80 5.80 15-Jun-86
PAGE 1-31
REL
L06
L07
L08
L09
L10
L11
L12
L13
L14
L15
L16
L17
L18
L19
L20
L21
L22
L23
L24
L25
L26
L27
L28
L29
L30
L31
L32
L33
L34
L35
L36
L37
L38
L39
L40
L41
L42
L43
L44
L45
L46
L47
L48
L49
L50
L51
L52
L53
L54
L55
L56
L57
L58
L59
L60
L61
L62
L63
L64
L65
L66
L67
L68
L69
L70
L71
L72
L73
L74
L75
L76
L77
L78
L79
L80
L81
L82
L83
L84
L85
L86
L87
L88
L89
L90
L91
L92
L93
L94
L95
L96
L97
L98
L99
L100
L101
L102
L103
L104
L105
L106
L107
L108
L109
L110
L111
L112
L113
L114
L115
L116
L117
L118
L119
L120
L121
L122
L123
L124
L125
L126
L127
L128
L129
L130
L131
L132
L133
L134
L135
L136
L137
L138
L139
L140
L141
L142
L143
L144
L145
L146
L147
L148
L149
L150
L151
L152
L153
L154
L155
L156
L157
L158
L159
L160
L161
L162
L163
L164
L165
L166
L167
L168
L169
L170
L171
L172
L173
L174
L175
L176
L177
L178
L179
L180
L181
L182
L183
L184
L185
L186
L187
L188
L189
L190
L191
L192
L193
L194
L195
L196
L197
L198
L199
L200
L201
L202
L203
L204
L205
L206
L207
L208
L209
L210
L211
L212
L213
L214
L215
L216
L217
L218
L219
L220
L221
L222
L223
L224
L225
L226
L227
L228
L229
L230
L231
L232
L233
L234
L235
L236
L237
L238
L239
L240
L241
L242
L243
L244
L245
L246
L247
L248
L249
L250
L251
L252
L253
L254
L255
L256
L257
L258
L259
L260
L261
L262
L263
L264
L265
L266
L267
L268
L269
L270
L271
L272
L273
L274
L275
L276
L277
L278
L279
L280
L281
L282
L283
L284
L285
L286
L287
L288
L289
L290
L291
L292
L293
L294
L295
L296
L297
L298
L299
L300
L301
L302
L303
L304
L305
L306
L307
L308
L309
L310
L311
L312
L313
L314
L315
L316
L317
L318
L319
L320
L321
L322
L323
L324
L325
L326
L327
L328
L329
L330
L331
L332
L333
L334
L335
L336
L337
L338
L339
L340
L341
L342
L343
L344
L345
L346
L347
L348
L349
L350
L351
L352
L353
L354
L355
L356
L357
L358
L359
L360
L361
L362
L363
L364
L365
L366
L367
L368
L369
L370
L371
L372
L373
L374
L375
L376
L377
L378
L379
L380
L381
L382
L383
L384
L385
L386
L387
L388
L389
L390
L391
L392
L393
L394
L395
L396
L397
L398
L399
L400
L401
L402
L403
L404
L405
L406
L407
L408
L409
L410
L411
L412
L413
L414
L415
L416
L417
L418
L419
L420
L421
L422
L423
L424
L425
L426
L427
L428
L429
L430
L431
L432
L433
L434
L435
L436
L437
L438
L439
L440
L441
L442
L443
L444
L445
L446
L447
L448
L449
L450
L451
L452
L453
L454
L455
L456
L457
L458
L459
L460
L461
L462
L463
L464
L465
L466
L467
L468
L469
L470
L471
L472
L473
L474
L475
L476
L477
L478
L479
L480
L481
L482
L483
L484
L485
L486
L487
L488
L489
L490
L491
L492
L493
L494
L495
L496
L497
L498
L499
L500
L501
L502
L503
L504
L505
L506
L507
L508
L509
L510
L511
L512
L513
L514
L515
L516
L517
L518
L519
L520
L521
L522
L523
L524
L525
L526
L527
L528
L529
L530
L531
L532
L533
L534
L535
L536
L537
L538
L539
L540
L541
L542
L543
L544
L545
L546
L547
L548
L549
L550
L551
L552
L553
L554
L555
L556
L557
L558
L559
L560
L561
L562
L563
L564
L565
L566
L567
L568
L569
L570
L571
L572
L573
L574
L575
L576
L577
L578
L579
L580
L581
L582
L583
L584
L585
L586
L587
L588
L589
L590
L591
L592
L593
L594
L595
L596
L597
L598
L599
L600
L601
L602
L603
L604
L605
L606
L607
L608
L609
L610
L611
L612
L613
L614
L615
L616
L617
L618
L619
L620
L621
L622
L623
L624
L625
L626
L627
L628
L629
L630
L631
L632
L633
L634
L635
L636
L637
L638
L639
L640
L641
L642
L643
L644
L645
L646
L647
L648
L649
L650
L651
L652
L653
L654
L655
L656
L657
L658
L659
L660
L661
L662
L663
L664
L665
L666
L667
L668
L669
L670
L671
L672
L673
L674
L675
L676
L677
L678
L679
L680
L681
L682
L683
L684
L685
L686
L687
L688
L689
L690
L691
L692
L693
L694
L695
L696
L697
L698
L699
L700
L701
L702
L703
L704
L705
L706
L707
L708
L709
L710
L711
L712
L713
L714
L715
L716
L717
L718
L719
L720
L721
L722
L723
L724
L725
L726
L727
L728
L729
L730
L731
L732
L733
L734
L735
L736
L737
L738
L739
L740
L741
L742
L743
L744
L745
L746
L747
L748
L749
L750
L751
L752
L753
L754
L755
L756
L757
L758
L759
L760
L761
L762
L763
L764
L765
L766
L767
L768
L769
L770
L771
L772
L773
L774
L775
L776
L777
L778
L779
L780
L781
L782
L783
L784
L785
L786
L787
L788
L789
L790
L791
L792
L793
L794
L795
L796
L797
L798
L799
L800
L801
L802
L803
L804
L805
L806
L807
L808
L809
L810
L811
L812
L813
L814
L815
L816
L817
L818
L819
L820
L821
L822
L823
L824
L825
L826
L827
L828
L829
L830
L831
L832
L833
L834
L835
L836
L837
L838
L839
L840
L841
L842
L843
L844
L845
L846
L847
L848
L849
L850
L851
L852
L853
L854
L855
L856
L857
L858
L859
L860
L861
L862
L863
L864
L865
L866
L867
L868
L869
L870
L871
L872
L873
L874
L875
L876
L877
L878
L879
L880
L881
L882
L883
L884
L885
L886
L887
L888
L889
L890
L891
L892
L893
L894
L895
L896
L897
L898
L899
L900
L901
L902
L903
L904
L905
L906
L907
L908
L909
L910
L911
L912
L913
L914
L915
L916
L917
L918
L919
L920
L921
L922
L923
L924
L925
L926
L927
L928
L929
L930
L931
L932
L933
L934
L935
L936
L937
L938
L939
L940
L941
L942
L943
L944
L945
L946
L947
L948
L949
L950
L951
L952
L953
L954
L955
L956
L957
L958
L959
L960
L961
L962
L963
L964
L965
L966
L967
L968
L969
L970
L971
L972
L973
L974
L975
L976
L977
L978
L979
L980
L981
L982
L983
L984
L985
L986
L987
L988
L989
L990
L991
L992
L993
L994
L995
L996
L997
L998
L999
L1000

```

```

; se introduce in B lungimea numarului de bon,
; iar rezultatul se transfera din IMPBCD la
; locul ei in DAYBCD
; totalul zilei se transfera in
; buferul IMPBCD
; totalul clientului se transfera in
; buferul DATBCD
; se introduce in buferul de imprimare
; PRTBUF,
; de unde se afiseaza
; totalul clientului se despacheteaza din nou
; si se transfera in zona SUM din EDBUF
; in C masca de selectie
; se imprima bonul clientului
; totalul clientului se reinitializeaza cu 0
; ( in HL adresa, in B lungimea, in C valoarea
; de introdus si se apeleaza rutina de
; initializare )

```



```

MACRO-80 5.80      15-Jun-86      PAGE      1-33
;
;
; Rutina PRITICK
;
; Imprima pe bonuri liniile din EDBUF selectate cu masca de selectie primita
; de la rutina apelanta in C. Un caz special reprezinta bonul de tip sinteza.
; In acest caz se apeleaza rutina SYNTTOTS care imprima toate totalurile de
; sortiment fara a le mai introduce in PRIBUF.
;
PRITICK:
LD MREDLNS          ; in B numarul de linii din EDBUF
LD IX,EDBUF+2      ; in IX adresa de inceput a textului
LD DE,EDLEN        ; din prima linie
LD A,(IX-1)        ; in DE lungimea liniilor din EDBUF
AND C
JR Z,NEXTEDLN
LD A,C
CP SYNTMASK
JR NZ,NORMLINE
LD A,(IX-2)
CP 86H
JR NZ,NORMLINE
SPEC:
CALL SYNTTOTS
JR NEXTEDLN
NORMLINE:
CALL TRANLINE
CALL PRTLINE
NEXTEDLN:
ADD IX,DE
DJNZ EDLINE
HL,EDBFRE
LD A,EDBTIME
CALL BEEP
RET

```


Rutina SYNTTOTS

; Imprima toate totalurile de sortiment, in cadrul bonului de tip sinteza.

```

SYNTTOTS:  PUSH BC
           DE BC,DETCEN-3
           BC 00000000
           PUSH BC
           LD 02H C,01X
           CALL 02H IMPLCODE
           LD 02H B,100

NSORTT:   PUSH BC
           CALL SORTADR
           LD 02H DE,PRTBUF+4
           CALL BCDCI
           LD 02H HL,CODE
           LD 02H DE,PRTBUF+1
           LD 02H BC,SLEN
           LD 02H DIR
           LD 02H A,ASTER
           LD 02H (PRTBUF+15),A
           CALL PRTLINE
           LD 02H B,SLEN
           LD 02H HL,CODE+SLEN-1
           CALL EBCDINC
           POP 02H BC
           DJNZ NSORTT

           POP BC
           POP DE
           RET
           PAGE

```

```

; se salveaza registrii utilizati
; se initializeaza zona CODE
; cu codul 00
; se initializeaza numaratorul totalurilor
; se salveaza numaratorul
; determina adresa totalului corespunzator co-
; dului din CODE si transfera totalul,despa-
; chetat,in PRTBUF
; codul din CODE se introduce in PRTBUF
; inaintea totalului
; se introduce "*" dupa total
; se imprima linia formata
; se incrementeaza codul din CODE ( in B
; lungimea,in HL adresa si cheama
; rutina de incrementare )
; se reface numaratorul,se decrementeaza si
; daca nu am terminat,salt inapoi pentru
; urmatorul total
; se refac registrii salvati

```



```

;
; Rutina SUBBCD
;
; Scade continutul buferului DATBCD din continutul buferului TMPBCD, rezultatul
; formindu-se in TMPBCD. Indicatorul CY indica rutinei apelante daca rezultatul
; scaderii este corect ( la rezultat negativ CY = 1 ).
; Distruger: A
; Registrii utilizati:
; DE - pentru adresarea locatiilor din TMPBCD
; HL - pentru adresarea locatiilor din DATBCD
; B - numarator
; A - rezultate partiale
SUBBCD.

```

```

7575 BC
7575 PUSH DE
7576 PUSH HL
7577 PUSH DE, TMPBCD+BCDLEN-1
7578 LD HL, DATBCD+BCDLEN-1
757B LD B, BCDLEN
757E XOR A
7580
7581
7581 LD A, (DE)
7582 SBC A, (HL)
7583 DAA
7584 LD (DE), A
7585 DEC DE
7586 DEC HL
7587 DJNZ SUBBYTE
7589 POP HL
758A POP DE
758B POP BC
758C RET
PAGE

```

Rutina MULTBCD

```

;
;
; Rutina MULTBCD
; Este apelata din PRELPROC, daca indicatorul de inmultire este setat.
; In momentul apelului avem in TMPBCD inmultitorul, iar in DATBCD deinmultitul.
; Rutina trece inmultitorul in WORKBCD, aduce TMPBCD la 0 si executa adunarile
; repetate, in functie de cifrele inmultitorului. La sfirsit rezultatul final
; se trece din TMPBCD in DATBCD.
; Distruge: A,B,C,D,E,H,L
;

```

```

MULTBCD:
LD DE,WORKBCD ; se transfera inmultitorul din IMPBCD
CALL TMPMOV ; in WORKBCD
LD B,BCDLEN ; buferul TMPBCD se initializeaza cu 0
LD 21,6E03 ; ( in B lungimea, in HL adresa, in C valoarea
LD 0E 00 ; si cheama rutina de initializare )
CALL FILLBYTES
LD B,BCDLEN ; in B lungimea buferelor implicate
LD 0E 0A ; in C numarul de cifre ai operanzilor

SHIFTS:
LD 21,6E02 ; cifrele din WORKBCD se deplaseaza
CALL MUL10 ; cu o pozitie la stinga ( cea mai semnificati-
; va intra in A )
LD E,A ; se salveaza
XOR A ; A = 0
LD 21,6E07 ; cifrele din TMPBCD se deplaseaza cu
CALL MUL10 ; o pozitie la stinga ( in pozitia cea mai
; putin semnificativa intra 0 )

INC E
MULT:
DEC JR 2,MULTENDT ; se aduna deinmultitul la rezultatul partial
CALL ADDRBCD ; de cite ori indica cifra cea mai semnifica-
JR MULT ; tiva a inmultitorului

MULTENDT:
DEC JR NZ,SHIFTS ; se decrementeaza numaratorul cifrelor
; daca nu am terminat inmultirea, salt inapoi

```



```

MACRO-80 5.80      15-Jun-86      PAGE 1-40
;
; Rutina MULTOP
;
; Apelata in cazul actionarii tastei "*" dupa introducerea unui pret, rutina
; analizeaza pretul introdus si daca este corect, il transfera in buferul
; IMPBCD si seteaza indicatorul de inmultire.
MULTOP:
; se salveaza A
PUSH AF
; se analizeaza pretul introdus si daca
CALL SYNTAN
; este eronat, salt la sfirsitul rutinei
JR NZ, ENDMOP
; daca a fost corect, atunci in urma analizei
; se gaseste in buferul EBCD
CALL EBCDDBC
; pretul din EBCD in DATBCD
LD HL, DATBCD
; iar din DATBCD in buferul IMPBCD
MOV TMP, HL
;
; se seteaza indicatorul de inmultire
SET O, E
EXX
;
; la sfirsitul rutinei se sterg KEYBUF si
; LEDBUF si se reface A
CALL CLBUFDP
POP AF
RET
PAGE
    
