

Octavian G. Mustafa

# Easy6502

Asamblare și execuție într-un emulator  
scris în JavaScript  
de Stian Søreng și Nick Morgan



Publicațiile DAL  
Craiova

Fișier prelucrat în data de [February 3, 2025]



## Avertisment

Acest eseu nu a fost raportat vreunui referent. În consecință, conținutul său trebuie considerat “ca atare.”

Autorul vă așteaptă comentariile la adresa lui de e-mail<sup>1</sup> și vă mulțumește anticipat pentru efortul depus.

Fiecare proiect de la *Publicațiile DAL* trebuie considerat “șantier” dacă nu este declarat altfel. Versiunea sa este cea a datei de pe pagina cu titlul.

Craiova, Mai 18, 2015

*O.G.M.*

---

<sup>1</sup> octavian@yahoo.com



# Prefață

Oricine petrece suficient de mult timp scriind programe pentru calculatorul personal află de la cunoscuți că anumite părți din acestea ar putea fi îmbunătățite dacă cererile cuprinse în ele s-ar adresa *direct* microprocesorului, adică *dacă ar fi rescrise în limbaj de asamblare*.

```
Asamblează Execută Resetează HexDump Dezasamblează Informații/Șterge

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; dupa Stian Soreng, www.6502asm.com ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

buc1a: lda $fe      ; A=aleatoriu
      sta $00      ; ZP(0)=A
      lda $fe
      and #$3      ; A=A&3
      clc          ; resetez fanionul de transport
      adc #$2      ; A+=2
      sta $01      ; ZP(1)=A
      lda $fe      ; A=rnd
      ldy #$0      ; Y=0
      sta ($00),y  ; ZP(0),ZP(1)=y
      jmp buc1a
```

**Fig. 0.1** Editorul de text.

Cărțile privitoare la o astfel de programare *optimizată* sunt numeroase însă dificil de parcurs: încercând să explice cum se petrec lucrurile *sub capota* unui calculator

performant, de *gaming*, ele nu se grăbesc cu avertismente de genul *Ai grijă, registrele de memorie nu se găsesc în același loc cu stiva de memorie* ș.a.m.d.

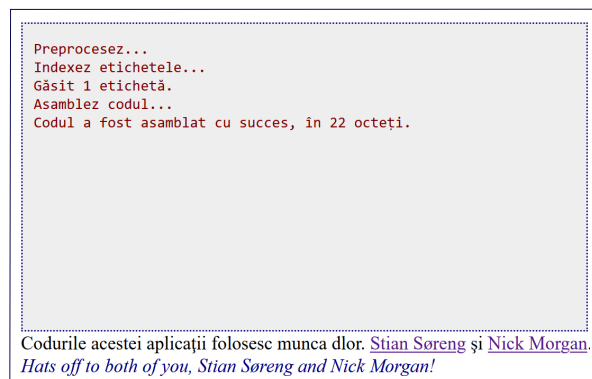


**Fig. 0.2** Panoul de mesaje.

Ca să înțeleagă câte ceva, programatorul în chestiune are nevoie de *arheologie*: întâi caută un microprocesor *antic* — a se citi *simplu* —, apoi se documentează din manualele timpului și... Din păcate, asemenea microprocesoare necesită *periferice* pe măsură: memorii, terminale grafice, tastaturi, imprimante. De negăsit!

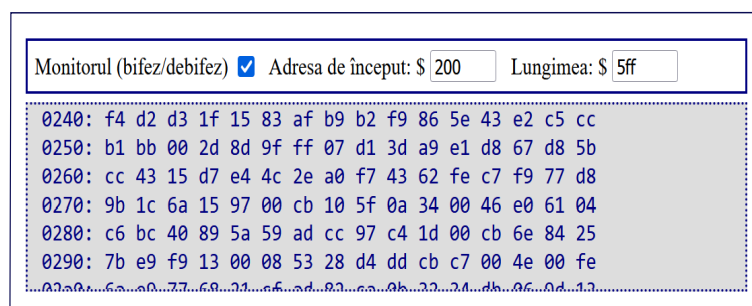


**Fig. 0.3** Monitorul execuției.

Dnul. Nick Morgan, un *creator* de programe profesionist, vine cu o soluție *parțială* a acestei probleme spinoase a deprinderii cu programarea în limbaj de asamblare: *emulatorul* Easy6502. Imitând un *microcomputer*, Easy6502 are sub capotă un *asamblor*, un *editor de text* permisiv — vezi Figura 0.1 —, un panou de *mesagerie* — Figura 0.2 —, un *monitor* al execuției — Figura 0.3 —, un *ecran* și un *depanator* de programe — Figura 0.4 —, un *descărcător* al conținutului memoriei și un *dezasamblor* — Figura 0.5 —.

Dnul. Morgan le oferă amatorilor de programare *web*, pe platforma `Github`, codul-sursă al emulatorului `Easy6502`. Face aceasta sub licența **CC BY 4.0** [9], adică permițând modificarea, reorganizarea, popularizarea acestei aplicații eficiente, pe care a adaptat-o, la rândul său, după aplicația *originală*, datorată dnului. Stian Sørensen și conservată de binevoitori pe platforma `Wayback Machine`.

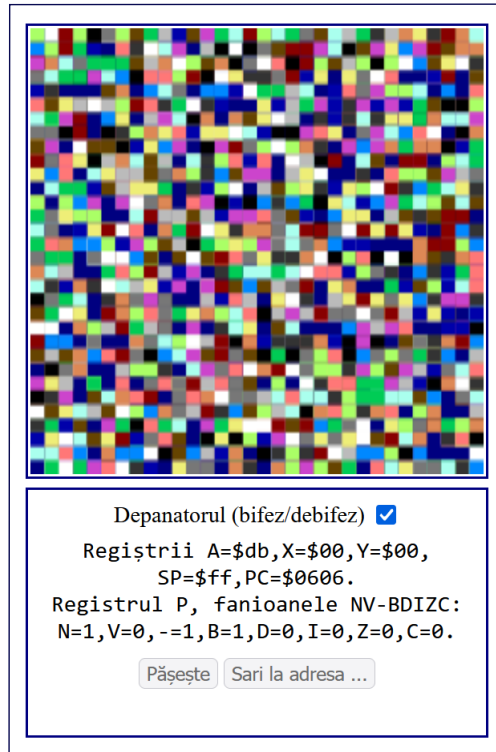


**Fig. 0.4** Terminalul grafic și depanatorul.

În ce mă privește, în materialul de față, prezint o versiune *cu modificări, parțială* în scop pedagogic, a codului-sursă al emulatorului `Easy6502`, versiune ce poate fi preluată, modificată, reorganizată și popularizată de cititori sub licența **CC BY 4.0**.

Programul care *produce* aplicația `Easy6502` — într-un *browser* de Internet — cuprinde instrucțiuni scrise în HTML, CSS și JavaScript. Comentariile din secțiunile de prezentare a instrucțiunilor îi permit cititorului să se familiarizeze cu diverse elemente de programare în aceste limbaje de nivel înalt, fără a fi nevoie de experiență semnificativă. De îndată ce detaliile codului-sursă îi vor fi la îndemână, cititorul atent va ști, printre altele, de ce *registrele de memorie nu se găsesc în același loc cu stiva de memorie* [70, pag. 40].

Materialul de față le poate fi de folos unor studenți automatiști sau electrotehniști care urmează cursuri de tipul *Sisteme cu microprocesoare* ori *Echipamente numerice*. Porțiuni ale sale au fost prezentate studenților de la Universitatea din Craiova specializați în *matematică informatică* și care au participat la opțiunile ASOR (Arhitecturi, sisteme de operare, rețele) și LFCMVIA (Limbaje formale, compilatoare, mașini virtuale și inteligență artificială).