```



```

MACR0-80 5.80      15-Jun-86      PAGE 1-41
;
;
; Rutina EBCDINC
;
; Incrementeaza numarul ( in BCD extins ) din zona de memorie indicata de HL
; ( HL indica cifra cea mai putin semnificativa ).
; Distruge: A,B,H,L
; Registrii utilizati:
; B - numarul de octeti ai zonei de incrementat
; HL - pentru adresa octetilor zonei
; A - utilizat in operatii
;
EBCDINC: INC (HL) ; se incrementeaza cifra curenta a numarului
LD A,(HL) ;
CP 10 ; si se testeaza daca s-a ajuns la 10
RET NZ ; daca nu, operatia de incrementare s-a terminat
LD (HL),0 ; daca da, se schimba 10 cu 0 si
DEC HL ; se trece la cifra mai semnificativa
DJNZ EBCDINC
RET
PAGE

```

```

1845 CB
1846 BA
1847 AV
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500

```

; SYNTAX:
; LD B, EBCDLEN
; LD HL, EBCD
; LD C, 0
; CALL FILLBYTES
; LD HL, KEYBUF
; LD B, BUFLen
; LD D, 0
; CALL SRCMBLK
; LD E, A
; RES 7, A
; CP BLANK
; JR Z, DECPART
; INTPART: CALL STOREINT
; TINTEND: LD A, B
; OR A
; JR Z, FORMMES
; LD A, E

; Rutina SYNTAX
; Analizeaza pretul din KEYBUF si il transfera in EBCD. La sfirsit pozitioneaza
; indicatorul Z pentru a indica rutinei apelante daca pretul a fost corect
; sau nu.
; Distruge: A, B, C, D, E, H, L, IX
; Registrii utilizati:
; A - la diferite operatii
; B - numarator al locatiilor buferului KEYBUF
; C - numarator al cifrelor zecimale din EBCD
; D - indicator de eroare
; E - pentru salvarea din A
; HL - pentru adresarea locatiilor buferului KEYBUF
; IX - pentru adresarea partii zecimale a buferului EBCD

; in B lungimea buferului EBCD,
; in HL adresa
; se initializeaza locatiile buferul cu 0
; in continuare HL va adresa locatiile din
; KEYBUF, in B lungimea buferului
; D va fi indicatorul de eroare
; se cauta in KEYBUF, de la stinga, prima locatie
; care nu contine spatiu
; se salveaza octetul citit in E si
; se reseteaza bitul indicator de punct zecimal
; daca a ramas spatiu, inseamna ca pretul nu are
; parte intreg, deci salt la analiza partii
; zecimale
; daca nu, incepe analiza partii intregi
; prima cifra se memoreaza in EBCD
; se testeaza daca numaratorul locatiilor lui
; KEYBUF a ajuns la 0; daca da, analiza s-a
; terminat si salt la formarea mesajului
; se reface octetul citit si

MACRO-80	5.80	15-Jun-86	PAGE	1-43
7615	CB 7F		BIT	7,A
7617	20 0B		JR	NZ,DECPART
7619	5E		LD	E,(HL)
761A	23		INC	HL
761B	05		DEC	B
761C	7B		LD	A,E
761D	CB BF		RES	7,A
761F	CD 76E0		CALL	STOREINT
7622	18 EC		JR	TINTEND
7624	DD 21 6E15		DECPART:	LD IX,EBCD+BUFLEN
7628	0E 02		LD	C,2
762A	5E		LD	E,(HL)
762B	23		INC	HL
762C	CB 7B 0B		BIT	7,E
762E	28 04		JR	Z,WITHOUTP
7630			LD	D,1
7630	16 01		LD	A,C
7632	18 0C		JR	FORMMES
7634			LD	A,C
7634	79		OR	A,A
7635	B7 0A		OR	A,A
7636	28 06		JR	Z,TDECEND
7638			LD	(IX+0),E
7638	DD 73 00		INC	IX
763B	DD 23		DEC	C
763D	0D		JR	NEXTDEC
763E	10 EA		FORMMES:	LD A,D
7640			OR	A
7641	B7		RET	
7642	C9		PAGE	1-43

; se testeaza bitul indicator al punctului
; daca este 1, atunci analiza partii intrregi
; s-a terminat si salt la partea zecimala
; daca a fost 0, se ia un nou octet din KEYBUF
; se decrementeaza numaratorul octetilor
; octetul citit se transfera in A
; se reseteaza bitul indicator de punct zecimal
; si se memoreaza cifra in EBCD
; IX va adresa partea zecimala din EBCD, iar
; C va fi numaratorul cifrelor zecimale
; se ia un nou octet din KEYBUF
; se testeaza bitul indicator de punct zecimal
; daca nu contine punct, salt
; daca da, inseamana ca sint mai multe puncte in
; pret, cea ce este eronat, deci se seteaza
; indicatorul de eroare si salt la formare
; mesaj
; se testeaza daca numaratorul cifrelor zecima-
; le este 0; daca da, inseamna ca cifra nu face
; parte din primele 2 cifre zecimale, deci nu
; se retine
; in caz contrar se memoreaza cifra
; si se decrementeaza numaratorul cifrelor
; zecimale
; daca KEYBUF mai are cifre neanalizate, salt
; inapoi
; indicatorul de eroare in A si
; se pozitioneaza indicatorul Z

Rutina BCDCI

; Despatcheaza numarul de la adresa indicata de HL in cod intern, la adresa
 ; indicata de DE, introduce punctul zecimal si inlocuieste zerourile
 ; nesemnificative cu spatii.
 ; Distruge: A,B,C,D,E,H,L
 ; Registrii utilizati:
 ; HL- pentru adresarea zonei sursa in timpul despachetarii si a zonei
 ; ce contine numarul despachetat la introducerea punctului zecimal
 ; la inlocuirea zerourilor nesemnificative
 ; DE- pentru adresarea zonei destinatie in timpul despachetarii

```

7666 MOVMP      ; se transfera numarul de despachetat in HL
7667 LD HL, TMPBCD ; TMPBCD si in HL adresa buferului
7668 LD B, BCDLEN ; in B numarul de octeti de despachetat
7669 LD DEC C, OFFH ; A = 0 si indicatorii pozitionati
7670 AF DE      ; se salveaza adresa la care se formeaza
7671 RB D5     ; numarul in cod intern

7672          ; cifra mai semnificativa a octetului curent
7673 ED 6F      ; din numarul de despachetat se transfera in
7674 12         ; zona destinatie
7675 13         ; A = 0 si se reface cifra mai putin semnifi-
7676 AF 05      ; cativa in octetul curent din numarul de
7677 ED 67      ; despachetat si se transfera in zona
7678 ED A0      ; destinatie
7679           ; daca nu am terminat, salt inapoi
767B 10 F5     ; in BC numarul cifrelor zecimale,
767D 01 0002   ; si aceste doua cifre se deplaseaza cu o
7680 62        ; pozitie la dreapta pentru a face loc
7681 68        ; punctului zecimal
7682 2B 8B     ;
7683 ED 88     ;
7685 INC       ; se introduce punctul zecimal
7686 LD (HL), POINT ;
  
```


17. PROGRAM COMENTAT

```

;
;
;
; Rutina STORDISP
;
; Introduce cifra sau punctul primit de la rutina apelanta prin A in buferele
; KEYBUF si LEDBUF.
; Nu distruge nici un registru.
STORDISP:  CALL STORDNUM
;
; CALL STORENUM
; CALL DISPNUM
; RET
;
; Rutina STOREINT
;
; Memoreaza cifra primita de la rutina apelanta prin A in EBCC, in pozitia cea
; mai putin semnificativa a intregilor.
; Nu distruge nici un registru.
; Registrul utilizati:
; A - contine cifra de introdus in EBCC
; HL - indica adresa sursa la deplasarea spre stinga
; DE - indica adresa destinatie la deplasare
; BC - numarator la deplasare
;
; STOREINT:  PUSH BC
;           PUSH DE
;           PUSH HL
;           LD DE,EBCC
;           LD HL,EBCC+1
;           LD BC,EBCOLEN-3
;           LDIR
;           LD (DE),A
;           POP HL
;           POP DE
;           POP BC
;           RET
;
; se salveaza registrul utilizati
; se refac registrii salvati

```

```

76E0 C5
76E1 D5
76E2 E5
76E3 11 6E0D
76E6 21 6E0E
76E9 01 0007 2'80 12-300-89
76EC ED 80
76EE 12
76EF E1
76F0 01
76F1 C9
76F2 C9

```


76F3
 76F4
 76F5
 76F6
 76F7
 76F8
 76F9
 76FA
 76FB
 76FC
 76FD
 76FE
 7701
 7702
 7704

MACRO-80 5.80

15-Jun-86

1-VOCE
 2-SEL
 3-BO
 4-BO
 5-BO
 6-BO
 7-BO
 8-BO
 9-BO
 10-BO
 11-BO
 12-BO
 13-BO
 14-BO
 15-BO
 16-BO
 17-BO
 18-BO
 19-BO
 20-BO
 21-BO
 22-BO
 23-BO
 24-BO
 25-BO
 26-BO
 27-BO
 28-BO
 29-BO
 30-BO
 31-BO
 32-BO
 33-BO
 34-BO
 35-BO
 36-BO
 37-BO
 38-BO
 39-BO
 40-BO
 41-BO
 42-BO
 43-BO
 44-BO
 45-BO
 46-BO
 47-BO
 48-BO
 49-BO
 50-BO
 51-BO
 52-BO
 53-BO
 54-BO
 55-BO
 56-BO
 57-BO
 58-BO
 59-BO
 60-BO
 61-BO
 62-BO
 63-BO
 64-BO
 65-BO
 66-BO
 67-BO
 68-BO
 69-BO
 70-BO
 71-BO
 72-BO
 73-BO
 74-BO
 75-BO
 76-BO
 77-BO
 78-BO
 79-BO
 80-BO
 81-BO
 82-BO
 83-BO
 84-BO
 85-BO
 86-BO
 87-BO
 88-BO
 89-BO
 90-BO
 91-BO
 92-BO
 93-BO
 94-BO
 95-BO
 96-BO
 97-BO
 98-BO
 99-BO
 100-BO

PAGE 1-51

1-BO
 2-BO
 3-BO
 4-BO
 5-BO
 6-BO
 7-BO
 8-BO
 9-BO
 10-BO
 11-BO
 12-BO
 13-BO
 14-BO
 15-BO
 16-BO
 17-BO
 18-BO
 19-BO
 20-BO
 21-BO
 22-BO
 23-BO
 24-BO
 25-BO
 26-BO
 27-BO
 28-BO
 29-BO
 30-BO
 31-BO
 32-BO
 33-BO
 34-BO
 35-BO
 36-BO
 37-BO
 38-BO
 39-BO
 40-BO
 41-BO
 42-BO
 43-BO
 44-BO
 45-BO
 46-BO
 47-BO
 48-BO
 49-BO
 50-BO
 51-BO
 52-BO
 53-BO
 54-BO
 55-BO
 56-BO
 57-BO
 58-BO
 59-BO
 60-BO
 61-BO
 62-BO
 63-BO
 64-BO
 65-BO
 66-BO
 67-BO
 68-BO
 69-BO
 70-BO
 71-BO
 72-BO
 73-BO
 74-BO
 75-BO
 76-BO
 77-BO
 78-BO
 79-BO
 80-BO
 81-BO
 82-BO
 83-BO
 84-BO
 85-BO
 86-BO
 87-BO
 88-BO
 89-BO
 90-BO
 91-BO
 92-BO
 93-BO
 94-BO
 95-BO
 96-BO
 97-BO
 98-BO
 99-BO
 100-BO

1-BO
 2-BO
 3-BO
 4-BO
 5-BO
 6-BO
 7-BO
 8-BO
 9-BO
 10-BO
 11-BO
 12-BO
 13-BO
 14-BO
 15-BO
 16-BO
 17-BO
 18-BO
 19-BO
 20-BO
 21-BO
 22-BO
 23-BO
 24-BO
 25-BO
 26-BO
 27-BO
 28-BO
 29-BO
 30-BO
 31-BO
 32-BO
 33-BO
 34-BO
 35-BO
 36-BO
 37-BO
 38-BO
 39-BO
 40-BO
 41-BO
 42-BO
 43-BO
 44-BO
 45-BO
 46-BO
 47-BO
 48-BO
 49-BO
 50-BO
 51-BO
 52-BO
 53-BO
 54-BO
 55-BO
 56-BO
 57-BO
 58-BO
 59-BO
 60-BO
 61-BO
 62-BO
 63-BO
 64-BO
 65-BO
 66-BO
 67-BO
 68-BO
 69-BO
 70-BO
 71-BO
 72-BO
 73-BO
 74-BO
 75-BO
 76-BO
 77-BO
 78-BO
 79-BO
 80-BO
 81-BO
 82-BO
 83-BO
 84-BO
 85-BO
 86-BO
 87-BO
 88-BO
 89-BO
 90-BO
 91-BO
 92-BO
 93-BO
 94-BO
 95-BO
 96-BO
 97-BO
 98-BO
 99-BO
 100-BO

1-BO
 2-BO
 3-BO
 4-BO
 5-BO
 6-BO
 7-BO
 8-BO
 9-BO
 10-BO
 11-BO
 12-BO
 13-BO
 14-BO
 15-BO
 16-BO
 17-BO
 18-BO
 19-BO
 20-BO
 21-BO
 22-BO
 23-BO
 24-BO
 25-BO
 26-BO
 27-BO
 28-BO
 29-BO
 30-BO
 31-BO
 32-BO
 33-BO
 34-BO
 35-BO
 36-BO
 37-BO
 38-BO
 39-BO
 40-BO
 41-BO
 42-BO
 43-BO
 44-BO
 45-BO
 46-BO
 47-BO
 48-BO
 49-BO
 50-BO
 51-BO
 52-BO
 53-BO
 54-BO
 55-BO
 56-BO
 57-BO
 58-BO
 59-BO
 60-BO
 61-BO
 62-BO
 63-BO
 64-BO
 65-BO
 66-BO
 67-BO
 68-BO
 69-BO
 70-BO
 71-BO
 72-BO
 73-BO
 74-BO
 75-BO
 76-BO
 77-BO
 78-BO
 79-BO
 80-BO
 81-BO
 82-BO
 83-BO
 84-BO
 85-BO
 86-BO
 87-BO
 88-BO
 89-BO
 90-BO
 91-BO
 92-BO
 93-BO
 94-BO
 95-BO
 96-BO
 97-BO
 98-BO
 99-BO
 100-BO

; Transfera in buferul de afisaj LEDBUF numarul continut de buferul PRIBUF.
 ; Distruge: A,B,H,L
 ; Registrii utilizati:
 ; HL - pentru adresarea buferului PRIBUF
 ; B - numarator
 ; A - la transmiterea cifrei de afisat rutinei DISPNUM
 ; Transfera in buferul de afisaj LEDBUF numarul de afisat
 ; Distruge: A,B,H,L
 ; Registrii utilizati:
 ; HL - pentru adresarea buferului PRIBUF
 ; B - numarator
 ; A - la transmiterea cifrei de afisat rutinei DISPNUM
 ; Transfera in buferul de afisaj LEDBUF numarul de afisat
 ; Distruge: A,B,H,L
 ; Registrii utilizati:
 ; HL - pentru adresarea buferului PRIBUF
 ; B - numarator
 ; A - la transmiterea cifrei de afisat rutinei DISPNUM

Rutina DISPNUM

LD HL,PRIBUF+4
 LD B,BUFLEN+2+1
 CALL SRCNBLK
 DEC HL
 INC B
 LD A,(HL)
 DISPNUM
 INC HL
 DJNZ DISP1
 PAGE

; se transfera cifra curenta a numarului in A
 ; si se introduce in buferul de afisaj
 ; daca mai sunt cifre de transferat,salt inapoi
 ; la buferul de afisaj

```

;
;
; Rutina MOVTMP
; Transfera continutul zonei indicate de HL in IMPBCD.
; Distruge: H,L
; Registrii utilizati:
; HL- pentru adresarea zonei sursa
; DE- pentru adresarea zonei destinatie ( IMPBCD )
; BC- numarul de octeti de transferat
;
MOVTMP: PUSH BC ; se salveaza registrii utilizati
        PUSH DE
        LD HL,IMPBCD ; in DE adresa buferului IMPBCD,iar
        LD BC,BCDLEN ; in BC lungimea buferului
        LDIR ; se executa transferul
        POP BC
        RET
;
; Rutina TMPMOV
; Transfera continutul buferului IMPBCD la adresa indicata de DE
; Distruge: D,E
; Registrii utilizati:
; HL- pentru adresarea zonei sursa ( IMPBCD )
; DE- pentru adresarea zonei destinatie
; BC- numarul de octeti de transferat
;
TMPMOV: PUSH BC ; se salveaza registrii utilizati
        PUSH HL
        LD HL,TMPBCD ; in HL adresa,iar
        LD BC,BCDLEN ; in BC lungimea buferului IMPBCD
        LDIR ; se executa transferul
        POP HL
        POP BC
        RET

```



```

331E C8
331D C1
331C E1
331V ED B0
331A MACRO-80, 5, 80
331V S1 0E03
3313 E2
3313 C2
3313 80 5, 80 15-Jun-86
PAGE 1-53
BC BCODEM
HL LMBUCD
HC
BC

```

Rutina CLBUFDP

```

; Sterge buferele KEYBUF si LEDBUF, introducind in ele spatii.
; Nu distruge nici un registru.
CLBUFDP:

```

```

771F
771F CD 7726
7722 CD 7735
7725 C9

```

```

CALL CLKEYBUF ; se sterge KEYBUF
CALL CLDISP ; se sterge LEDBUF
RET

```

Rutina CLKEYBUF

```

; Sterge buferul KEYBUF, introducind spatii.
; Nu distruge nici un registru.
CLKEYBUF:

```

```

7726 C5
7727 E5
7728 06 08
772A 21 6E29
772D 0E 20
772F CD 774A
7732 E1
7733 C1
7734 C9

```

```

CALL PUSH BC ; se salveaza registrii utilizati
CALL PUSH HL ; in B lungimea buferului
LD B, BUFLen ; in HL adresa de inceput
LD HL, KEYBUF ; in C valoarea de introdus, adica spatiu
LD C, BLANK ; si se apeleaza rutina de initializare
CALL POP R15, HL
CALL POP R15, BC
RET
PAGE

```

```

;
; Rutina CLDISP
;
; Sterge, introducind spatii, buferul LEDBUF de afisaj.
; Nu distruge nici un registru.
; Registrii utilizati:
; HL - pentru adresa buferului
; B - lungimea buferului
; C - contine valoarea ce se introduce in bufer
; se salveaza registrii utilizati
; in B lungimea buferului
; in HL adresa de inceput
; in C spatiu de afisaj ( segmentele stinse )
; se apeleaza rutina de initializare
; se refac registrii salvati

;
; Rutina CLEARCOD
; Sterge buferele KEYBUF si LEDBUF si reinitializeaza numaratorul cifrelor
; codului. Distruge: B
; Registru utilizat: B - numarator al cifrelor codului
CLEARCOD:
CALL CLBUFDP ; se sterg KEYBUF si LEDBUF
LD B, SCLEN+1 ; se reinitializeaza numaratorul
RET

;
; Rutina FILLBYTES
; Initializeaza o zona de memorie ce incepe la adresa data de HL, avind lungimea
; indicata de B, cu valoarea continuta de C. Distruge: B, H, L
FILLBYTES:
LD (HL), C ; se transfera continutul lui C in locatia
INC HL ; curenta si se trece la urmatoarea locatie
DJNZ FILLBYTES
RET

```



```

PAGE 1-55
MACRO-80 5.80 15-Jun-66
LD PAGE 1-55
INC 33
LD 33
MACRO-80 5.80
15-Jun-66
Rutina EXPAND
; Transfera mesajele din EPROM ( MESSAGE ) in RAM ( EDBUF ) expandand prin
; completare cu blankuri fiecare linie pina la lungimea de 15 caractere.
; Distruge: A,B,C,D,E,H,I
; Registrii utilizati:
; HL- pentru adresarea zonei EDBUF
; DE- pentru adresarea zonei MESSAGE
; C - numarul de linii de transferat
; B - numarul de octeti ai unei linii
; A, (DE) - adresa din EPROM a mesajului
; B, EDI, EDIEN - adresa din RAM a mesajului
; C, M, EDI, EDIEN - adresa din EPROM a mesajului
; DE, M, EDI, EDIEN - adresa din RAM a mesajului
; HL, EDI, EDIEN - adresa din EPROM a mesajului
; M, EDI, EDIEN - adresa din RAM a mesajului
EXPAND: LD C, M
LD DE, MESSAGE
LD HL, EDBUF
LD A, (DE)
EXPLINE: LD B, EDI
LD B, EDIEN
EXPCHRS: LD HL, A
INC DE
INC HL
DEC B
LD A, (DE)
LD B, 7
BIT 7, A
JR Z, EXPCHRS
EXPSPAC: LD HL, BLANK
INC HL
DJNZ EXPSPAC
DEC C
JR NZ, EXPLINE
LD HL, SUM+11
LD HL, A
LD HL, A
RET
PAGE

```


MACR0-80 5.80

15-Jun-86

PAGE 1-57

779A F5 3E
779B 01 0090779E ED 78
77A0 2F 03
77A1 E6 0C
77A2 20 F977A5 F1 1B 1B
77A6 C1 1B 1B
77A7 E1 0E 0E
77A8 C9 20 20F977A9 2C 00 11
77AA 2C 00 11
77AB CB 00 1177AD 38 D5
77AF 2E FF 1F
77B1 18 D177B3 FE FF
77B5 FD 01 7F
77B7 FB 03 7777B9 F7 05 77
77BB EF 07 20
77BD DF 09 7777BF BF 0B 77
77C1 7F 0D 77PUSH
LDAF
BC,SVSYNIN
CPL
AND
JRPOP
POP
POP
RETINC
INC
RLC
JRLD
LD
LD
JRC,NXTLIN
L,-1
NXTLINDB
DB
DB
DB
DB
DB
DB
PAGE

; KEYTAB:

11111108,OFFH
1111101B,01H
1111011B,03H
1110111B,05H
1101111B,07H
1011111B,09H
0111111B,0BH
0111111B,0DHPUSH
LDAF
BC,SVSYNIN
CPL
AND
JRPOP
POP
POP
RETINC
INC
RLC
JRLD
LD
LD
JRC,NXTLIN
L,-1
NXTLINDB
DB
DB
DB
DB
DB
DB
PAGE

; KEYTAB:

11111108,OFFH
1111101B,01H
1111011B,03H
1110111B,05H
1101111B,07H
1011111B,09H
0111111B,0BH
0111111B,0DH

; salveaza valoarea formata

; in programul apelant se revine numai dupa ce

; se ridica tasta

; se refac codul tastei si registrii utilizati

; daca tastele liniei curente nu sint active

; se trece la urmatoarea linie,shimbind

; codul liniei curente si masca

; daca nu mai sint linii netestate,se revine

; la prima din dreapta

```

MACRO-80 5.80 15-Jun-86 1-58
PAGE 1-58
;
; Rutina BEEP
; Este rutina generatoare de sunet.
; Distrugte: A,H,L
; Registrii utilizati:
; HL- contine frecventa in Hz
; A - contine un numar proportional cu durata D
;
; BEEP:
; PUSH BC
; PUSH DE
; PUSH HL
; SCF
; CCF
; TIME = D * 128
; ( se calibreaza durata tonului )
; RRA
; LD HL,0
; LD HL,A
; LD HL,A
;
; NCYCLE = ONEHZ / F
; ( se calculeaza numarul de temporizari elementare )
; (SP),HL
; HL = F,(SP) = TIME
; LD HL,0
; LD HL,A
; LD HL,A
;
; CALL COMPUTE
;
; MRIMP = TIME / NCYCLE
; ( se calculeaza numarul de impulsuri de frecventa F ce vor fi generate pentru
; a se obtine durata D )
; EX (SP),HL
; POP DE
; PUSH DE

```

17. PROGRAM COMENTAT


```

MACRO-80 5.80      15-Jun-86      PAGE 1-59
D1
D3
77D9 CD 7821      ; NRIMP = TIME / NCYCLE, NRIMP in BC
77DC 03          ; se evita BC = 0

MACRO-80 5.80      15-Jun-86      ; Se emite sunetul
D1 POP DE
77DE CD 77E4      CALL SING DE
77E1 D1          / POP DE
77E2 C1          / POP DE
77E3 C9          RET BC, DE
77E4
77E5
77E6
77E7
77E8
77E9
77EA
77EB
77EC
77ED
77EE
77EF
77F0
77F1
77F2
77F3
77F4
77F5
77F6
77F7
77F8
77F9
77FA
77FB
77FC
77FD
77FE
77FF
7800
7801
7802
7803
7804
C9

; Rutina SING DEPT
; Subrutina de emitere a sunetului. Primeste in BC numarul de impulsuri de emis
; ( NRIMP ), in DE durata unui impuls ( NCYCLE ).
; In dreptul instructiunilor numarul de stari necesare executiei.
SING:
SINGI:
LD A, (WITNESS)
SET BEEPBIT, A
POP DE
PUSH DE
OUT (SYSOUT), A
CALL DELAYO, A, 17+DE*50+10
LD C, A, (WITNESS)
RES BEEPBIT, A
POP DE
PUSH DE
OUT (SYSOUT), A
CALL DELAYO, A, 17+DE*50+10
DEC BC
LD A, C
OR B
JP NZ, SINGI
POP DE
RET

; se salveaza NCYCLE
; (13) se foloseste celula maror a portului
; de iesire
; (8)
; (10)
; (10)
; (11)
; (13)
; (8)
; (10)
; (11)
; (13)
; (6) NRIMP = NRIMP - 1
; (4)
; (4)
; (10)

```

```

; Rutina COMPUTE
; Rutina de impartire cu rotunjirea citului.Primeste de la rutina apelanta
; in HL deimpartitul,iar in DE impartitorul si returneaza in HL citul
; rotunjit.
COMPUTE: CALL DIVIDE

```

```

CALL DIVIDE
PUSH BC ; se salveaza citul nerotunjit
ADD HL,HL ; se dubleaza restul
CALL COMP ; daca restul dublat este mai mic decat
POP HL ; impartitul,atunci citul nu se schimba
DIA RET C ; altfel,el este incrementat,rotunjindu-se
INC VDD HL ; rezultatul
RET ;

; Rutina COMP
; Compara registrii DE si HL.Indicatorul CY setat indica DE > HL.
COMP: OR STV A
PUSH HL
SBC HL,DE
POP HL
RET

```

```

; Rutina DELAY
; Asigura decrementarea variabilei NCYCLE de la o valoare initiala la 0,
; astfel incit fiecare iteratie sa dureze 50 tacti procesor.
DELAY: DEC DE
JR DELAY1
DELAY1: JR DELAY2
DELAY2: LD A,E
OR D
NZ,DELAY
RET

```

```

; (6)
; (12) salt de temporizare
; (12) salt de temporizare
; (4) se testeaza daca DE = 0
; (4)
; (12) daca nu,salt inapoi

```


Rutina PRTLINE

```

;
;
; Rutina PRTLINE
; Imprima continutul buferului PRIBUF pe ambele bonuri, iar la terminarea
; imprimarii sterge PRIBUF, introducand spatii
;

```

```

PRTLINE: PUSH AF

```

```

          PUSH BC

```

```

          PUSH DE

```

```

          PUSH HL

```

```

          CALL MOTON

```

```

          WAITPOZ:

```

```

            IN A,(SYSIN)

```

```

            BIT CARLIMIT,A

```

```

            JR NZ,WAITPOZ

```

```

            LD E,INERTDEL

```

```

            CALL DELAY

```

```

            CALL PRT18C

```

```

            LD E,BITDEL

```

```

            CALL DELAY

```

```

            CALL PRT18C

```

```

            LD B,PRTLEN

```

```

            LD HL,PRIBUF

```

```

            LD C,BLANK

```

```

            CALL FILLBYTES

```

```

            CALL MOTOFF

```

```

            HL POP DE

```

```

            POP BC

```

```

            POP AF

```

```

            RET

```

```

            PAGE PAGE

```

783A

783A

783B

783C

783D

783E

7841

7841

7843

7845

7847

7849

784C

784F

7851

7854

7857

7859

785C

785E

7861

7864

7865

7866

7867

7868