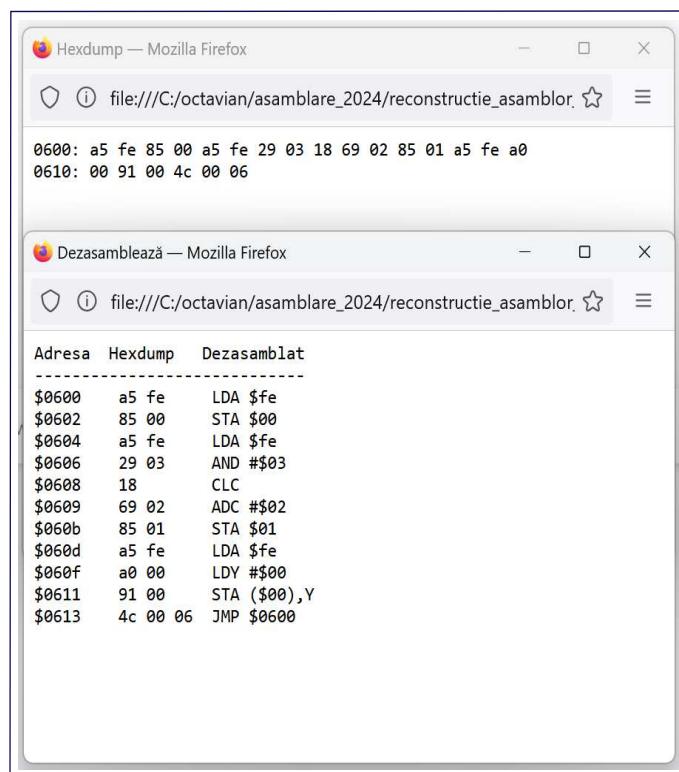


**Fig. 0.5** Ferestrele de *hexdump* și de dezasamblare.

O variantă a materialului de față a fost tipărită la Editura Sitech din Craiova, în anul 2024 (**ISBN** 978-606-11-8709-6).

Emulatorul este disponibil *online* la adresa <https://bit.ly/easy6502>.

# Cuprins

|          |                                                                         |           |
|----------|-------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Interfața cu utilizatorul a aplicației Easy6502</b>                  | <b>1</b>  |
| <b>2</b> | <b>Codul JavaScript al aplicației Easy6502</b>                          | <b>11</b> |
| 2.1      | Scheletul aplicației                                                    | 11        |
| 2.2      | Funcțiile auxiliare                                                     | 17        |
| 2.3      | Funcția initializareAplicatie()                                         | 19        |
| 2.4      | Funcția Ecranul()                                                       | 21        |
| 2.5      | Funcția Memoria()                                                       | 22        |
| 2.6      | Funcția InterfataCuUtilizatorul()                                       | 24        |
| 2.7      | Funcția Etichetele()                                                    | 27        |
| 2.8      | Microprocesorul 6502. Schema-bloc                                       | 31        |
| 2.9      | Funcția Asamblorul()                                                    | 33        |
| 2.9.1    | Funcțiile assembleazaCodul() și sanitizeazaLinia()                      | 33        |
| 2.9.2    | Funcția preprocesare()                                                  | 36        |
| 2.9.3    | Funcțiile assembleazaLinia() și DCB()                                   | 37        |
| 2.9.4    | Funcțiile incearcaParsarea...()                                         | 42        |
| 2.9.5    | Funcțiile verifica...()                                                 | 44        |
| 2.9.6    | Funcțiile urca...(), ...FereastraPopUp() și hexDumpAsamblorul()         | 54        |
| 2.9.7    | Funcțiile citesteModulSiCodul() și dezasambleazaCodul() ..              | 56        |
| 2.9.8    | Funcția creeazaInstructiunea()                                          | 58        |
| 2.9.9    | Codurile operaționale                                                   | 60        |
| 2.10     | Funcția Simulatorul()                                                   | 67        |
| 2.10.1   | Funcțiile setare...()                                                   | 67        |
| 2.10.2   | Funcțiile BIT(), CLC(), SEC(), CLV(), DEC(), INC() și saltRamificatie() | 69        |
| 2.10.3   | Funcțiile fanionul...(), modulZecimal() și faComparatia() ..            | 70        |
| 2.10.4   | Funcțiile testeaza...()                                                 | 71        |
| 2.10.5   | Funcțiile urca...(), coboara...()                                       | 73        |
| 2.10.6   | Funcțiile executa...(), multiExecuta()                                  | 76        |

|                                                                                                                 |            |
|-----------------------------------------------------------------------------------------------------------------|------------|
| 2.10.7 Funcțiile actualizeazaMonitorul() și manevreLaSchimba-<br>reaPlajeiMonitorului() . . . . .               | 78         |
| 2.10.8 Funcțiile depaneazaExecutia(), actualizeazaInformatiaDe-<br>panatorului() și adresaUndeMergi() . . . . . | 79         |
| 2.10.9 Funcțiile ...Depanatorul(),...Simulatorul() . . . . .                                                    | 81         |
| 2.10.10 Instrucțiunile executabile . . . . .                                                                    | 82         |
| <b>Referințe Bibliografice . . . . .</b>                                                                        | <b>101</b> |
| <b>Index . . . . .</b>                                                                                          | <b>105</b> |

## Lista de Figuri

|     |                                                                                            |      |
|-----|--------------------------------------------------------------------------------------------|------|
| 0.1 | Editorul de text. ....                                                                     | vii  |
| 0.2 | Panoul de mesaje. ....                                                                     | viii |
| 0.3 | Monitorul execuției. ....                                                                  | viii |
| 0.4 | Terminalul grafic și depanatorul. ....                                                     | ix   |
| 0.5 | Ferestrele de <i>hexdump</i> și de dezasamblare. ....                                      | x    |
| 1.1 | Structura aplicației Easy6502. Itemii care nu conțin simbolul “.”<br>sunt directoare. .... | 1    |
| 1.2 | Elementele HTML ale interfeței cu utilizatorul. ....                                       | 2    |
| 2.1 | Schema-bloc a microprocesorului 6502 [70, pag. 16], [54]. ....                             | 31   |



# Capitolul 1

## Interfața cu utilizatorul a aplicației Easy6502

Într-un *director* plasat convenabil — intitulat, de exemplu, `Easy6502_versiunea1` — construim elementele descrise în Figura 1.1.

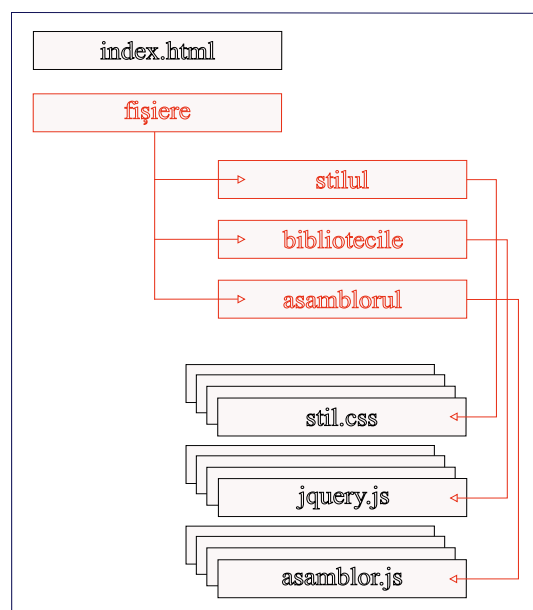


**Fig. 1.1** Structura aplicației Easy6502. Itemii care nu conțin simbolul “.” sunt directoare.

Acestea sunt fișiere-*text*, cu *extensiile*<sup>1</sup> `html`, `css` și `js`, respectiv subdirectoare dedicate elementelor de stil grafic — `stilul` —, programelor auxiliare — `bibliotecile` —, codului principal al aplicației — `asamblorul` —.

<sup>1</sup> Lucrăm în sistemul de operare Windows 11 al companiei Microsoft.

Construcția și testarea — *debugging*-ul — aplicației vor fi realizate în *browser*-ul de Internet al fundației Mozilla, *Firefox* [27].

Fișierele \*.html conțin instrucțiuni ale limbajului de prezentare hipertextuală HTML [43], fișierele \*.css instrucțiuni ale limbajului de redare grafică de conținut CSS [10] iar fișierele \*.js instrucțiuni ale limbajului de programare tipizat dinamic JavaScript [47].

Program:> În codul care urmează intervine instrucțiunea `id="număr"` [5]. Aceasta ne permite să localizăm elementele HTML [15] ilustrate în Figura 1.2.

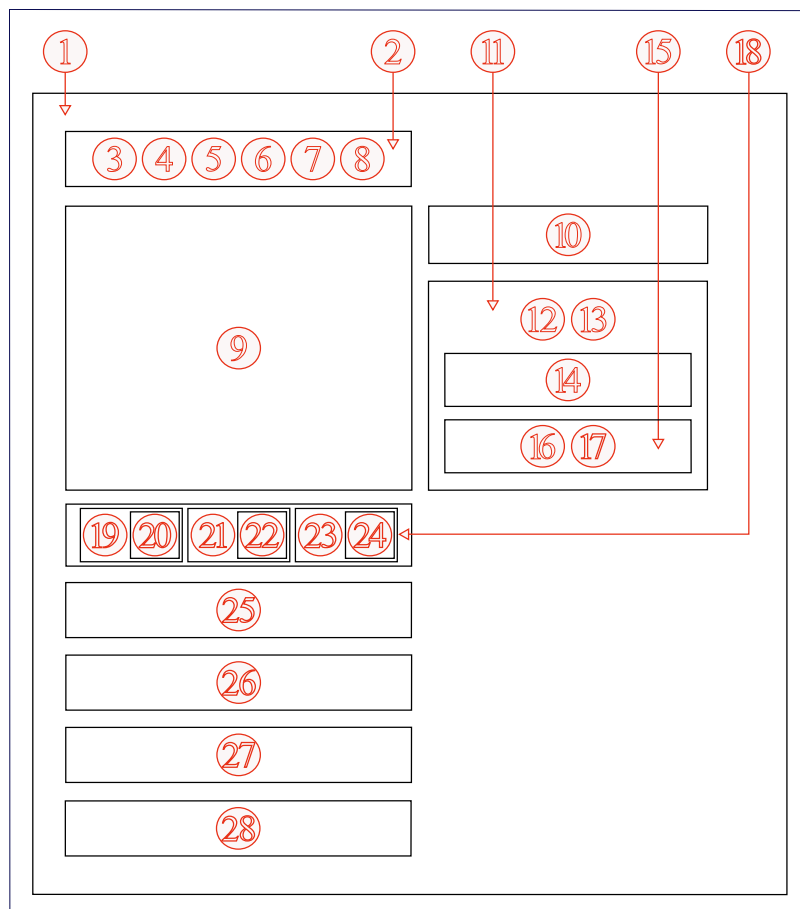


**Fig. 1.2** Elementele HTML ale interfeței cu utilizatorul.

```
index.html
1 <!doctype html>
2 <html
3   lang           = "ro"
```

```

4 >
5 <head>
6   <meta
7     charset      ="utf-8"
8   >
9   <link
10     href         ="fisiere/stilul/stil.css"
11     rel          ="stylesheet"
12     type         ="text/css"
13   >
14   <title>
15     Asamblor &#351;i emulator
16     pentru microprocesorul 6502
17   </title>
18 </head>
19
20 <body>
21   <div
22     id           ="1"
23     class        ="aplicatia"
24   >
25     <div
26       id         ="2"
27       class       ="butoanele"
28     >
29       <input
30         id        ="3"
31         type       ="button"
32         value      ="Asambleaz&#259;"
33         class      ="butonAsamblare"
34       >
35       <input
36         id        ="4"
37         type       ="button"
38         value      ="Execut&#259;"
39         class      ="butonExecutie"
40       >
41       <input
42         id        ="5"
43         type       ="button"
44         value      ="Reseteaz&#259;"
45         class      ="butonResetare"
46       >
47       <input
48         id        ="6"
49         type       ="button"
50         value      ="HexDump"
51         class      ="butonHexDump"
52       >
53       <input
54         id        ="7"
55         type       ="button"
56         value      ="Dezasambleaz&#259;"
57         class      ="butonDezasamblare"

```

```

58     >
59     <input
60         id         ="8"
61         type        ="button"
62         value       ="Informa&#355;ii/&#350;terge"
63         class       ="butonInformatii "
64     >
65 </div>
66 <textarea
67     id         ="9"
68     class      ="codul "
69     spellcheck ="false"
70 ></textarea>
71 <canvas
72     id         ="10"
73     class      ="ecranul "
74 ></canvas>
75 <div
76     id         ="11"
77     class      ="depanatorul "
78 >
79     <label
80         id         ="12"
81     >
82         Depanatorul (bifez/debifez)
83         <input
84             id         ="13"
85             type        ="checkbox"
86             class       ="depaneaz "
87             name        ="depanare"
88         >
89     </label>
90     <div
91         id         ="14"
92         class      ="minidepanatorul "
93     ></div>
94     <div
95         id         ="15"
96         class      ="butoanele"
97     >
98         <input
99             id         ="16"
100             type       ="button"
101             value      ="P&#259;&#351;e&#351;te"
102             class      ="butonPasire"
103         >
104         <input
105             id         ="17"
106             type       ="button"
107             value      ="Sari la adresa ..."
108             class      ="butonSariLa"
109         >
110     </div>
111 </div>

```

```

112     <div
113         id             ="18"
114         class          ="controaleleMonitorului "
115     >
116         <label
117             id             ="19"
118         >
119             Monitorul (bifez/debifez)
120             <input
121                 id             ="20"
122                 type           ="checkbox"
123                 class          ="monitorizez "
124                 name           ="monitorizare "
125             >
126         </label>
127         <label
128             id             ="21"
129         >
130             Adresa de &#238;nceput: &#36;
131             <input
132                 id             ="22"
133                 type           ="text "
134                 value          ="0"
135                 class          ="inceputul "
136                 name           ="inceput "
137                 spellcheck     ="false"
138             >
139         </label>
140         <label
141             id             ="23"
142         >
143             Lungimea: &#36;
144             <input
145                 id             ="24"
146                 type           ="text "
147                 value          ="ff"
148                 class          ="lungimea"
149                 name           ="lungime"
150                 spellcheck     ="false"
151             >
152         </label>
153     </div>
154     <div
155         id             ="25"
156         class          ="monitorul "
157     >
158         <pre><code></code></pre>
159     </div>
160     <div
161         id             ="26"
162         class          ="mesajele"
163     >
164         <pre><code></code></pre>
165     </div>

```

```
166         <div
167             id             ="27"
168             class          ="informatiile"
169         >
170 Informații:
171
172 Adrese de memorie speciale
173
174 Atunci când execut o
175 instrucțiune, Emulatorul
176 generează un octet
177 aleatoriu. Acesta va fi plasat
178 la adresa de memorie și
179 Mai multe detalii se găsesc
180 în codul funcției
181 executa().
182
183 La adresa de memorie și
184 găsim codul ASCII al
185 ultimei taste apăsate.
186 Mai multe detalii se găsesc
187 în codul funcției
188 stocheazaApasareaPeTasta().
189
190 Plaja de adrese de memorie
191 și 200 (colul din
192 stângă sus) și
193 și 5ff (colul din
194 dreapta jos) le corespunde
195 pixelilor ecranului. Stocarea
196 unui octet pe această
197 plajă va produce colorarea
198 unui pixel cu una din
199 următoarele 16 culori.
200 Ecranul are 32x32 pixeli.
201 Mai multe detalii se găsesc
202 în codul funcției
203 stocheazaOctetul().
204
205 Cele 16 culori (Commodore 64):
206 0: Negru
207 1: Alb
208 2: Roșu
209 3: Cyan
210 4: Purpuriu
211 5: Verde
212 6: Albastru
213 7: Galben
214 8: Portocaliu
215 9: Maro
216 a: Roșu deschis
217 b: Gri închis
218 c: Gri
219 d: Verde deschis
```

```

220 &#36;e: Albastru deschis
221 &#36;f: Gri deschis
222
223 &#206;nteruperile
224
225 Nu sunt implementate.
226 Instruc&#355;iunile CLI, SEI
227 sunt asamblate dar emularea
228 execu&#355;iei lor va arunca
229 excep&#355;ii JavaScript.
230
231 Temporizatoare &#351;i
232 cicluri-ma&#351;in&#259;
233
234 Nu sunt implementate. Detalii
235 privind modificarea vitezei de
236 execu&#355;ie se g&#259;sesc
237 &#238;n codul func&#355;iei
238 executaBinarul().
239 </div>
240 <div
241     id            ="28"
242     class         ="multumiri"
243 >
244     Codurile acestei aplica&#355;ii
245     folosesc munca dlor.
246     <a
247         href      ="https://web.archive.org/web\
248         (cont.)/20070516072609/http%3A/www.6502asm.com/"
249         target    ="_blank"
250     >Stian S&#248;reng</a>
251     &#351;i
252     <a
253         href      ="https://github.com/skilldrick/easy6502"
254         target    ="_blank"
255     >Nick Morgan</a>.<br>
256     <em>
257         Hats off to both of you,
258         Stian S&#248;reng and Nick Morgan!
259     </em>
260 </div>
261 <script
262     src            ="fisiere/bibliotecile/es5-shim.js"
263 ></script>
264 <script
265     src            ="fisiere/bibliotecile/jquery_v3_7_1.js"
266 ></script>
267 <script
268     src            ="fisiere/asamblorul/asamblor.js"
269 ></script>
270 </body>
271 </html>