```

;
; Rutina PRTCOL
;
; Imprima o coloana dintr-un caracter, cu intirzierile corespunzatoare.
; Rutina apelanta transmite coloana prin A. Coloana de simbolii este justa
; Distruge: A,D,E
; Registrii utilizati:
; A - pentru a transmite comenzile la portul acelor
; E - ca parametru pentru rutina DELAY
; D - utilizat de rutina DELAY
;
; PRTCOL:
;      OUT (PRTPORT),A ; coloana primita se trimite la portul acelor
;      LD E,SEIDEL ; intirziere de ace active,
;      CALL DELAY ; de 300 microsecunde
;      LD A,RESHEAD ; comanda de ridicare a acelor
;      OUT (PRTPORT),A ; se trimite la port
;      LD E,RESEDEL ; intirziere de ace resetate,
;      CALL DELAY ; de 1200 microsecunde
;      RET
;
; Rutina DELAY
;
; Realizeaza o intirziere de ( 5,6 x BASEDEL x valoarea din E ) microsecunde.
; Distruge: D,E
; Registrii utilizati:
; E - parametru de intirziere trimis de rutina apelanta
; D - numarator de intirziere
;
; DELAY:
;      LD D,BASEDEL ; valoarea de decrementat pentru intirzierea
;      DEL: DEC D ; de baza ( 300 microsecunde )
;          ; (4) se decrementeaza D
;          JP NZ,DEL ; (10) pina cind devine 0
;          DEC E ; (4) se decrementeaza E; daca nu s-a ajuns
;          JP NZ,DELAY ; (10) la 0, se revine la un nou ciclu a lui D
;          RET
;
; PAGE

```

```

38CB MACRO 80 5.80 15-Jun-84
38CC 58 E3
38CD 4E 50
38CE 02
38CF 53
38D0 3E
38D1
38D2
38D3
38D4
38D5
38D6
38D7
38D8
38D9
38DA
38DB
38DC
38DD
38DE
38DF
38E0
38E1
38E2
38E3
38E4
38E5
38E6
38E7
38E8
38E9
38EA
38EB
38EC
38ED
38EE
38EF
38F0
38F1
38F2
38F3
38F4
38F5
38F6
38F7
38F8
38F9
38FA
38FB
38FC
38FD
38FE
38FF
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
390A
390B
390C
390D
390E
390F
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
391A
391B
391C
391D
391E
391F
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
392A
392B
392C
392D
392E
392F
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
393A
393B
393C
393D
393E
393F
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
394A
394B
394C
394D
394E
394F
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
395A
395B
395C
395D
395E
395F
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
396A
396B
396C
396D
396E
396F
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
397A
397B
397C
397D
397E
397F
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
398A
398B
398C
398D
398E
398F
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999

```

```

78B1 3E B0
78B2 D3 B0
78B3 C9
78B4
78B5
78B6
78B7
78B8
78B9
78BA
78BB
78BC
78BD
78BE
78BF
78C0
78C1
78C2
78C3
78C4
78C5
78C6
78C7
78C8
78C9
78CA
78CB
78CC
78CD
78CE
78CF
78D0
78D1
78D2
78D3
78D4
78D5
78D6
78D7
78D8
78D9
78DA
78DB
78DC
78DD
78DE
78DF
78E0
78E1
78E2
78E3
78E4
78E5
78E6
78E7
78E8
78E9
78EA
78EB
78EC
78ED
78EE
78EF
78F0
78F1
78F2
78F3
78F4
78F5
78F6
78F7
78F8
78F9
78FA
78FB
78FC
78FD
78FE
78FF
7900
7901
7902
7903
7904
7905
7906
7907
7908
7909
790A
790B
790C
790D
790E
790F
7910
7911
7912
7913
7914
7915
7916
7917
7918
7919
791A
791B
791C
791D
791E
791F
7920
7921
7922
7923
7924
7925
7926
7927
7928
7929
792A
792B
792C
792D
792E
792F
7930
7931
7932
7933
7934
7935
7936
7937
7938
7939
793A
793B
793C
793D
793E
793F
7940
7941
7942
7943
7944
7945
7946
7947
7948
7949
794A
794B
794C
794D
794E
794F
7950
7951
7952
7953
7954
7955
7956
7957
7958
7959
795A
795B
795C
795D
795E
795F
7960
7961
7962
7963
7964
7965
7966
7967
7968
7969
796A
796B
796C
796D
796E
796F
7970
7971
7972
7973
7974
7975
7976
7977
7978
7979
797A
797B
797C
797D
797E
797F
7980
7981
7982
7983
7984
7985
7986
7987
7988
7989
798A
798B
798C
798D
798E
798F
7990
7991
7992
7993
7994
7995
7996
7997
7998
7999

```

```

; Rutina MOTON
;
; Forneste motorul imprimantei.
; Distruge: A
; Registru utilizat:
; A - pentru transmiterea comenzii la port
;
; MOTON:
; LD A,MOTSTART ; comanda de pornire a motorului
; OUT (PRTPORT),A ; se trimite la port
; RET
;
; Rutina MOTOFF
;
; Asteapta ca semnalul de la limitatorul cursei capului de imprimare sa treaca
; din 0 in 1,dupa care opreste motorul imprimantei.
; Distruge: A
; Registru utilizat:
; A - pentru citirea portului SYSIN si transmiterea comenzii de oprire.

```

```

78B6 DB 90
78B7 CB 7F
78B8 28 FA
78B9 3E 00
78BA D3 B0
78BB C9
78BC
78BD
78BE
78BF
78C0
78C1
78C2
78C3
78C4
78C5
78C6
78C7
78C8
78C9
78CA
78CB
78CC
78CD
78CE
78CF
78D0
78D1
78D2
78D3
78D4
78D5
78D6
78D7
78D8
78D9
78DA
78DB
78DC
78DD
78DE
78DF
78E0
78E1
78E2
78E3
78E4
78E5
78E6
78E7
78E8
78E9
78EA
78EB
78EC
78ED
78EE
78EF
78F0
78F1
78F2
78F3
78F4
78F5
78F6
78F7
78F8
78F9
78FA
78FB
78FC
78FD
78FE
78FF
7900
7901
7902
7903
7904
7905
7906
7907
7908
7909
790A
790B
790C
790D
790E
790F
7910
7911
7912
7913
7914
7915
7916
7917
7918
7919
791A
791B
791C
791D
791E
791F
7920
7921
7922
7923
7924
7925
7926
7927
7928
7929
792A
792B
792C
792D
792E
792F
7930
7931
7932
7933
7934
7935
7936
7937
7938
7939
793A
793B
793C
793D
793E
793F
7940
7941
7942
7943
7944
7945
7946
7947
7948
7949
794A
794B
794C
794D
794E
794F
7950
7951
7952
7953
7954
7955
7956
7957
7958
7959
795A
795B
795C
795D
795E
795F
7960
7961
7962
7963
7964
7965
7966
7967
7968
7969
796A
796B
796C
796D
796E
796F
7970
7971
7972
7973
7974
7975
7976
7977
7978
7979
797A
797B
797C
797D
797E
797F
7980
7981
7982
7983
7984
7985
7986
7987
7988
7989
798A
798B
798C
798D
798E
798F
7990
7991
7992
7993
7994
7995
7996
7997
7998
7999

```