```

Butoanele — *input*-uri [45] de tip `button` —, identificabile prin cercurile roșii cu numerele **3–8**, **16**, **17**, sunt activate, respectiv dezactivate de *stările* [75, pag. 33] aplicației. Aceste stări vor fi definite în cadrul codului JavaScript. Itemul **25** este invizibil. El va deveni vizibil doar dacă este bifat itemul **20** — un *input* de tip `checkbox` —. Și itemul **27** este invizibil. Conținutul său va fi vizualizat prin apăsarea butonului **8**: afișarea propriu-zisă a acestui conținut se va face în interiorul itemului **26**.

În listările de cod din acest tutorial ne bazăm pe capacitatea *compilatoarelor* [75, pag. 139] de a ignora spațiile goale — formate din caractere tipografice invizibile — [65]. Totuși, ca la rândurile **158**, **164**, **249**, **254**, trebuie ținut seama de *contextul* [76] exprimat de instrucțiuni. Aici, spațiile goale produc efecte nedorite. În particular, datorită caracterului de *bloc* [6] al elementului HTML `pre` [62], textul de la rândurile **170–238** nu trebuie *indentat*.

Formatul corect al elementelor HTML folosite este fie

```
<etichetă ...>conținut</aceeași etichetă>
```

— unde *conținutul* poate lipsi, vezi elementul `script` [72] — fie

```
<etichetă>.
```

Ultima situație s-a întâlnit la elementele `input`, respectiv `br` [7]. Conformitatea cu *standardul* HTML a programului anterior poate fi testată *online* [82].

Atributul `spellcheck` [77], de control al despărțirii în silabe și scrierii, este opțional [78].

Modul în care sunt procesate instrucțiunile de către *browser*-ul Firefox poate fi vizualizat folosind setul de unelte Web Developer Tools [74], accesibil prin apăsarea simultană a tastelor `Ctrl`, `Shift` și `I`.

Program:> Codul CSS care urmează presupune un *display*<sup>2</sup> cu *rezoluția* [68] de **2560** de pixeli pe orizontală și **1600** de pixeli [52] pe verticală, evident, în orientarea `landscape`. El caracterizează *clasele* — vezi atributul `className` [5] — diverselor elemente HTML din listarea anterioară.

```
stil.css
1 .aplicatia {
2   width:           940px;
3   margin:          30px auto;
4   font-size:       16px;
5   line-height:     1.2;
6 }
7 .butoanele {
8   margin:          8px 0;
9 }
10 .codul {
11   width:           600px;
12   height:          506px;
13   margin:          0 0 6px 0;
14   padding:         2px;
15   border:          2px dotted navy;
16   font-family:     monospace;
17   font-size:       18px;
```

<sup>2</sup> În românește, *terminal grafic*, respectiv *ecran* sau *afișor*. Vezi [4, pp. 70, 71].

```
18  color:          maroon;
19  overflow:       auto;
20  float:          left;
21 }
22 .ecranul {
23  width:          320px;
24  height:         320px;
25  margin:         0 0 2px 0;
26  padding:        2px;
27  border:         2px solid navy;
28  float:          right;
29 }
30 .depanatorul {
31  width:          320px;
32  height:         160px;
33  margin:         4px 0 0 0;
34  padding:        8px 2px;
35  border:         2px solid navy;
36  text-align:     center;
37  float:          right;
38 }
39 .minidepanatorul {
40  margin:         4px 0 0 0;
41  padding:        2px;
42  font-family:    monospace;
43  font-size:      16px;
44 }
45 .controaleleMonitorului {
46  margin:         0 0 6px 0;
47  padding:        6px;
48  border:         2px solid navy;
49  width:          592px;
50  clear:          both;
51 }
52 .inceputul, .lungimea {
53  width:          48px;
54 }
55 .controaleleMonitorului input {
56  margin-right:   8px;
57 }
58 .monitorul {
59  width:          580px;
60  height:         100px;
61  margin:         6px 0 0 0;
62  padding:        12px;
63  border:         2px dotted navy;
64  background-color: #ddd;
65  color:          navy;
66  overflow:       auto;
67  display:        none;
68 }
69 .monitorulInvalidat {
70  padding:        2px 1px;
71  border:         2px dotted #c00;
```

```
72 }
73 .mesajele {
74     width:          580px;
75     height:         320px;
76     margin:         6px 0 0 0;
77     padding:        0 12px;
78     border:         2px dotted navy;
79     background-color: #eee;
80     overflow:       auto;
81     text-align:     left;
82     font-size:      16px;
83     color:          maroon;
84 }
85 .informatiile {
86     display:        none;
87 }
88 .multumiri {
89     font-size:      20px;
90     float:          left;
91 }
92 .multumiri em {
93     font-style:     italic;
94     color:          navy;
95 }
```

Conformitatea cu *standardul* CSS a programului anterior poate fi testată *online* [81].

## Capitolul 2

# Codul JavaScript al aplicației Easy6502

### 2.1 Scheletul aplicației

Folosim faptul că funcțiile sunt elemente de *prima clasă* [29] ale limbajului de programare JavaScript, adică sunt tratate asemeni oricărei *variabile*. Tipurile de date utilizate în JavaScript sunt: numere, șiruri de caractere alfanumerice, valori de adevăr, simboluri, `BigInt` — întregi mari —, `Object` — obiecte —, `Null` — elementul *nul* —, `Undefined` — elementul *nedefinit* — [80]. Funcțiile sunt *obiecte* funcție [57].

Funcțiile pot fi utilizate ca *parametri* în funcții, respectiv *returnate* de funcții fără să apelăm la nicio sintaxă specială.

Program:> În codul următor, întrebuițăm scrierea *cămilă* [71] — ondulată — la numele variabilelor și al funcțiilor *imbricate* [30] ori *interioare* [31]. Funcțiile care returnează *obiecte* JavaScript [58] vor fi notate cu scrierea *Pascal* — scrierea cămilă care începe cu majusculă —.

```
asamblor.js
1 "use strict";
2
3 function AplicatiaEasy6502( nodul ) {
4     var $nod      = $( nodul );
5     var interfata = InterfataCuUtilizatorul();
6     var afisor    = Ecranul();
7     var memorie   = Memoria();
8     var etichete  = Etichetele();
9     var simulator = Simulatorul();
10    var asamblor  = Asamblorul();
11
12    function initializareAplicatie() {}
13
14    function InterfataCuUtilizatorul() {
15        function initializareInterfata( e ) {}
16        function pornesteInterfata() {}
17        function opresteInterfata() {}
18        function asamblareReusita() {}
```

```

19  function pornitDepanare() {}
20  function opritDepanare() {}
21  function monitorizareSchimbari( stare ) {}
22  function arataInformatii() {}
23  function captureazaTaburiInEditor( e ) {}
24
25  return {
26      interfataInitializare      : initializeInterfata,
27      interfataPornește         : porneșteInterfata,
28      interfataOprește          : opreșteInterfata,
29      interfataAsamblareReusita  : asamblareReusita,
30      interfataPornitDepanare    : pornitDepanare,
31      interfataOpritDepanare     : opritDepanare,
32      interfataMonitorizareSchimbari : monitorizareSchimbari,
33      interfataArataInformatii    : arataInformatii,
34      interfataCaptureazaTaburi   : captureazaTaburiInEditor
35  };
36  }
37
38  function Ecranul() {
39      function initializeEcran() {}
40      function resetareEcran() {}
41      function actualizarePixelEcran() {}
42
43      return {
44          ecranulInitializare      : initializeEcran,
45          ecranulResetare          : resetareEcran,
46          ecranulActualizarePixel : actualizarePixelEcran
47      };
48  }
49
50  function Memoria() {
51      function seteaza() {}
52      function citește() {}
53      function citeșteCuvantul() {}
54      function stocheazaOctetul() {}
55      function stocheazaApasareaPeTasta() {}
56      function formateaza() {}
57
58      return {
59          memoriaSeteaza          : seteaza,
60          memoriaCitește          : citește,
61          memoriaCiteșteCuvantul  : citeșteCuvantul,
62          memoriaStocheazaOctetul : stocheazaOctetul,
63          memoriaStocheazaApasarea : stocheazaApasareaPeTasta,
64          memoriaFormateaza       : formateaza
65      };
66  }
67
68  function Simulatorul() {
69      var registrula                = 0;
70      var registrulX                = 0;
71      var registrulY                = 0;
72      var registrulP                = 0;

```

```

73     var registrulPC                                = 0x600;
74     var registrulSP                                = 0xff;
75     var codInExecutie                              = false;
76     var depanareSimulator                          = false;
77     var monitorizareSimulator                      = false;
78     var identificatorulInstructiuniiDeExecutat;
79     var AND                                          =
80         setareFanioaneleNZVpentruRegistrulA;
81     var ASL                                          =
82         setareFanioaneleNZV;
83     var EOR                                          =
84         setareFanioaneleNZVpentruRegistrulA;
85     var LDA                                          =
86         setareFanioaneleNZVpentruRegistrulA;
87     var LDX                                          =
88         setareFanioaneleNZVpentruRegistrulX;
89     var LDY                                          =
90         setareFanioaneleNZVpentruRegistrulY;
91     var LSR                                          =
92         setareFanioaneleNZV;
93     var ORA                                          =
94         setareFanioaneleNZVpentruRegistrulA;
95     var ROL                                          =
96         setareFanioaneleNZV;
97     var ROR                                          =
98         setareFanioaneleNZV;
99     var instructiunile                             = {};
100
101     function setareFanioaneleNZV( valoarea ) {}
102     function setareFanionulCdeLaBitul0( valoarea ) {}
103     function setareFanionulCdeLaBitul7( valoarea ) {}
104     function setareFanioaneleNZVpentruRegistrulA() {}
105     function setareFanioaneleNZVpentruRegistrulX() {}
106     function setareFanioaneleNZVpentruRegistrulY() {}
107     function setareFanionulV() {}
108     function BIT( valoarea ) {}
109     function CLC() {}
110     function SEC() {}
111     function CLV() {}
112     function DEC( adresa ) {}
113     function INC( adresa ) {}
114     function saltRamificatie( deplasamentul ) {}
115     function fanionulVsetat() {}
116     function fanionulCsetat() {}
117     function fanionulNsetat() {}
118     function fanionulZsetat() {}
119     function modulZecimal() {}
120     function faComparatia( registrul, valoarea ) {}
121     function testeazaSBC( valoarea ) {}
122     function testeazaADC( valoarea ) {}
123     function urcaPeStiva( valoarea ) {}
124     function coboaraDinStiva() {}
125     function coboaraUnOctetDinMemorie() {}
126     function coboaraUnCuvantDinMemorie() {}

```

```

127 function executaUrmatoareaInstructiune() {}
128 function executa( codInDepanare ) {}
129 function executaBinarul() {}
130 function multiExecuta() {}
131 function actualizeazaMonitorul() {}
132 function manevreLaSchimbareaPlajeiMonitorului() {}
133 function depaneazaExecutia() {}
134 function actualizeazaInformatiaDepanatorului() {}
135 function adresaUndeMergi() {}
136 function opresteDepanatorul() {}
137 function activeazaDepanatorul() {}
138 function reseteazaSimulatorul() {}
139 function opresteSimulatorul( e ) {}
140 function monitorizareSchimbareSimulatorul( starea ) {}
141
142 return {
143     simulatorulExecuta           : executaBinarul,
144     simulatorulStartDepanare     : activeazaDepanatorul,
145     simulatorulStopDepanare      : opresteDepanatorul,
146     simulatorulDepaneaza         : depaneazaExecutia,
147     simulatorulAdresa            : adresaUndeMergi,
148     simulatorulReseteaza         : reseteazaSimulatorul,
149     simulatorulOpreste           : opresteSimulatorul,
150     simulatorulMonitorizeaza     :
151         monitorizareSchimbareSimulatorul,
152     simulatorulManevreSchimbare :
153         manevreLaSchimbareaPlajeiMonitorului
154 };
155 }
156
157 function Etichetele() {
158     function indexeazaLiniile( liniile, simbolurile ) {}
159     function indexeazaLinia( intrarea, simbolurile ) {}
160     function urca( numele ) {}
161     function gaseste( numele ) {}
162     function seteazaRegistrulPC( numele, adresa ) {}
163     function citesteRegistrulPC( numele ) {}
164     function afiseazaMesajulEtichete() {}
165     function reseteazaEtichete() {}
166
167     return {
168         eticheteleIndexeaza : indexeazaLiniile,
169         eticheteleGaseste   : gaseste,
170         eticheteleCitestePC : citesteRegistrulPC,
171         eticheteleMesaj     : afiseazaMesajulEtichete,
172         eticheteleReseteaza : reseteazaEtichete
173     };
174 }
175
176 function Asamblorul() {
177     var valoareaRegistruluiPCpentruCodObisnuit;
178     var lungimeaCodului;
179     var codulAsamblatCorect           = false;
180     var depasireaPlajeiRamificarii    = false;

```

```

181     var codurileOperationale           = [];
182     var modurileDeAdresare             = [
183         null ,
184         "Imm" ,
185         "ZP" , "ZPX" , "ZPY" ,
186         "ABS" , "ABSX" , "ABSY" ,
187         "IND" , "INDX" , "INDY" ,
188         "SNGL" ,
189         "BRA"
190     ];
191     var lungimileInstructiunilor       = {
192         Imm : 2,
193         ZP : 2, ZPX : 2, ZPY : 2,
194         ABS : 3, ABSX : 3, ABSY : 3,
195         IND : 3, INDX : 2, INDY : 2,
196         SNGL : 1,
197         BRA : 2
198     };
199
200     function asambleazaCodul() {}
201     function sanitizeazaLinia( linia ) {}
202     function preprocesare( liniile ) {}
203     function asambleazaLinia(
204         ceIntra, numarulLiniei, simboluri ) {}
205     function DCB( parametrul ) {}
206     function incearcaParsareaOperanduluiOctet(
207         parametrul, simboluri ) {}
208     function incearcaParsareaOperanduluiCuvant(
209         parametrul, simboluri ) {}
210     function verificaRamificare(
211         parametrul, codOperational ) {}
212     function verificaImediat(
213         parametrul, codOperational, simboluri ) {}
214     function verificaIndirect(
215         parametrul, codOperational, simboluri ) {}
216     function verificaIndirectX(
217         parametrul, codOperational, simboluri ) {}
218     function verificaIndirectY(
219         parametrul, codOperational, simboluri ) {}
220     function verificaUnSingurOctet(
221         parametrul, codOperational ) {}
222     function verificaPaginaZero(
223         parametrul, codOperational, simboluri ) {}
224     function verificaAdresareaAbsolutaX(
225         parametrul, codOperational, simboluri ) {}
226     function verificaAdresareaAbsolutaY(
227         parametrul, codOperational, simboluri ) {}
228     function verificaPaginaZeroX(
229         parametrul, codOperational, simboluri ) {}
230     function verificaPaginaZeroY(
231         parametrul, codOperational, simboluri ) {}
232     function verificaAdresareaAbsoluta(
233         parametrul, codOperational, simboluri ) {}
234     function urcaOctetul( valoarea ) {}

```

```

235     function urcaCuvantul( valoarea ) {}
236     function deschideFereastraPopUp(
237         continutul, titlul ) {}
238     function hexDumpAsamblorul() {}
239     function citesteModulSiCodul( octetul ) {}
240     function creeazaInstructiunea( adresa ) {}
241     function dezasambleazaCodul() {}
242
243     return {
244         asamblorulFaLinia           : asambleazaLinia,
245         asamblorulFaCodul          : asambleazaCodul,
246         asamblorulCitesteValoareaPCcurenta :
247             function() {
248                 return valoareaRegistruluiPCpentruCodObisnuit;
249             },
250         asamblorulHexDump          : hexDumpAsamblorul,
251         asamblorulDesfaCodul        : dezasambleazaCodul
252     };
253 }
254
255 function auxiliarEliminareSpatiiGoale() {}
256 function auxiliarConvertescNumarLaHexa( numarul ) {}
257 function auxiliarConvertescAdresaLaHexa( adresa ) {}
258 function auxiliarSeteazaAleatoriuOctetul() {}
259 function auxiliarAfiseazaMesajul( textul ) {}
260
261 initializeAplicatie();
262 }
263
264 $( document )
265 .ready(
266     () => $( ".aplicatia" )
267     .each(
268         function( index, element ) {
269             AplicatiaEasy6502( this );
270             if ( index > 0 ) {
271                 return false;
272             }
273         }
274     )
275 );

```

La rândul 247 introducem o funcție *anonimă* [28] iar la rândul 266 o funcție *săgeată* [41].