```

;
;
; Rutina INT (VCKB) 2b
;
; Deserveste cererile de intrerupere care sosesc cu o intermitenta de 2 ms.
; Scopul ei este de a balea dispozitivul de afisare ( imaginea afisata pe
; LED-uri trebuie sa fie oglinda codurilor continute in buferul de afisaj
; LEDBUF ).
; Registrii utilizati:
; B -- numarator circular cu valori intre 8 si 1
; C -- contine adresa portului de afisaj LEDPORT
; HL -- contine adresa curenta din LEDBUF
;
; INT:
; EX AF,AF ; se salveaza registrul A
; EXX ; si se comuta pe setul auxiliar de registrii
; OUTI ; se transfera la LEDPORT valoarea de la adre-
; AND ; sa curenta din LEDBUF si se decrementeaza
; LD A,(WITNESS) ; numarator circular
; AND MASK2 ; se ia valoarea curenta a portului sistem din
; OR B ; celula martor si se mascheaza, pentru a nu
; LD (WITNESS),A ; afecta decit bitii implicati
; OUT (SYSOUT),A ; se pozitioneaza bitii de selectie
; XOR A ; si noua valoare se salveaza in celula martor
; OR B ; valoarea creata se trimite la port
; JR NZ,INT1 ; se testeaza valoarea numaratorului circular
; LD HL,LEDBUF ; daca nu este 0, salt la sfirsit
; LD B,BUFLEN ; daca este 0, se reinitializeaza variabilele
; INT1: ; implicate
; EXX ; se revine la setul principal
; EX AF,AF ; se refaca A
; EI ; si se activeaza sistemul de intreruperi
; RETI
; PAGE

```

Tabela de texte MESSAGE

; ;
 ; Contine textele ce se tiparesc pe bonuri. Textele sint codificate in cod
 ; intern.

; MESSAGE:

78F9	81 08 20 1C	81H,08H,20H,1CH,18H,1CH	; TOTAL:
78FD	18 1C		
78FF	10 15 21 18	10H,15H,21H	
7902	82 04 20 1C	82H,04H,20H,1CH,18H,1CH	; TOTAL COD
7906	18 1C		
7908	10 15 20 11	10H,15H,20H,11H,18H,12H	
790C	18 12		
790E	83 02 2B 1C	83H,02H,2DH,1CH,18H,1CH	; TOTALUL ZILEI
7912	18 1C		
7914	10 15 1D 15	10H,15H,1DH,15H,20H,1FH	
7918	20 1F		
791A	14 15 13 14	14H,15H,13H,14H	
791E	84 01 20 1B 06	84H,01H,20H,1BH,14H,17H	; SINTEZA ! (rind gol)
7922	14 17		; VINZARILOR
7924	1C 13 1F 10 10 50 13	1CH,13H,1FH,10H	
7928	85 01 20 20 10	85H,01H,20H,20H,20H,20H	
792C	20 20 79 8C 0E 50 50		
792E	1E 14 17 1F 0W 0W	1EH,14H,17H,1FH,10H,1AH	
7932	10 1A 03 4C 0V 50 13		
7934	14 15 18 1A	14H,15H,18H,1AH	
7938	86 0F		; (total)
793A	87 0F		; (rind gol)
793C	88 0F		; (nr.bon casier)
793E	89 08 20 0D	89H,08H,20H,0DH,1EH,10H	; *VA MULTUMIM*
7942	1E 10		
7944	20 16 1D 15	20H,16H,1DH,15H,1CH,1DH	
7948	1C 1D		
794A	16 14 16 0D	16H,14H,16H,0DH	
794E	8A 0F		
7950	8B 0F 20 1D	8BH,0FH,20H,1DH,17H,14H	; (rind gol)
7954	17 14		; UNIT. NR.

MACRO-80 5.80 15-Jun-86

PAGE 1-72

Syntax:

7976	BE C5 C9 D1	80	BE	0 = 00H	0080	;	0 = 00H
7976	BE C5 C9 D1	80	BE	0 = 00H	0080	;	0 = 00H
7977	CO 0 FF C2	80	CO	1 = 01H	0080	;	1 = 01H
7977	CO 0 FF C2	80	CO	1 = 01H	0080	;	1 = 01H
7984	E6 C9 D1 D1	80	E6	2 = 02H	0080	;	2 = 02H
7984	E6 C9 D1 D1	80	E6	2 = 02H	0080	;	2 = 02H
7985	A2 D5 C9 C9	80	A2	3 = 03H	0080	;	3 = 03H
7985	A2 D5 C9 C9	80	A2	3 = 03H	0080	;	3 = 03H
7989	C1	80	C1	4 = 04H	0080	;	4 = 04H
7989	C1	80	C1	4 = 04H	0080	;	4 = 04H
798A	90 FF 92 94	80	90	5 = 05H	0080	;	5 = 05H
798A	90 FF 92 94	80	90	5 = 05H	0080	;	5 = 05H
798F	B9 C5 C5 C5	80	B9	6 = 06H	0080	;	6 = 06H
798F	B9 C5 C5 C5	80	B9	6 = 06H	0080	;	6 = 06H
7993	A7	80	A7	7 = 07H	0080	;	7 = 07H
7994	B1 CA CC C8	80	B1	8 = 08H	0080	;	8 = 08H
7994	B1 CA CC C8	80	B1	8 = 08H	0080	;	8 = 08H
7998	83 85 89 F1	80	83	9 = 09H	0080	;	9 = 09H
7998	83 85 89 F1	80	83	9 = 09H	0080	;	9 = 09H
7999	81	80	81	0 = 0AH	0080	;	0 = 0AH
7999	81	80	81	0 = 0AH	0080	;	0 = 0AH
799E	B6 C9 C9 C9	80	B6	1 = 10H	0080	;	1 = 10H
799E	B6 C9 C9 C9	80	B6	1 = 10H	0080	;	1 = 10H
79A2	B6	80	B6	2 = 10H	0080	;	2 = 10H
79A2	B6	80	B6	2 = 10H	0080	;	2 = 10H
79A3	86 89 99 A9	80	86	3 = 03H	0080	;	3 = 03H
79A3	86 89 99 A9	80	86	3 = 03H	0080	;	3 = 03H
79A7	C6	80	C6	4 = 04H	0080	;	4 = 04H
79A7	C6	80	C6	4 = 04H	0080	;	4 = 04H
79AB	80 80 CO 80	80	80	5 = 05H	0080	;	5 = 05H
79AB	80 80 CO 80	80	80	5 = 05H	0080	;	5 = 05H
79AC	80	80	80	6 = 06H	0080	;	6 = 06H
79AC	80	80	80	6 = 06H	0080	;	6 = 06H
79AD	88 88 BE 88	80	88	7 = 07H	0080	;	7 = 07H
79AD	88 88 BE 88	80	88	7 = 07H	0080	;	7 = 07H
79B2	88	80	88	8 = 08H	0080	;	8 = 08H
79B2	88	80	88	8 = 08H	0080	;	8 = 08H
79B6	94 88 BE 88	80	94	9 = 09H	0080	;	9 = 09H
79B6	94 88 BE 88	80	94	9 = 09H	0080	;	9 = 09H
79B8	94	80	94	0 = 0AH	0080	;	0 = 0AH
79B8	94	80	94	0 = 0AH	0080	;	0 = 0AH
79BC		80		1 = 10H	0080	;	1 = 10H
79BC		80		1 = 10H	0080	;	1 = 10H
79C1	FC 92 91 92	80	FC	2 = 10H	0080	;	2 = 10H
79C1	FC 92 91 92	80	FC	2 = 10H	0080	;	2 = 10H
79C6		80		3 = 03H	0080	;	3 = 03H
79C6		80		3 = 03H	0080	;	3 = 03H
79CA	A2 C1 C1 C1	80	A2	4 = 04H	0080	;	4 = 04H
79CA	A2 C1 C1 C1	80	A2	4 = 04H	0080	;	4 = 04H
79CB		80		5 = 05H	0080	;	5 = 05H
79CB		80		5 = 05H	0080	;	5 = 05H
79CF		80		6 = 06H	0080	;	6 = 06H
79CF		80		6 = 06H	0080	;	6 = 06H

Generatorul de caractere pentru imprimanta

PRIGEM:

DB	OBEH,OC5H,OC9H,OD1H,OBEN	;	0 = 00H
DB	OC0H,OC0H,OFFH,OC2H,OC4H	;	1 = 01H
DB	OC6H,OC9H,OD1H,OD1H,OE6H	;	2 = 02H
DB	OA2H,OD5H,OC9H,OC9H,OC1H	;	3 = 03H
DB	90H,OFFH,92H,94H,98H	;	4 = 04H
DB	OB9H,OC5H,OC5H,OC5H,OA7H	;	5 = 05H
DB	OB1H,OCAH,OCCH,OC8H,OB0H	;	6 = 06H
DB	83H,85H,89H,OF1H,81H	;	7 = 07H
DB	OB6H,OC9H,OC9H,OC9H,OB6H	;	8 = 08H
DB	86H,89H,99H,OA9H,OC6H	;	9 = 09H
DB	80H,80H,OC0H,80H,80H	;	0 = 0AH
DS	5	;	FUNC (nu se imprima)
DB	88H,88H,OB6H,88H,88H	;	+ = 0CH
DB	94H,88H,OB6H,88H,94H	;	* = 0DH
DS	5	;	TOTAL (nu se imprima)
DS	5	;	CLEAR (nu se imprima)
DB	OFCH,92H,91H,92H,OFCH	;	A = 10H
DB	OA2H,OC1H,OC1H,OC1H,OBEN	;	C = 11H

MACRO-80	5.80	15-Jun-86	PAGE	1-73
9C A2 C1 C1	DB	9CH,0A2H,0C1H,0C1H,OFFH		D = 12H
FF	DB	0C1H,0C9H,0C9H,0C9H,OFFH		E = 13H
C1 C9 C9 C9	DB	80H,0C1H,OFFH,0C1H,80H		I = 14H
FF	DB	0C0H,0C0H,0C0H,0C0H,OFFH		L = 15H
79DA	DB	OFFH,82H,8CH,82H,OFFH		M = 16H
79DE	DB	OFFH,0B0H,88H,84H,OFFH		N = 17H
80 C1 FF C1	DB	0BEH,0C1H,0C1H,0C1H,0BEH		O = 18H
80 C0 C0 C0	DB	86H,89H,89H,89H,OFFH		P = 19H
FF 82 8C 82	DB	0C6H,0A9H,99H,89H,OFFH		R = 1AH
FF	DB	0B2H,0C9H,0C9H,0C9H,0A6H		S = 18H
FF B0 88 84	DB	81H,81H,OFFH,81H,81H		T = 1CH
FF	DB	0BFH,0C0H,0C0H,0C0H,0BFH		U = 1DH
BE C1 C1 C1	DB	9FH,0A0H,0C0H,0A0H,9FH		V = 1EH
BE	DB	0C3H,0C5H,0C9H,0D1H,0E1H		Z = 1FH
86 89 89 89	DB	80H,80H,80H,80H,80H		= 20H
FF	DB	80H,80H,0A2H,80H,80H		= 21H
C6 A9 99 89	DB	88H,88H,88H,88H,88H		= 22H
FF				
B2 C9 C9 C9				
A6				
81 81 FF 81				
BF C0 C0 C0				
9F A0 C0 A0				
9F				
C3 C5 C9 D1				
E1				
80 80 80 80				
80				
80 80 A2 80				
30				
88 88 88 88				
88				

END

Macros:

Symbols:

ADDBCD	755D	ADDBYT	7569	ADD	7436	ADDSOR	743A
AFTERC	72EC	ANULAR	0010	ASTEPT	779E	ASTER	000D
BASEDE	0035	BCDC1	7666	BCDC11	7672	BCDLEN	0005
BEEP	77C3	BEPFI	0007	BEGPR	735C	BLANK	0020
BLKDIS	00FF	BTESTT	7148	BTWCHD	0005	BTWIDE	0080
BUFLEN	0008	BUYMAS	0008	BUYTIC	7145	CARLIM	0007
CASE	7178	CASETA	0086	CHANGE	768C	CLBUFD	771F
CLDISP	7735	CLEAR	000F	CLEARC	7744	CLKEYB	7726
CLRKN	6EBD	CLRKNL	0003	CNTFUN	7194	CODDIG	7376
CODE	6E57	CODED	72E9	CODELE	72F5	COLDST	70A1
COMP	7811	COMPUT	7805	CONTR	7098	CONTRL	0009
CORREC	73D4	CTC	00C0	CTCCMD	0097	CTCCOU	00FA
CTCPR0	705F	CTC	6E08	DATE	6FOA	DATEL	0008
DAYBCD	6C00	DAYMAS	0002	DECPAR	7624	DEL	78AB
DELAY	78A6	DELAYO	7817	DELAY1	781A	DELAY2	781C
DESKN	6F01	DESKNL	0002	DISP1	78FD	DISPHR	719C
DISPNU	76B3	DISPSU	76F3	DIVIDO	7828	DIVIDI	7832
DIVIDE	7821	DPNUM	768B	DPPOIN	76CF	DROPOU	78C9
EBCD	6E0D	EBCDBC	7650	EBCDIN	75E7	EBCDLE	000A
EDBFRE	03E8	EDBTIM	0014	EDBUF	6E39	EDLINE	74EF
EDLEN	0011	EMITBU	7458	EMITER	749F	ENDBUY	7157
ENDDP	76D4	ENDERA	7232	ENDERP	7357	ENDMOP	75E2
ENDNXP	736C	ENDOP	740E	ENDPP	732B	ENDST	76AF
ENDSYN	72DE	ENDTDA	72BC	ENDTRA	720C	ENDTSO	7281
EPERM	00AF	EPROM	700A	EPROMB	7000	EPROME	7016
ERAERR	7237	ERATIC	7219	ERPRIC	732C	ERRCYC	705B
ERSYN	72D6	ERRTDA	7280	ERRTSO	7275	ERSDFR	07D0
ERSDIM	0014	ETESTT	7223	EXPAND	774F	EXPCHR	775A
EXPLIN	7758	EXPSPA	7763	FCSDFR	08BB	FCSDTI	0014
FILLBY	774A	FORMME	7640	FUNC	000B	FUNCNU	0007
GUESTB	6DF9	HEADPO	7089	IMPLCO	739B	INCODE	7370
INERTD	0070	INITSO	7082	INKEY	7771	INPRIC	72E3
INT	78DC	INT1	78F4	INTMOD	706B	INTPAR	76D0
INTTAB	6FF8	INTVAR	7077	KEY1	0001	KEY2	0002
KEYBUF	6E29	KEYTAB	77B3	KYBDMA	000C	LAMPT	70CC
LEDBUF	6E31	LEDGEN	796C	LEDPOR	00A0	LMP	70D1
M10	75C8	MAINL0	712C	MASK1	000C	MASK2	00F8
MAXFUN	0002	MESSAG	78F9	MINUS	0022	MOTOFF	78B6
MOTON	78B1	MOTSTA	0080	MOTSTO	0000	MOVTMP	7705

MACR0-80 5.80

15-Jun-86

PAGE 5-1

MULT	75B0	MULTBC	758D	MULTEM	756B
NEXTDE	762A	NEXTED	750C	NEXTKE	7302
N0POIN	7208	NORMLI	7506	NORMSO	00BF
NKFUNC	6F2A	NKORPT	000B	NSORPT	7532
NXTLIN	778A	OMEHZ	61A8	OPERFR	03E8
OPERTM	0014	OPFORB	73AB	OTASTA	7799
PACKBY	7658	PARAMP	71C6	PCHAR	786F
PLUS	000C	POINT	000A	PRCDD	737F
PR0CCL	7173	PROCDI	721F	PR0CFU	715B
PR0DUC	73B7	PRPRIC	7318	PRT18C	7869
PRTCHA	787F	PRTCOL	7895	PRIGEM	7976
PRTLIN	783A	PRIPOR	00B0	PRITIC	74E6
RAMBYT	7021	RAMEME	00B7	RAMERH	2710
RAMERR	7031	RAMERT	00C8	RAMESD	7037
RAMLEN	0400	RAMRD	70BA	RAMT1	701B
RAMUR	70AA	RESDEL	0004	RESNEA	0080
SENDFR	0320	SEMDTM	0014	SETDEL	0001
SETUP	71B5	SETUPT	720D	SHIFTA	778D
SHIFTS	75A1	SHOPN	6EFO	SHOPNL	0003
SING	77E4	SING1	77E5	SORTAD	740F
SORTT0	6C05	SPEC	7501	SRCMBL	78C1
STACK2	6FF0	STACKR	6F28	START	7000
STORDI	76D9	STORED	7638	STOREI	76E0
STPOIN	76AA	STRITY	708F	SUBBCD	7575
SUBCOR	733E	SUBERR	7343	SUBSOR	7449
SYNIAN	75F2	SYNTH	72BD	SYNTMA	0001
SYNIN	0090	SYSOUT	0090	TAKEYB	71FA
TCL0CK	0190	TDECNP	763E	TENDP	730F
TESTFU	713E	TESTSU	733C	TICKNR	6EB3
IMPBCD	6E03	IMPMOV	7712	TOTAL	000E
T0TSOR	7244	TRAMLI	7519	TRANSP	71EB
WADR0U	7051	WAITPO	7841	WARMST	711B
WITHP	7630	WITNES	6F27	WORKBC	6DFE

No Fatal error(s)

Elaborarea unor materiale de sinteză, care să ajute reînțelegerea problemei de către aceeași persoană (peste ani de zile) sau de către altă persoană este o activitate care trebuie să însoțească elaborarea fiecărui pachet de software.

În acest capitol prezentăm următoarele materiale :

- Harta memoriei RAM
- Fluxul de date (aproximația finală)
- Nivelele ierarhice ale software-ului realizat
- Lista rutinelor încorporate în produs

Ne vom referi pe scurt la fiecare din ele.

18.1. Harta memoriei RAM

În memoria RAM (1 kbyte) se vor memora atât datele legate de vânzări, cât și variabilele implicite ale rutinelor elaborate precum și stiva de lucru a procesorului.

■ Rezervarea memoriei se face începînd de la adrese inferioare. Adresa inferioară a memoriei RAM considerată (6C00_H) o numim RAMBOT.

- DAYBCD (6C00) — registrul care conține suma vânzărilor (deci total de bani din casă) îl vom dispune, datorită importanței lui, în primele celule RAM.
- Urmează totalurile de vânzări aferente celor 100 de clase de sortimente (TOTSORT₀—TOTSORT₉₉: 6C05—6DF8), fiecărei sume rezervîndu-i-se cîte 5 octeți. Această tabelă ocupă aproape jumătate din memoria RAM disponibilă.
- GUESTBCD (6DF9) este registrul RAM în care se generează suma clientului ;
- WORKBCD (6DFE), TMPBCD (6E03) și DATBCD (6E08) sînt regiștrii de lucru ai aritmeticii BCD cu virgulă fixă și fără semn ;
- EBCD (6E0D) — este o zonă de 10 octeți în care se vor depune numerele normalizate, în format BCD extins.

- **PRTBUF (6E17—6E28)** este bufferul de ieșire pentru imprimantă ;
- **KEYBUF (6E29—6E30)** și **LEDBUF (6E31—6E38)** vor fi zonele de manevră pentru tastatură și dispozitivul de afișaj.
- **EDBUF (6E39—6F26)** — este bufferul de editare pentru imprimantă în care pe lângă cele 14 linii se disting următoarele puncte de intrare :