Funcțiile `ready()` [39], de la rândul 265, și `each()` [35], de la rândul 267, îi aparțin bibliotecii JavaScript `jQuery` [48]. Expresia `$( element din DOM )` anunță folosirea acestei biblioteci [1].

## 2.2 Funcțiile auxiliare

Aceste funcții se găsesc la rândurile **255–259** ale listării de cod din Secțiunea 2.1.

Program:> În codul care urmează, utilizăm *expresii regulate*<sup>1</sup> [21], [75, pag. 43] pentru eliminarea caracterelor tipografice invizibile, respectiv *măști de biți* pentru extragerea de informații numerice.

```

1 function auxiliarEliminareSpatiiGoale() {
2   var textul1 = $nod.find( ".codul" ).val();
3   textul1 = textul1.replace( /^[\\r\\n]+/, "" );
4   textul1 = textul1.replace( /[\\s]+$/, "" );
5   $nod.find( ".codul" ).val( textul1 );
6 }
7
8 function auxiliarConvertescNumarLaHexa( numarul ) {
9   var cifrele = "0123456789abcdef";
10  var patruBitiSuperior = (numarul & 0xf0) >> 4;
11  var patruBitiInferior = numarul & 0x0f;
12  return (
13    cifrele.substring(
14      patruBitiSuperior,
15      patruBitiSuperior + 1
16    )
17    + cifrele.substring(
18      patruBitiInferior,
19      patruBitiInferior + 1
20    )
21  );
22 }
23
24 function auxiliarConvertescAdresaLaHexa( adresa ) {
25  return (
26    auxiliarConvertescNumarLaHexa(
27      (adresa >> 8) & 0xff
28    )
29    + auxiliarConvertescNumarLaHexa(
30      adresa & 0xff
31    )
32  );
33 }
34
35 function auxiliarSeteazaAleatoriuOctetul() {
36  memorie.memoriaSeteaza(
37    0xfe,
38    Math.floor( Math.random() * 256 )
39  );
40 }
41
```

---

<sup>1</sup> În limba engleză, *regular expression* (sing.).

```

42 function auxiliarAfiseazaMesajul ( textul ) {
43     if ( textul.length > 1 ) {
44         textul += "\r\n";
45     }
46     $nod.find ( ".mesajele code" ).append ( textul )
47     .scrollTop ( 10000 );
48 }

```

La rândul **2**, folosind funcția jQuery `find()` [37] și *selectorul* jQuery de clase HTML [73] “.codul”, capturăm un șir de caractere alfanumerice [40].

La rândul **3**, folosind expresia regulată din primul argument al funcției JavaScript `replace()` [34], eliminăm — dacă există — caracterele `\r` — carriage return —, respectiv `\n` — newline — de la *începutul* șirului capturat. Ele sunt responsabile de existența *rândurilor goale* în șirul de caractere. La rândul **4**, folosind expresia regulată din primul argument al funcției `replace()`, eliminăm — dacă există — caracterele `\s` — spații goale într-un rând de text — de la *sfârșitul* șirului capturat.

La rândul **8**, presupunem că *valoarea* primită de argumentul `numarul` este un număr întreg reprezentat în *baza de numerație* 10 printr-un element din intervalul — *plaja* de valori —  $[0, 255]$ . Această valoare va desemna, în baza de numerație 2, un *octet* de date. Octetul este compus din două *cvarțete*<sup>2</sup> consecutive de date, corespunzând celor două cifre ale bazei de numerație 16 necesare reprezentării valorii. Izolăm primul cvarțet într-un octet *nou*, la rândul **10**, folosind masca de biți `0xf0`. Aceasta are expresia `11110000` în baza de numerație 2. Apoi, pentru determinarea *primeia* din cele două cifre hexazecimale ale valorii de prelucrat, ne deplasăm *la dreapta* cu 4 biți în octetul nou. La rândul **11**, într-un octet nou, izolăm cea de-a doua cifră hexazecimală căutată. Masca folosită, `0x0f`, are expresia `00001111` în baza de numerație 2.

La rândul **24**, presupunem că *valoarea* primită de argumentul `adresa` este un număr întreg reprezentat în baza de numerație 10 printr-un element din intervalul  $[0, 65535]$ . Această valoare va desemna, în baza de numerație 2, doi octeți consecutivi de date. Acești octeți vor fi izolați, la rândurile **27** și **30**, cu ajutorul măștii de biți `0xff`. Masca are expresia `11111111` în baza de numerație 2. Ordinea în care *concatenăm* cele două șiruri de caractere formate din câte două cifre hexazecimale pe care le returnează funcția `auxiliarConvertescNumarLaHexa()`, la rândurile **26** și **29**, ne permite reconstituirea valorii de prelucrat sub forma unui șir de patru cifre hexazecimale.

La rândul **38**, prin înmulțire cu 256 a valorii returnate de funcția JavaScript `Math.random()` [32], construim un număr întreg *aleatoriu* din intervalul  $[0, 255]$ . El va fi *memorat*, folosind instrucțiunea de la rândurile **36** și urm., la adresa `0xfe` — adică, 254 în baza de numerație 10 —.

Funcția `auxiliarAfiseazaMesajul()` va produce afișarea valorii argumentului `textul`, de la rândul **42**, în cadrul elementului HTML **26** din Figura 1.1.

<sup>2</sup> În limba engleză, *nibble* (sing.). Sau *tetradă*.

## 2.3 Funcția `initializareAplicatie()`

Această funcție se găsește la rândul **12** al listării de cod din Secțiunea 2.1.

Program:> În codul următor, *editorul* de text al aplicației Easy6502 — elementul HTML **9** din Figura 1.2 — se bazează pe funcția jQuery `on()` [38] pentru captarea instrucțiunilor de programare.

```

1  function initializareAplicatie () {
2      var editorul;
3
4      auxiliarEliminareSpatiiGoale();
5      interfata.interfataInitializare();
6      afisor.ecranulInitializare();
7      simulator.simulatorulReseteaza();
8
9      $nod.find( ".butonAsamblare" ).click(
10         function() {
11             asamblor.asamblorulFaCodul();
12         }
13     );
14     $nod.find( ".butonExecutie" ).click(
15         simulator.simulatorulExecuta
16     );
17     $nod.find( ".butonExecutie" ).click(
18         simulator.simulatorulStopDepanare
19     );
20     $nod.find( ".butonResetare" ).click(
21         simulator.simulatorulReseteaza
22     );
23     $nod.find( ".butonHexDump" ).click(
24         asamblor.asamblorulHexDump
25     );
26     $nod.find( ".butonDezasamblare" ).click(
27         asamblor.asamblorulDesfaCodul
28     );
29     $nod.find( ".depanez" ).change(
30         function() {
31             var depanarea = $( this ).is( ":checked" );
32             if ( depanarea ) {
33                 interfata.interfataPornitDepanare();
34                 simulator.simulatorulStartDepanare();
35             } else {
36                 interfata.interfataOpritDepanare();
37                 simulator.simulatorulStopDepanare();
38             }
39         }
40     );
41     $nod.find( ".monitorizez" ).change(
42         function() {
43             var starea = this.checked;
44             interfata

```

```

45     .interfataMonitorizareSchimbari( starea );
46     simulator
47     .simulatorulMonitorizeaza( starea );
48 }
49 );
50 $nod.find( ".inceputul, .lungimea" ).on(
51     "blur",
52     simulator.simulatorulManevreSchimbare
53 );
54 $nod.find( ".butonPasire" ).click(
55     simulator.simulatorulDepaneaza
56 );
57 $nod.find( ".butonSariLa" ).click(
58     simulator.simulatorulAdresa
59 );
60 $nod.find( ".butonInformatii" ).click(
61     interfata.interfataArataInformatii
62 );
63
64 editorul = $nod.find( ".codul" );
65 editorul.on(
66     "keydown",
67     simulator.simulatorulOpreste
68 );
69 editorul.on(
70     "keydown",
71     interfata.interfataInitializare
72 );
73 editorul.on(
74     "keydown",
75     interfata.interfataCaptureazaTaburi
76 );
77
78 $( document ).on(
79     "keypress",
80     memorie.memoriaStocheazaApasarea
81 );
82
83 simulator.simulatorulManevreSchimbare();
84 }

```

La rândurile **51**, **66**, **79** întâlnim *evenimente* [20] privitoare la *nodurile* din DOM [14]. Evenimentul “blur” [16] are ca *urmare* modificarea atributelor elementelor HTML **22** și **24** din Figura 1.2. Evenimentul “keydown” [17] este tratat de funcțiile de *reacție*<sup>3</sup> de la rândurile **67**, **71** și **75** — ultima dintre ele permițând, d.ex., inserarea de Tab-uri în programul de asamblat fără să se modifice starea butoanelor interfeței cu utilizatorul —. Chiar dacă evenimentul “keypress” [18] este *dezafectat*<sup>4</sup> [19], în versiunea de față a aplicației Easy6502 el nu poate fi înlocuit fără a i se actualiza funcția de reacție de la rândul **80**.

<sup>3</sup> În limba engleză, *handler* (sing.). Sau *rutinele de tratare* [4, pag. 68].

<sup>4</sup> În limba engleză, *deprecated*.

## 2.4 Funcția Ecranul()

Această funcție se găsește la rândul **38** al listării de cod din Secțiunea 2.1.

Program:> În codul care urmează, suprafața elementului HTML **10** din Figura 1.2 este împărțită într-o rețea de  $32 \times 32$  de pătrate egale, *pixelii* afișorului. Aceștia pot fi colorați cu una din cele 16 culori predefinite — descrise la rândurile **206–221** ale listării de cod `index.html` din Capitolul 1 —.

```

1 function Ecranul() {
2   var paleta = [
3     "#000000", "#ffffff", "#880000", "#aaffee",
4     "#cc44cc", "#00cc55", "#0000aa", "#eeee77",
5     "#dd8855", "#664400", "#ff7777", "#333333",
6     "#777777", "#aaff66", "#0088ff", "#bbbbbb"
7   ];
8   var contextul;
9   var latimea;
10  var inaltimea;
11  var xNumeric = 32;
12  var yNumeric = 32;
13  var xPixel;
14  var yPixel;
15
16  function initializareEcran() {
17    var panza = $nod.find( ".ecranul" )[0];
18    latimea = panza.width;
19    inaltimea = panza.height;
20    contextul = panza.getContext( "2d" );
21    resetareEcran();
22    xPixel = latimea / xNumeric;
23    yPixel = inaltimea / yNumeric;
24  }
25
26  function resetareEcran() {
27    contextul.fillStyle = "navy";
28    contextul.fillRect( 0, 0, latimea, inaltimea );
29  }
30
31  function actualizarePixelEcran( adresa ) {
32    contextul.fillStyle
33      = paleta[
34        (memorie.memoriaCiteste( adresa )) & 0x0f
35      ];
36    var m = adresa - 0x200;
37    var y = Math.floor( m / yNumeric );
38    var x = m % xNumeric;
39    contextul.fillRect(
40      x * xPixel, y * yPixel, xPixel, yPixel
41    );
42  }
43

```

```

44  return {
45      ecranulInitializare      : initializareEcran,
46      ecranulResetare         : resetareEcran,
47      ecranulActualizarePixel : actualizarePixelEcran
48  };
49  }

```

Numărul de pixeli ai afișorului este controlat de valorile variabilelor `xNumeric` și `yNumeric` de la rândurile **11**, **12**.

La rândul **34**, masca de biți `0x0f` izolează din octetul de date returnat de funcția `memoriaCiteste()` cvartetul *drept*. La rândurile **3–6** se găsesc codurile RGB hexazecimale [8] ale celor 16 culori.

Adresele memoriei *grafice*<sup>5</sup> încep de la poziția `0x200 — 512` în baza de numerație 10 —, vezi rândul **36**.

## 2.5 Funcția Memoria()

Această funcție se găsește la rândul **50** al listării de cod din Secțiunea 2.1.

Program:> În codul următor este preparată *afișarea* conținutului memoriei pe rânduri de câte 16 octeți de date, la hexdump — *deversarea* hexazecimală —, respectiv la *monitorizarea* execuției.

```

1  function Memoria() {
2      var tablouMemorie = new Array( 0x600 );
3
4      function seteaza( adresa, valoarea ) {
5          return (tablouMemorie[ adresa ] = valoarea);
6      }
7
8      function citeste( adresa ) {
9          return (tablouMemorie[ adresa ]);
10     }
11
12     function citesteCuvantul( adresa ) {
13         return (
14             citeste( adresa )
15             + (citeste( adresa + 1 ) << 8)
16         );
17     }
18
19     function stocheazaOctetul( adresa, valoarea ) {
20         seteaza( adresa, (valoarea & 0xff) );
21         if ( (adresa >= 0x200) && (adresa <= 0x5ff) ) {
22             afisor.ecranulActualizarePixel( adresa );
23         }

```

---

<sup>5</sup> Sau memoria *ecranului* [4, pag. 131]. În *arhitectura* [3] 6502, *dispozitivele* de intrare-ieșire sunt *mapate* în memorie [70, pp. 14, 17].

```

24 }
25
26 function stocheazaApasareaPeTasta( e ) {
27     var valoarea = e.which;
28
29     memorie
30     .memoriaStocheazaOctetul( 0xff, valoarea );
31 }
32
33 function formateaza( inceputul, lungimea ) {
34     var sirulHTML = "";
35     var numarul;
36     var iteratorul;
37
38     for ( iteratorul = 0; iteratorul < lungimea;
39         iteratorul++ ) {
40         numarul = inceputul + iteratorul;
41         if ( (iteratorul & 0x0f) === 0 ) {
42             if ( iteratorul > 0 ) {
43                 sirulHTML += "\r\n";
44             }
45             sirulHTML
46             += auxiliarConvertescNumarLaHexa(
47                 (numarul >> 8) & 0xff
48             );
49             sirulHTML
50             += auxiliarConvertescNumarLaHexa(
51                 numarul & 0xff
52             );
53             sirulHTML += ": ";
54         }
55         sirulHTML
56         += auxiliarConvertescNumarLaHexa(
57             memorie.memoriaCiteste( numarul )
58         );
59         sirulHTML += " ";
60     }
61
62     return sirulHTML;
63 }
64
65 return {
66     memoriaSeteaza           : seteaza,
67     memoriaCiteste           : citeste,
68     memoriaCitesteCuvantul   : citesteCuvantul,
69     memoriaStocheazaOctetul   : stocheazaOctetul,
70     memoriaStocheazaApasarea : stocheazaApasareaPeTasta,
71     memoriaFormateaza         : formateaza
72 };
73 }

```

Memoria are 1536 de *poziții*<sup>6</sup> — 0x600 în baza de numerație 16 —, vezi rândul 2. Fiecare poziție îi corespunde unui octet de date [70, pag. 35], fapt implementat la rândul 20 cu ajutorul măștii de biți 0xff. Plaja de poziții 0x200–0x5ff — cu 1024 de elemente — îi este rezervată memoriei afișorului, vezi rândul 21.

*Cuvintele*<sup>7</sup> de memorie au câte 2 octeți și sunt stocate în formatul *little endian*<sup>8</sup>: octetul *drept* precede octetul stâng [70, pag. 34], vezi rândurile 14, 15.