 - DATE : (6F0A)** — 8 byte — zonă pentru memorarea datei calendaristice
 - DESKN : (6F01)** — 2 byte — zonă pentru numărul casei de marcat
 - SHOPN : (6EFO)** — 3 byte — zonă pentru numărul magazinului
 - CLRKN : (6EBD)** — 3byte — zonă pentru identificatorul casierului

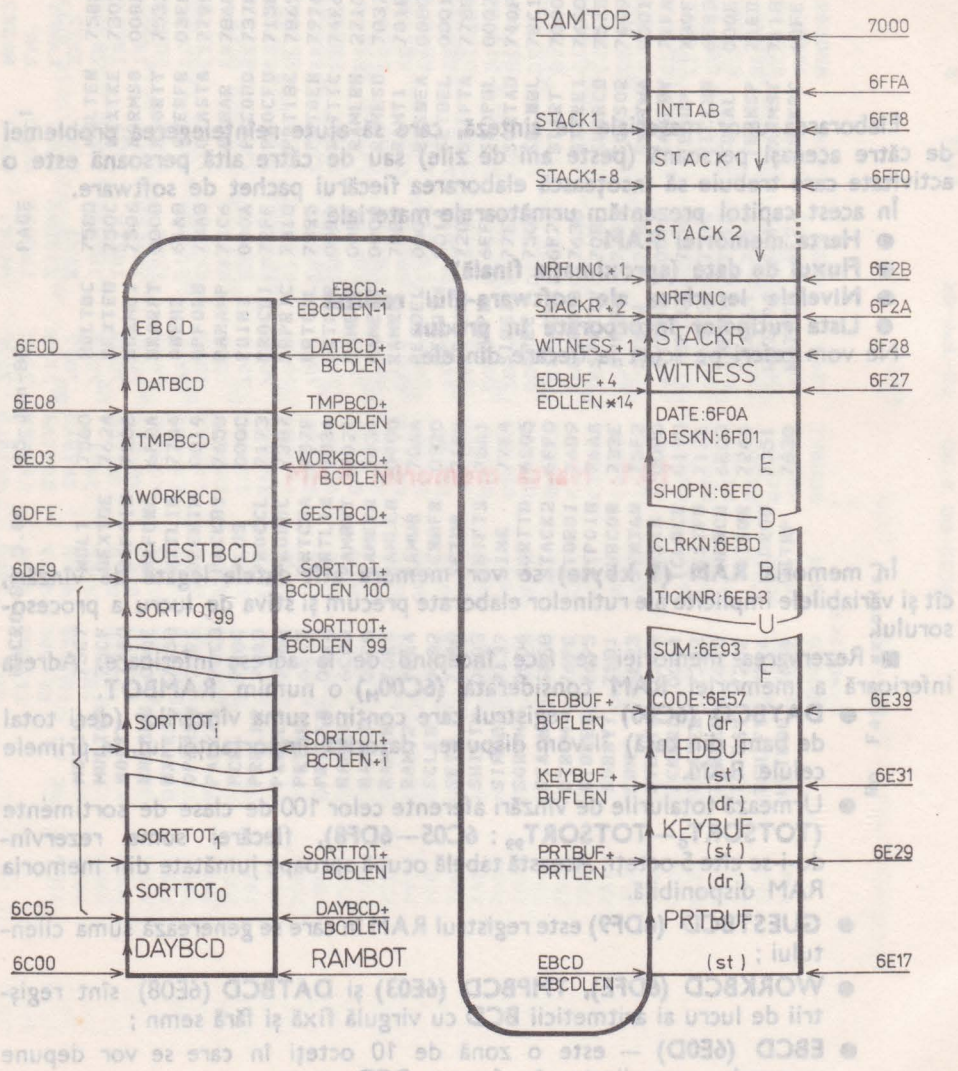


Fig. 18.1. Harta memoriei RAM (6C00—6FFF)

- TICKNR : (6EB3) — 4 byte — zonă pentru numărul curent de bor
 - SUM : (6E93) — 11 byte — zonă pentru sumă totală
 - CODE : (6E57) — 2 byte — zonă pentru cod de sortiment
 - WITNESS (6F27) este celula martor a portului de ieșire SYSOUT
 - STACKR (6F28 — 6F29) — sînt locațiile în care se salvează valoarea curentă a indicatorului de stivă, la apariția căderii de tensiune (DROPOUT)
 - NRFUNC (6F2B) — este o variabilă de lucru, care contorizează numărul de tastări FUNC (1 sau 2) în cazul bonurilor normale și a celor de anulare ;
- Elementele de stivă se definesc față de capătul superior al memoriei RAM (RAMTOP=6FFF), stiva fiind descrescătoare.
 - Zona (6FFF—6FF8) este rezervată pentru vectori de întrerupere. Avînd o singură sursă de întreruperi (circuitul CTC pentru afișaj) vom avea un singur vector : INTTAB (6FF8—6FF9).
 - Zona STACK1 (6FF8—6FF0) este stiva locală care va fi folosită de către secvența program cuprinsă între adresa 0 (reset) și punctul de ramificație STARTTYPE. Fiecare din cele două ramuri CSTART și WSTART care pornesc din acest punct, își vor reinițializa indicatorul de stivă : pornirea rece îl va inițializa la STACK2(6FF0), iar cea caldă după conținutul celei STACKR.
 - Zona STACK2(6FF0—6F2B) este stiva de lucru propriu-zisă folosită de casa de marcat. Dimensiunea ei CS_H va permite efectuarea a 62 de salvări succesive pe stivă, fără ca să pericliteze integritatea zonei de variabile a memoriei RAM.

18.2. Fluxul de date în casa de marcat (aproximația finală)

Schema din fig. 18.2. o completează pe cea din fig. 13.15 concretizînd numele unor activități și suplimentînd figura cu ramuri care nu le-am întrezărit la elaborarea proiectului.

18.3. Ierarhia modulelor de software elaborate

Mulțimea rutinelor cuprinse în listingul din Cap. 17 a fost împărțită în trei clase A, B, C.

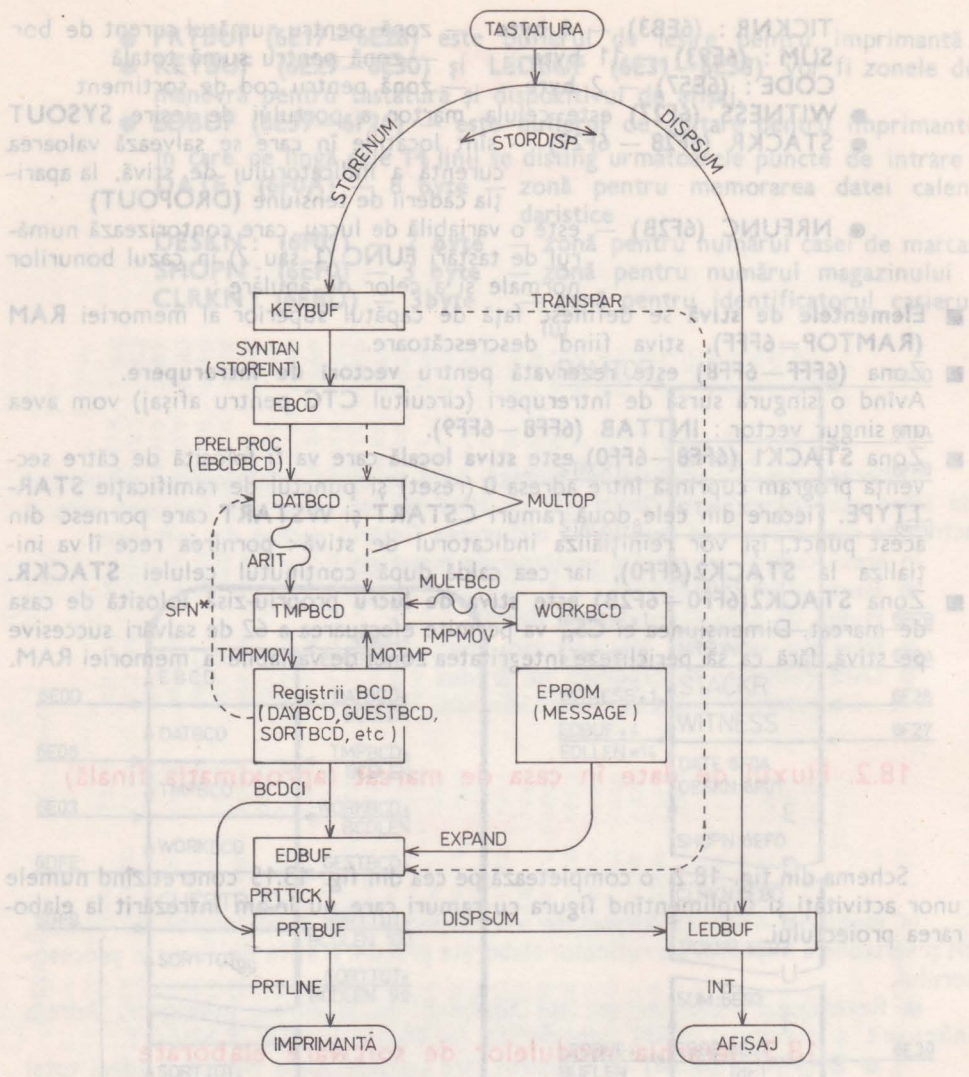
A — software pur, rezultat al specificației funcționale

B — secvență de inițializări și teste de trezire

C — rutine de tratare fizică a perifericelor (rutine „driver” sau „handler”)

Cele trei clase fiind indispensabile, le-am acordat aceeași prioritate, neîncercînd o interclasificare a lor. De aceea în fig. 18.3. le regăsim ca ramuri paralele. Referindu-ne în continuare la modulele ramurii A, specificăm criteriile care au stat la baza ierarhizării :

- în nivelul 0 am inclus modulele software care fac joncțiunea dintre specificația tehnică și programele care vor realiza aceste funcții impuse. Ele



SFN - Secvență fără nume

Fig. 18.2. Fluxul de date în casa de marcat (aproximația finală)

	B	A	C
Nivel 0	STARTTYPE CS*ART, WSTART, DROPOUT	MAINLOOP PROCFUNC, BUYTICK, ERATICK SETUP, TOTSORT, TODAY, SYNTH	INKEY, BEEP, PRTLINE, INT
Nivel 1	EPROMT, RAMT1, RAMT2 LAMPT	INPRICE, PRPRICE, ERPRICE NXPRICE, EMITBUYT, EMITERAT OPFORBUY, PRTTICK SORTADR, ADDSORT, SUBSORT INCODE, IMPLCODE, CNTFUNC	SING PRT18C PRTCHAR
Nivel 2	RAMINIT, INTMOD, CTC*PROG	SYNTAN, EBCBCD, BCD*CI ADDBCD, SUBBCD, MULTBCD EBCDINC, MUL10	COMPUTE PRTCOL DIVIDE COMP
Nivel 3	HEADPOZ	EXPAND, TRANSPAR, STORDISP DISPSUM, DISPNUM, STOREINT MOVTMP, TMPMOV, FILLBYTS STORENUM, CLBU*FDP, CLDISP	DELAYO DELAY MOTON MOTOFF

Fig. 18.3. Nivelele ierarhice ale software-ului realizat

crează de fapt cadrul pentru nivelele inferioare, fără să rezolve probleme concrete.

- în **nivelul 1** regăsim modulele care rezolvă *principal* sarcinile impuse prin specificația tehnică.
- în **nivelul 2** se găsesc rutinele de uz general, care sînt cvasiindependente de produs : aritmetică BCD și programele de conversie.
- ultimul nivel ierarhic (**nivelul 3**) cuprinde *module slave* de uz general, precum și cîteva excepții (**TRANSPAR,EXPAND**)

18.4. Lista rutinelor încorporate în produs

Ca un ultim element dedicat propriei noastre memorii elaborăm lista tuturor rutinelor din pachetul elaborat, specificînd în dreptul fiecăreia numele rutinelor pe care ea le apelează.

1. **START** (EPROMT, RAMT1, RAMT2, MOTON, MOTOFF, COLDSTRT, WSTART, LAMPT, RAMINIT)
2. **MAINLOOP** — INKEY, BUYTICK, PROCFUNC
3. **BUYTICK** — CLDISP, INPRICE, PRPRICE, NXPRICE, EMITBUYT
4. **PROCFUNC** — CLDISP, CNTFUNC, INKEY, DISPNUM, BUYTICK, SETUP, ERATICK, TOTSORT, TODAY, SYNTH
5. **CNTFUNC** — CLDISP, BEEP
6. **SETUP** — INKEY, CLBU*FDP, STORDISP, TRANSPAR, DISPNUM, BEEP
7. **TRANSPAR** —
8. **ERATICK** — INPRICE, ERPRICE, PRPRICE, EMITERAT, BEEP, DISPNUM
9. **TOTSORT** — INCODE, SORTADR, BCD*CI, DISPSUM, PRTTICK, BEEP, DISPNUM
10. **TODAY** — BCD*CI, DISPSUM, PRTTICK, BEEP, DISPNUM.
11. **SYNTH** — PRTTICK, BEEP, DISPNUM

12. **INPRICE** — INKEY, INCODE, IMPLCODE, STORDISP, CLBUFGP, MULTOP
13. **PRPRICE** — SYNTAN, PRELPROC, ADDSORT, OPFORBUY
14. **ERPRICE** — SYNTAN, PRELPROC, SUBSORT, OPFORBUY, BCDCI, DISPUM, BEEP
15. **NXPRICE** — INKEY, CNTFUNC, CLBUFGP
16. **INCODE** — INKEY, STORDISP, CLBUFGP
17. **IMPLCODE** — FILLBYTS
18. **OPFORBUY** — MOVTMP, ADDBCD, SUBBCD, TMPMOV, BCDCI, DISPUM, PRTLINE, SUBSORT
19. **SORTADR** — ADDD
20. **ADDD** —
21. **ADDSORT** — SORTADR, MOVTMP, ADDBCD, TMPMOV
22. **SUBSORT** — SORTADR, MOVTMP, SUBBCD, TMPMOV
23. **EMITBUYT** — EBCDINC, MOVTMP, ADDBCD, TMPMOV, BCDCI, DISPUM, PRTTICK
24. **EMITERAT** — EBCDINC, MOVTMP, SUBBCD, TMPMOV, BCDCI, DISPUM, PRTTICK, FILLBYTS
25. **PRTTICK** — SYNTTOTS, TRANLINE, PRTLINE, BEEP
26. **TRANLINE** —
27. **SYNTTOTS** — IMPLCODE, SORTADR, BCDCI, PRTLINE, EBCDINC
28. **ADDBCD** —
29. **SUBBCD** —
30. **MULTBCD** — TMPMOV, FILLBYTS, MUL10, ADDBCD
31. **MUL10** —
32. **MULTOP** — SYNTAN, EBCDBCD, MOVTMP, CLBUFGP
33. **EBCDINC** —
34. **SYNTAN** — FILLBYTS, SRCNBLK, STOREINT
35. **PRELPROC** — EBCDBCD, MULTBCD
36. **EBCDBCD** —
37. **BCDCI** — MOVTMP
38. **STORENUM** —
39. **DISPNUM** —
40. **STORDISP** — STORENUM, DISPNUM
41. **STOREIND** —
42. **DISPSUM** — SRCNBLK, DISPNUM
43. **MOVTMP** —
44. **TMPMOV** —
45. **CLBUFGP** — CLKEYBUF, CLDISP
46. **CLKEYBUF** — FILLBYTS
47. **CLDISP** — FILLBYTS
48. **CLEARCOD** — CLBUFGP
49. **FILLBYTS** —
50. **EXPAND** —
51. **INKEY** —
52. **BEEP** — COMPUTE, DIVIDE, SING
53. **SING DELAY** —
54. **COMPUTE** — DIVIDE, COMP
55. **COMP** —

- 56. DELAY0 —
- 57. DIVIDE —
- 58. PRTLINE — MOTON, DELAY, PRT18C, FILLBYTES, MOTOFF
- 59. PRT18C — PRTCHAR, DELAY
- 60. PRTCHAR — ADDD, PRTCOL
- 61. PRTCOL — DELAY
- 62. MOTON —
- 63. MOTOFF —
- 64. SRCNBLK —
- 65. DROPOUT —
- 66. INT —

Cartea de față fiind dedicată în totalitate instruirii, problemele și exercitiile...

Reamintim faptul că ambele calculatoare folosesc coduri ASCII, motiv pentru care EMKEY...

Duș cum s-a realizat programul de marcat alfabetic reprezentat o entitate de sine stătătoare...

Puteți vizualiza oricând secvențe ale acestor programe cu ajutorul simulatului...

Întrucât scopul acestui manual este să vă ajute să înțelegeți și să utilizați...

Întrucât scopul acestui manual este să vă ajute să înțelegeți și să utilizați...

ÎN LOC DE PROBLEME ȘI EXERCIȚII „Învierea” casei de marcat electronice pe calculatoarele PRAE și aMIC

Cartea de față fiind dedicată în totalitate instruirii, problemele și exercițiile nu pot lipsi. Ne vom abate — încă o dată — de pe „drumurile bătute”, enunțînd nu un șir de probleme, ci o temă de lucru. Finalizarea ei va oferi celor perseverenți satisfacția unei realizări palpabile.

19.1. Enunțul temei de lucru

După cum ați realizat probabil, casa de marcat elaborată reprezintă o entitate de sine stătătoare, dotată cu interfețe om-mașină proprii : tastatură, dispozitiv de afișaj cu LED-uri, imprimantă matricială.

Cu toate că proiectul software elaborat în capitolele precedente este complet și este înregistrat în totalitate pe caseta **Visible—Z80**, casa de marcat electronică nu va „învia” pe calculatoarele dumneavoastră. Motivul este cît se poate de simplu : rutinele de intrare/ieșire sînt specifice proiectului realizat și au fost elaborate pentru un hardware virtual, care diferă cert de cel al calculatoarelor **Prae** și **aMIC**.

Puteți vizualiza oricînd secvențe ale acestui software, cu ajutorul simulatorului grafic încorporat în **Visible—Z80**, folosind comanda — **G** urmată de o adresă aleasă după plac din listingul capitolului 17. Dar astfel veți putea vedea doar „copacii” nu și „pădurea”. Dacă doriți să vedeți casa de marcat „în acțiune”, sau intenționați să verificați exactitatea implementării, este necesar să modificați proiectul software realizat.

Iată deci enunțul temei de lucru :

Să se modifice software-ul, al cărui listing sursă se află în cap. 17, astfel încît el să funcționeze pe calculatorul personal **Prae** sau **aMIC**.

Sarcina vi se va părea simplă sau mai complicată, în funcție de nivelul dumneavoastră de cunoștințe software și în funcție de informațiile sistem pe care le posedați despre aceste calculatoare. Depinde de individualitatea fiecăruia în parte, dacă pornește la drum sau nu. Cert este că pentru începători aceasta este continuarea firească a procesului de asimilare, în materie de programare. Să schițăm pe scurt :

1. Încearcă să înțelegi programe existente, scrise de alții.

2. *Incearcă să modifice programele înțelese, folosind criteriile bine definite — exerciții impuse.*
3. *Incearcă să definești criteriile de modificare proprii și implementează-le — figuri liber alese.*
4. *Solicită teme de lucru din partea unor programatori consacrați, și rezolvă-le.*
5. *Incearcă să enunți teme de lucru și implementează-le.*

Începătorii care au parcurs cartea pînă la acest punct, se află la cea de-a doua etapă din lista schițată. Lor le vom elabora un ghid de lucru.

19.2. Ghid de lucru

19.2.1. Exerciții Impuse

■ a. Se va elabora o rutină, s-o numim **EMKEY (emulator de tastatură)**, menită să substituie rutina **INKEY**. Ea va citi tastatura calculatorului (**Præ** sau **aMIC**) și va preda în registrul **A** codul tastei apăsate, neafectînd alți regiștrii interni. Recomandăm folosirea *rutinelor de citire a tastaturii existente în EPROM-urile calculatoarelor amintite*.

● Reamintim faptul că ambele calculatoare folosesc coduri **ASCII**, motiv pentru care **EMKEY** va trebui să efectueze și o *transcodare*, din cod **ASCII** în codul intern, specific casei de marcat (vezi cap. 13). Ca metodă de transcodare vă recomandăm utilizarea unei *tabele a cărei intrare (adrese) o reprezintă codul ASCII și a cărei ieșire (date) vor fi codurile interne*.

● Știind că hardware-ul casei de marcat preconiza doar 16 taste, *recomandăm ca EMKEY să filtreze toate codurile de tastă care diferă de cele 16 definite: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ..+, *, F-func, C-clear și T-total*. Astfel revenirea din **EMKEY** se va face doar în momentul obținerii unui *cod intern valid*.

■ b. Se va elabora o rutină, fie ea **EMDP (emulator pentru dispozitivul de afișaj)**, care va substitui rutina **DISPNUM**.

● Definim pe ecran o *zonă pe care se vor afișa numerele pe care casa de marcat le-ar afișa pe dispozitivul de afișaj cu LED-uri*. Propunem ca zona respectivă să fie locată pe cel de-al doilea rînd (începînd din marginea inferioară a ecranului), centrat pe axa de simetrie verticală a imaginii.

● Recomandăm *utilizarea rutinelor de scriere pe ecran, rezidente în EPROM-urile calculatoarelor*.

Sarcinile pe care această rutină le va rezolva vor fi următoarele :

● să afișeze întocmai ca și proiectul inițial, *de la dreapta la stînga* ; în acest scop se vor folosi tehnicile de poziționare a cursorului, specifice fiecărui calculator (vezi par. 19.4).

● va trebui să efectueze o *transcodare din cod Intern în cod ASCII* utilizînd o metodă asemănătoare celei descrise la punctul a.

● înainte de a afișa un caracter (obligatoriu cifră sau punct zecimal), *rutina va deplasa conținutul zonei ecran cu o poziție la stînga*, cifra cea mai semnificativă ieșind din zona vizualizată (dacă numărul de cifre semnificative depășește numărul de 8) — ea va trebui să dispară.

■ c. Se va elabora o rutină, a cărei nume să fie **SIMPRT (simulator pentru imprimantă)**, care va substitui rutina **PRTLINE**.

● Întrucât imprimanta lipsește cu prisosință din majoritatea sistemelor realizate cu calculatoarele **Prae** și **aMIC**, ne propunem să o simulăm tot pe ecranul "full-graphic" al acestor calculatoare personale.

● Definim pe ecran o zonă (fereastră) de imprimare. Rezervăm în acest scop câte 18 caractere centrate pe axa verticală a imaginii, pe primele 24 rânduri (numărate din extrema superioară).

Funcțiile pe care **SIMPRT** le are de efectuat sînt analoge cu cele ale rutinei **PRTLINE**, pe care ea o va substitui. Să le recapitulăm :

● la apelare, ea va transfera conținutul zonei tampon pentru imprimantă (**PRTBUF**) pe rîndul inferior al zonei de imprimantă, rezervată pe ecran (24 rînduri \times 18 coloane) ;

● reținem faptul că **PRTBUF** conține informațiile codificate în cod intern, iar rutinele de afișare pe ecran ale calculatoarelor **Prae** și **aMIC** solicită coduri **ASCII** ; se va efectua deci o transcodare cod intern \rightarrow cod **ASCII**, folosind tabela definită la punctul b. ; recomandăm să folosiți și în această tabelă un cod rezervat (ex. **FF_H**) pentru marcarea caracterelor nedefinite în proiectul de casă de marcat ; dacă rutina primește — printr-o întîmplare — un cod nepermis (**FF_H**), atunci ea îl va filtra, netrasmițîndul spre afișare ;

● înainte de a „imprima” un rînd (pe rîndul inferior al zonei ecran de 24 \times 18), conținutul zonei de imprimantă se va muta în sus (defilare) cu un rînd, și se va șterge (afișînd 18 blancuri) rîndul inferior al zonei, rînd pe care urmează să se facă imprimarea ;

● după afișarea rîndului dorit, **SIMPRT** va șterge conținutul zonei **PRTBUF**, aidoma rutinei **PRTLINE**.

■ d. Se va elabora o rutină **EMCLDP (emularea ștergerii dispozitivului de afișaj)**, care va substitui rutina **CLDISP**.

● Întrucît rutina **CLDISP** „șterge” dispozitivul de afișaj în mod nemijlocit, apelînd rutina **FILLBYTES**, **EMCLDP** va trebui să „umple” cu blancuri (spații) zona ecran care este menită să înlocuiască dispozitivul de afișaj cu **LED**-uri.

■ e. Se va modifica rutina **CNTFUNC**, astfel încît ea să afișeze valoarea număratorului de tastări succesive **FUNC** pe ecran, în extremitatea stîngă a zonei de afișaj (definită la punctul b.)

● Recomandăm utilizarea rutinelor de scriere pe ecran, rezidente, precum și folosirea tehnicilor de adresare a cursorului.

■ f. Se va elimina rutina **BEEP**, pentru a evita îngreunarea implementării modificărilor. (Rutina **BEEP** acționează portul de ieșire **SYSOUT**. Emițînd valori neadecvate pe porturile sistem ale calculatoarelor **Prae** și **aMIC**, se pot întîmpla lucruri absolut imprevizibile). Implementarea acestei rutine va fi făcută de cei mai ambițioși, doar în momentul în care ei vor cunoaște bine structura hardware a calculatorului pe care lucrează. Eliminarea rutinei **BEEP** se face substituind primul octet al rutinei cu codul instrucțiunii **RET (C9_H)**.

■ g. Se analizează secvența de trezire a casei de marcat, eliminînd (prin substituire cu **NOP**-uri **00_H**) toate instrucțiunile sau secvențele care ar putea deranja funcționarea calculatorului (ex. testele de trezire sînt inutile, fiindcă calculatorul este oricum trezit în momentul încărcării programului **Visible—Z80** ; se elimină referințele la sistemul de întrepreri **EI,IM x** ; se elimină inițializarea portului **SYSOUT** ; etc.).

■ Vom considera remanierile drept bune dacă comanda **C7000,712C(CR)** lansată din monitorul rezident al lui **Visible—Z80** se execută fără cusur, făcându-se revenirea în monitor odată cu atingerea adresei de breakpoint (**712C_H**). Amintim faptul că nu se vor elimina secvențele de inițializare ale variabilelor definite în proiectul de casă de marcat.

■ Programele nou elaborate se vor asambla în spațiul de adrese **6800—6DFF_H**.

■ În corpul inițial al programului de casă de marcat cuprins în spațiul de adrese **7000—7A24_H** se vor efectua substituirile necesare, pentru a devia salturile: din rutinele înlocuite, la rutinele înlocuitoare. În fiecare rutină înlocuită se vor modifica primii trei octeți, care vor fi înlocuiți cu câte o instrucțiune de salt necondiționat la începutul rutinei înlocuitoare.

■ Vă recomandăm să salvați programul modificat, pe casetă magnetică, sub forma unei înregistrări de sine stătătoare, înregistrând conținutul zonei de memorie: **6800_H—7FFF_H**.

Dacă rutinele elaborate și modificările făcute sînt corecte, atunci la comanda **C7000(CR)**, pe ecranul calculatorului dumneavoastră va „învia” casa de marcat electronică, proiectată în partea a III-a a cărții de față. Răbdare și perseverență.

Ajunși la sfîrșitul noului proiect, constatăm că software-ul inițial al casei de marcat a trecut destul de bine „proba de foc” a modificărilor, cerință considerată drept esențială (vezi experiențele formulate în cap. 10).

Trebuie totuși să recunoaștem că s-ar fi putut și mai bine. Într-un caz ideal, în corpul programului ar fi trebuit să facă doar cinci modificări (INKEY, DISPNUM, PRTLIN, BEEP, INIT) față de cele șapte enunțate mai sus.

Asta e! ce să-i facem? N-am rezistat ispitei de a trata dispozitivul de afișaj în mod neconvențional, renunțînd la serviciile rutinei DISPNUM în două cazuri: CLDISP, CNTFUNC. Atenție, nu e bine să alegeți calea cea mai ușoară!

19.2.2. Figuri „liber alese”

Cei care au elaborat lucrarea enunțată la paragraful 19.1. și definită la paragraful 19.2.1. vor putea să treacă mai departe dînd frîu liber imaginației lor. aducînd noi modificări proiectului de casă de marcat. Nu vrem să vă influențăm, exercițiile fiind „liber alese”, dar amintim cîteva posibilități intuite de noi:

■ dispozitivul de afișaj emulat pe ecran, să afișeze caractere similare cu cele pe care le-ar vizualiza afișorul cu LED-uri (cifre formate din 7 segmente — și nu caracterele proprii ale calculatoarelor Prae și aMIC);

■ în zona de imprimare rezervată pe ecran, să apară nu caracterele proprii ale calculatorului, ci cele definite în PRTGEN;

■ „imprimarea” pe ecran să fie asortată de un zgomot specific imprimantelor matriciale cu ace, zgomot realizat pe cale software; etc.

Cu cît proiectul rezultat va aproxima mai bine casa de marcat virtuală, definită în capitolul 11., cu atît emularea va fi mai perfectă.

Cei care au trecut prin etapele definite la paragraful 19.2. pot acum continua procesul de perfecționare în domeniul programării, continuînd de la punctul 4 al îndrumarului din paragraful 19.

Pe paginile următoare redăm lista codurilor ASCII, și cea a rutinelor uzuale ale calculatoarelor Prae și aMIC, venind astfel în ajutorul celor care își propun elaborarea lucrării enunțate în paragraful 19.1.

19.3. Tabela codurilor ASCII

Tab. 19.1. Codurile ASCII

CODURILE ASCII

HEX	MSD	0	1	2	3	4	5	6	7
LSD	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	-	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

NUL — Null

SOH — Start of Heading

STX — Start of Text

ETX — End of Text

EOT — End of Transmission

ENQ — Enquiry

ACK — Acknowledge

BEL — Bell

BS — Backspace

HT — Horizontal Tabulation

LF — Line Feed

VT — Vertical Tabulation

FF — Form Feed

CR — Carriage Return

SO — Shift Out

SI — Shift In

DLE — Data Link Escape

DC — Device Control

NAK — Negative Acknowledge

SYN — Synchronous idle

ETB — End of Transmission Block

CAN — Cancel

EM — End of Medium

SUB — Substitute

ESC — Escape

FS — File Separator

GS — Group Separator

RS — Record Separator

US — Unit Separator

SP — Space (Blank)

DEL — Delete

MSD reprezintă cifra cea mai semnificativă (Most Significant Digit — D6—D4) iar LSD cifra cea mai puțin semnificativă (D3—D0) a codului ASCII.

19.4. 'Lista rutinelor uzuale ale microcalculatoarelor PRAE și aMIC

În cei cîte 16 cocteți de memorie fixă a calculatoarelor PRAE și aMIC rezidă cîte un program monitor și cîte un interpretor BASIC. Aceste pachete software conțin multe rutine care pot fi folosite de către utilizatori, prin apelarea lor din programe proprii, scrise în limbaj de asamblare.

Redăm lista rutinelor importante :

PRAE

CI — **Consol Input.** Citește un caracter de la consola selectată (tastatura proprie sau interfața serială). Codul caracterului citit se găsește în registrul **A**.
3151_H

CO — **Consol Output.** Transmite caracterul al cărui cod se află în registrul **C** la consola selectată.
3154_H

CSTS — **Consol Status.** Verifică starea consolei de intrare selectate și remite în **A** — FF_H dacă nu s-a apăsat nici o tastă, sau codul caracterului tastat.
3169_H

SI — **Serial Input.** Citește un caracter de pe interfața serială. Codul caracterului citit se găsește în registrul **A**.
317B_H

SO — **Serial Output.** Transmite caracterul al cărui cod se află în registrul **C**, la interfața serială.
316C_H

KYI — **Keyboard Input.** Citește un caracter de la tastatura proprie. Codul caracterului citit se găsește în registrul **A**.
318D_H

CRTO — **CRT Output.** Afișează pe ecran pe poziția indicată de variabila **PNT** caracterul al cărui cod se află în registrul **C**. Avansează cursorul cu o poziție la dreapta.
3190_H

KSTS — **Keyboard Status.** Rutina are acțiune similară cu **CSTS**, dar se referă doar la tastatura proprie.
31A2_H

SSTS — **Serial Status.** Rutină similară cu **KSTS**, dar se referă la interfața serială.
319F_H

CASOUT — Înregistrează pe caseta magnetică un bloc de date aflat în bufferul de casetă 4080_H—40FF_H.
3187_H

CASIN — Citește de pe casetă magnetică un bloc de date și îl depune în bufferul de casetă 4080_H—40FF_H.
318A_H

RAMT — Test nedistructiv pentru memoria **RAM**. Afișează adresa primei celule din care nu s-a putut citi corect valoarea înscrisă.
31AE_H

EPROMT — Test pentru memoria **EPROM**. Afișează un mesaj de eroare și numărul **EPROM**-ului (0, 7) găsit a fi eronat.
31A8_H

CRLF — Transmite la consola de ieșire selectată, caracterele **CR** (0D_H) și **LF** (0A_H) (Retur de car și avans de linie).
01F2_H

BLK — Transmite la consola de ieșire selectată, caracterul spațiu (20_H).
01E5_H

LBYTE — Listează la consola de ieșire selectată, valoarea hexazecimală a octetului cuprins în registrul **A**.
024C_H

LADR — Listează la consola de ieșire selectată, valoarea hexazecimală a celor doi cocteți aflați în registrul dublu **HL**.
0247_H

Variabile sistem importante.

BEG : 4004_H—4005_H. Conține adresa de început ecran pentru rutinele de tipărire (E000_H la PRAE 48K și 6000_H la PRAE 16K).

BEGPLT : 401A—401B_H. Conține adresa de început ecran pentru rutinele de grafică.

FIN : 4008—4009_H. Conține adresa primului octet după sfârșitul memoriei de imagine (0000_H la **PRAE 48K** și 8000 la **PRAE 16K**).

PNT : 4006—4007_H. Adresa curentă a cursorului de pe ecran. Utilizatorul îi poate acorda valori în gama E000_H—FF00_H la **PRAE 48K** și 6000—7F00_H la **PRAE 16K**.

UPAD : 4010_H Parametrii interfeței seriale.

+0 **Martorul** portului de ieșire sistem

+1 **Adresa** portului de ieșire (80_H).

+2 **Număr de biți utili** ai cuvântului transferat.

+3 **Viteza de transfer** : FC_H—300 baud, 7E_H—600 baud, etc.

+4 **Tipul de paritate** : b₀ = 0 fără paritate ; b₀ = 1 și b₁ = 0 paritate pară ; b₀ = 1 și b₁ = 1 paritate impară.

+5 **Numărul biților de stop** (1 sau 2).

DENSIT : 4018_H Viteza de înregistrare a datelor pe caseta magnetică. Poate varia în limite largi 6—255, viteza de transfer fiind invers proporțională cu acest număr. La trezire se inițializează la 0A_H (2400 bit/s).

aMIC

a) Pentru calculatoarele echipate cu monitorul VO.1.

CIN — **Consol Input**. Citește un caracter de la tastatură și îl furnizează în A.

COUT — **Consol Output**. Trimite la display caracterul conținut de registrul C (codul ASCII și îl afișează în poziția curentă a cursorului de pe ecran).

KIN — **Cassette Input**. Citește de pe casetă un fișier în memoria micro-calculatorului, la adresa de la care a fost salvat.

KOUT — **Cassette Output**. Înscrie pe casetă un fișier din memoria calculatorului. La apelare HL va conține adresa de început, iar DE lungimea zonei de salvat.

Variabile sistem importante.

6000_H : Numărul rîndului alfanumeric, în care se află poziționat cursorul pe ecran. Va fi cuprins în limitele 00—1F_H, 00 corespunzând primului rînd de pe ecran.

6001_H : Numărul coloanei alfanumerice, în care se află poziționat cursorul pe ecran. Valoarea va fi cuprinsă în limitele 00—1D_H, 00 corespunzând primei coloane, din stînga ecranului.

b) Pentru calculatoarele echipate cu Monitor Z80 VO.0

Față de cele de la pct. a) apar diferențe de adresă la rutinele de casetă :

KIN — 3C1C_H

KOUT — 3BAE_H — parametrii de apel : **HL** — adresă de început
BC — număr total de octeți
DE — număr fișier (4 cifre hexa).

c) Pentru calculatoarele echipate cu MON. aMIC VO.3 sau monitorul DEST.

Toate rutinele se apelează cu **CALL 0005_H**.

Octetul conținut în registrul C va defini funcția de efectuat, iar D și E conțin eventualii parametri.

- C=0** : **RESET**. Ecranul este șters, variabilele cuprinse în zona 6000_H — $60FF_H$ sînt inițializate cu 0.
- C=1** : **CONIN**. Citește caracter de la consolă în registrul **A**.
- C=2** : **CONOUT**. Scrie caracterul din registrul **E** la consolă.
- C=3** : **RIN**. Citește caracterul de pe interfața serială (în **A**).
- C=4** : **POUT**. Scrie caracter la interfața serială (din **E**).
- C=5** : **LOUT**. Imprimă caracter pe miniimprimantă (din **E**).
- C=9** : **WSTRIN**. Scrie șir de caractere la consolă. (DE adresă șir de caractere, care se termină cu „\$” sau 00_H).
- C=0A_H** : **RSTRIN**. Citește și editează buffer consolă (DE adresă buffer).
- C=0B_H** : **CSTS**. Citește starea consolei. (Dacă s-a tastat caracter atunci $A=FF_H$, dacă nu atunci $A=00_H$).

1. ...

2. ...

3. ...

4. ...

5. ...

6. ...

7. ...

8. ...

9. ...

10. ...

11. DILKSTRAA, E. W. A Discipline of Programming. Prentice Hall, New Jersey, 1978.

12. WEINBERG, G. M. The Psychology of Computer Programming. Van Nostrand Reinhold Co, New York, 1975.

13. YOUNDON, E. N. Techniques of Program Structure and Design. Prentice Hall, New Jersey, 1978.

14. YOUNDON, E. N. Managing the Structured Techniques. Youdon Press, New York, 1978.

15. KNUTH, D. E. The Art of Computer Programming, vol. I — Fundamentals Algorithms Addison-Wesley Reading (Mass.), 1973.

16. KERNIGHAN, R. — PLÄNGER, P. Software Tools. Addison-Wesley Reading (Mass.), 1978.

17. PATRUBĂNY, M. Mikroproceszorok rev. Korunk Fuzetek nr. 2. Csi-Napoca, 1983.

18. KNUTH, D. E. Tratat de programare a calculatoarelor I—III. Editura Tehnică, București, ed. 1978—83.

19. GOLDSTEIN, H. H. Neumann seropce a számítástechnikában (in Neumann János élete és munkássága), Budapest, 1978.

20. PETRESCU, A. etc. Totul despre... calculatoarele personale. Editura Tehnică, București, 1983.

21. JENSEN, K. — WIRTH, N. Pascal, User Manual and Report. Lecture Notes in Computer Science nr. 18. Springer Verlag, Berlin, 1976.

22. GOCINILE, H. ELIS, P. et al. Limbajele de programare Pascal și Pascal concurrent. Editura Facultății de Informatică, Timișoara, 1983.

23. MAKARA ERNŐ. Tervezési segédlet a Z80 (MK380) típusú P mikrovezérlőhöz. I. rész. Informatikai Közvetítő Budapest, 1982.

24. ...

25. ...

26. ...

27. ...

Bibliografie

1. LUPU, CR., STĂNESCU, ST. Microprocesoare. Circuite. Proiectare, Editura Militară, București, 1986
2. LUPU, CR., etc. Microprocesoare. Aplicații. Editura Militară, București, 1982.
3. KRIZSÁN, GYÖRGY, A ZILOG cég mikroprocesszor családjai — vol. I — LSI OMIKK Budapest, 1984
4. BALTAC, VASILE Optimizarea sistemelor de operare ale calculatoarelor numerice, Editura Facla, Timișoara, 1974
5. MUNTEANU, EMIL, COSTEA V., MITROV, M. Programarea în limbaje de asamblare ASSIRIS, Editura Tehnică, București, 1976
6. MUNTEANU, EMIL Utilizarea calculatoarelor în prelucrarea datelor. Sistemul de operare, vol. I., Editura Dacia, Cluj-Napoca, 1974
7. BALTAC, V., ROMAN D. Calculatoarele electronice, grafica interactivă și prelucrarea imaginilor, Editura Tehnică, București, 1985
8. CĂPĂȚINA DAN, etc. Proiectarea cu microprocesoare, Editura Dacia, Cluj-Napoca, 1983
9. MUREȘAN, T., etc. Microprocesorul 8080 în aplicații, Editura Facla Timișoara, 1981
10. TOACȘE, GH. Introducere în microprocesoare, Editura Științifică și Enciclopedică, București, 1985.
11. PÉTERSON, J.L. Computer Organization and Assembler Language Programming, Academic Press, New York, 1978.
12. LESEA A. — ZAKS, R. Technique d'interface aux microprocesseurs, Sybex Inc. Paris, 1979
13. ZAKS, R. Programming the Z80, Sybex. Inc. Berkeley, 1982
14. WILLIAMS, S. Programming the 68000, Sybex, Inc. Berkeley, 1985
15. COFFRON, J. W. Programming the 8086, 8088, Sybex. Inc. Berkeley, 1983
16. BROOKS, F. P. The Mythical Man Month., Addison—Wesley Reading (Mass.), 1979
17. DIJKSTRAA, E. W. A Discipline of Programming. Prentice Hall, New Jersey, 1976
18. WEINBERG, G. M. The Psychology of Computer Programming, Van Nostrand Reinhold Co, New York, 1975
19. YOURDON, E. N. Techniques of Program Structure and Design, Prentice Hall New Jersey, 1975
20. YOURDON, E. N. Managing the Structured Techniques, Yourdon Press, New York, 1979
21. KNUTH, D. E. The Art of Computer Programming. vol. I — Fundamental Algorithms. Addison—Wesley Reading (Mass.), 1973
22. KERNIGHAN, B. — PLANGER, P. Software Tools. Addison—Wesley Reading (Mass.), 1976
23. PATRUBĂNY, M. Mikroprocesszorok, rev. Korunk Füzetek nr. 2, Cluj-Napoca, 1983
24. KNUTH D. E. Tratat de programare a calculatoarelor I—III, Editura Tehnică, București, ed. 1978—85.
25. GOLDSTINE, H. H. Neumann szerepe a számítástechnikában (in Neumann János élete és munkássága), Budapest, 1979
26. PETRESCU, A. etc. Totul despre... calculatorul personal aMIC. Editura Tehnică București, 1985
27. JENSEN, K. — WIRTH, N. Pascal, User Manual and Report. Lecture Notes in Computer Science nr. 18, Springer Verlag, Berlin, 1976.
28. CIOCĂRLIE, H., ELES P. ș.a. Limbajele de programare Pascal și Pascal concurrent. Editura Facla Timișoara, 1985
29. MAKARA ERNŐNÉ Tervezési segédlet a Z80 (MK3880) típusú μP alkalmazásához. I rész Ipari Informatikai Központ Budapest. 1982
30. *** Z80. Technical Manual. Mostek, 1979
31. *** Data Catalog. SGS ATEs, 1982

- 1— 2. L. Dumitrașcu
- 3— 4. A. Petrescu și colectiv IPB,
ITCI, Fabrica de calculatoare,
Liceul Dimitrie Cantemir,
CNOP
- 5— 6. A. Tănăsescu și colectiv IPB,
ISPIF
- 7—12. Colective largi
13. A. Davidoviciu și colectiv
ITCI, ASE
- 14—15. I. Văduva, V. Baltac,
Florescu V și colectiv ASE, ITCI
16. Rusu O., Brudaru I.
17. P. Constantinescu

Invățăm microelectronică interactivă
conversînd în BASIC. Totul despre...
BASIC în 14 conversații și 7 sinteze
pe Felix C, CORAL, INDEPENDENT,
Felix PC, M118, TPD, HC 85, aMIC,
PRAE, COMMOORE, AMSTRAD și
compatibile, vol. 1 și 2.

ABC de calcul electronic. Totul des-
pre... HC 85, vol. 1, și vol. 2, (o parte
a tirajului cu 2 casete cu programe,
acționînd calculatoare personale HC 85
și compatibile SINCLAIR SPECTRUM)

Grafică asistată de calculator: Programe
Fortran pe minicalculatoare, pentru re-
prezentări geometrice, vol. 1 și vol. 2.

Automatică, management, calculatoare
(AMC). Serie continuă de instruire, in-
formare, sinteze, cercetări aplicative în
sisteme electronice, automate, informa-
tice, de conducere, volumele 56—51.
Echipamente electronice și tehnică de
calcul — manuale de utilizare. Calcu-
latoare personale, programe ș.a.

Sistemul de operare MIX și limbajul
MACRO pentru minicalculatoare CO-
RAL/INDEPENDENT, 2 volume.

Informatică economică, 2 volume

Echilibrarea liniilor flexibile.

Sinergia, informația și geneza sistemelor.

Se difuzează prin unitățile centrelor de librării, spre care se îndrumă întreprin-
derile și cititorii.

**PENTRU ACESTE CARTI SE POT FACE, TOTUȘI, ȘI COMENZI FERME LA
EDITURA TEHNICĂ, PIAȚA SCINTEII 1, BUCUREȘTI.**

Comenzile întreprinderilor se semnează de director și contabil șef, cele ale citi-
torilor individuali au indicată adresa exactă. Comenzile se trimit de editură la cen-
trele de librării, cu indicarea unor priorități de satisfacere a lor. Plata nu se face
decît la primirea exemplarelor.

● În prima parte a volumului 1 se prezintă exhaustiv structura și funcționarea amănunțită a microprocesorului, iar în partea a doua sunt grupate: manualul de utilizare a casetei, comparația între limbajele de asamblare ale celor două microprocesoare uzuale (Z 80 și I 8080), fișele detaliate ale tuturor instrucțiunilor microprocesorului Z 80, pe clase și grupe, lista instrucțiunilor publicate de firmă și a celor descoperite de utilizatori ș.a.

● Volumul 2 este constituit din proiectarea completă a unei case de marcat electronice cu microprocesor Z 80, pornind de la principiile și metodele proiectării-programării structurate în limbaj de asamblare, până la listing-ul amplu comentat al tuturor modulelor proiectului.

● Ultimul capitol este „o provocare” și totodată un test: cititorii cărții ghidați de indicațiile autorului sunt invitați să „invie” casa de marcat pe ecranul televizorului.

● În esență, o carte originală, de largă respirație, scrisă de un reputat specialist, o ediție complexă (structură, casetă, culoare ș.a.), care deschide în Biblioteca de Automatică, Informatică, Electronică, Management (BAIEM), ciclul de produse-program, ce va continua curînd.

ISBN 973-31-0009-9

ISBN 973-31-0007-2