Testul de la rândul 41, bazat pe masca de biți 0x0f, prefățează rândurile de câte<sup>9</sup> 16 octeți de date cu un *contor* de forma “hhhh:”, unde h desemnează o cifră hexazecimală.

La rândul 27 este folosită proprietatea dezafectată *which* [64] a evenimentului e. *Browser*-ul Firefox permite, la momentul scrierii acestui tutorial, utilizarea sa. Vezi discuția de la pagina 20.

La rândul 30 este implementată capturarea *apăsării* pe acelea dintre taste care le corepund caracterelor tipografice ASCII<sup>10</sup>. Rezultatul capturării este memorat la adresa 0xff.

## 2.6 Funcția InterfataCuUtilizatorul()

Această funcție se găsește la rândul 14 al listării de cod din Secțiunea 2.1.

Program:> În codul care urmează sunt implementate — ca obiecte JavaScript— stările aplicației Easy 6502, menționate la pagina 8, respectiv elementele *interfeței* [63] de manipulare a lor.

```

1 function InterfataCuUtilizatorul () {
2   var stareaCurenta;
3   var pornesc = {
4     asamblare      : true ,
5     executie       : [ false , "Execut\u0103" ],
6     resetare       : false ,
7     hexdump        : false ,
8     dezasamblare   : false ,
9     depanare       : [ false , false ]
10  };
11  var asamblat = {
12    asamblare       : false ,
13    executie        : [ true , "Execut\u0103" ],

```

<sup>6</sup> Celule de memorie [4, pag. 7].

<sup>7</sup> [4, pag. 25].

<sup>8</sup> Memorie *inversată* [4, pag. 9]. Pentru citirea unui cuvânt de 4 biți din *memoria fixă* — ROM —, vezi [60, pp. 118, 120]. Pentru *scrierea* Write și *citirea* Read a unui cuvânt de 1 bit în, respectiv din *memoria cu citire și înscriere* — RAM — construită cu tranzistoare MOS, vezi [60, pp. 194, 195].

<sup>9</sup> [4, pag. 7].

<sup>10</sup> [4, pag. 11].

```

14     resetare      : true ,
15     hexdump       : true ,
16     dezasamblare  : true ,
17     depanare      : [ true, false ]
18 };
19 var inExecutie = {
20     asamblare      : false ,
21     executie       : [ true, "Opre\u0219te" ],
22     resetare       : true ,
23     hexdump        : false ,
24     dezasamblare   : false ,
25     depanare       : [ true, false ]
26 };
27 var inDepanare = {
28     asamblare      : false ,
29     executie       : undefined ,
30     resetare       : true ,
31     hexdump        : true ,
32     dezasamblare   : true ,
33     depanare       : [ true, true ]
34 };
35 var postDepanare = {
36     asamblare      : false ,
37     executie       : undefined ,
38     resetare       : true ,
39     hexdump        : true ,
40     dezasamblare   : true ,
41     depanare       : [ true, false ]
42 };
43
44 function setezStarea( starea ) {
45     $nod.find( ".butonAsamblare" )
46         .attr( "disabled", !starea.asamblare );
47     if ( starea.executie ) {
48         $nod.find( ".butonExecutie" )
49             .attr( "disabled", !starea.executie[0] );
50         $nod.find( ".butonExecutie" )
51             .val( starea.executie[1] );
52     }
53     $nod.find( ".butonResetare" )
54         .attr( "disabled", !starea.resetare );
55     $nod.find( ".butonHexDump" )
56         .attr( "disabled", !starea.hexdump );
57     $nod.find( ".butonDezasamblare" )
58         .attr( "disabled", !starea.dezasamblare );
59     $nod.find( ".depaneaz" )
60         .attr( "disabled", !starea.depanare[0] );
61     $nod.find( ".depaneaz" )
62         .attr( "checked", starea.depanare[1] );
63     $nod.find( ".butonPasire" )
64         .attr( "disabled", !starea.depanare[1] );
65     $nod.find( ".butonSariLa" )
66         .attr( "disabled", !starea.depanare[1] );
67     stareaCurenta = starea;

```

```

68     }
69
70     function initializareInterfata( e ) {
71         if ( (!e) || (e && (e.code !== "Tab")) ) {
72             setezStarea( pornesc );
73         }
74     }
75
76     function pornesteInterfata() {
77         setezStarea( inExecutie );
78     }
79
80     function opresteInterfata() {
81         setezStarea( asamblat );
82     }
83
84     function pornitDepanare() {
85         setezStarea( inDepanare );
86     }
87
88     function opritDepanare() {
89         setezStarea( postDepanare );
90     }
91
92     function asamblareReusita() {
93         setezStarea( asamblat );
94     }
95
96     function monitorizareSchimbari( stare ) {
97         $nod.find( ".monitorul" )
98         .toggle( stare );
99     }
100
101     function arataInformatii() {
102         var cuprinsul = $nod
103         .find( ".mesajele code" );
104
105         if ( cuprinsul.html() ) {
106             cuprinsul.html( "" );
107         } else {
108             cuprinsul.html(
109                 $nod.find( ".informatiile" )
110                 .html()
111             );
112         }
113     }
114
115     function captureazaTaburiInEditor( e ) {
116         if ( e.code === "Tab" ) {
117             e.preventDefault();
118
119             var inceputCursor = this.selectionStart;
120             var finalCursor   = this.selectionEnd;
121             var valoareCurenta = this.value;

```

```

122
123     this.value
124     = valoareCurenta.substring( 0, inceputCursor )
125     + "\t"
126     + valoareCurenta.substring( finalCursor );
127     this.selectionStart
128     = this.selectionEnd = inceputCursor + 1;
129 }
130 }
131
132 return {
133     interfataInitializare           : initializareInterfata,
134     interfataPorneste              : pornesteInterfata,
135     interfataOpreste               : opresteInterfata,
136     interfataAsamblareReusita      : asamblareReusita,
137     interfataPornitDepanare        : pornitDepanare,
138     interfataOpritDepanare         : opritDepanare,
139     interfataMonitorizareSchimbări : monitorizareSchimbări,
140     interfataArataInformatii       : arataInformatii,
141     interfataCaptureazaTaburi      : captureazaTaburiInEditor
142 };
143 }

```

La rândurile **5**, **13** și **21** se găsesc *unicoduri* [61] ale literelor românești. Ele vor apărea, în funcție de starea aplicației, pe suprafața elementului HTML **4** din Figura 1.2.

Codul de la rândul **71** permite inserarea de Tab-uri în editorul de text fără a opri *execuția* în emulator a programului deja asamblat.

Codul de la rândurile **106**, **108** implementează *ștergerea* conținutului alternativ cu *afișarea* informațiilor în cuprinsul elementului HTML **26** din Figura 1.2.

## 2.7 Funcția Etichetele()

Această funcție se găsește la rândul **157** al listării de cod din Secțiunea 2.1.

Program:> În codul următor sunt *calculate*, dacă există, *etichetele*<sup>11</sup> programului. Calculul folosește expresii regulate și implică *înlocuirea*, în momentul asamblării, a etichetelor cu valori ale *registrlui PC*. În registrul PC<sup>12</sup> este stocată adresa celulei de memorie care *urmează* să fie accesată [70, pag. 18].

```

1 function Etichetele() {
2     var indexEtichete = [];
3
4     function indexeazaLiniile( liniile, simbolurile ) {

```

<sup>11</sup> În limba engleză, *label* (sing.).

<sup>12</sup> Acesta este *contorul program(ului)* [4, pag. 11] sau *numărătorul de instrucții* [60, pag. 199]. În limba engleză, *Program Counter*. În general, registrul este un dispozitiv numeric destinat păstrării temporare a informației și având *lungimea* unui cuvânt [11, pag. 57].

```

5   var iteratorul;
6
7   for ( iteratorul = 0; iteratorul < liniile.length;
8       iteratorul++ ) {
9       if ( !indexeazaLinia(
10          liniile[ iteratorul ], simbolurile
11          )
12      ) {
13          auxiliarAfiseazaMesajul (
14              "**Etichet&#259; deja definit&#259; la linia "
15              + ( iteratorul + 1 )
16              + " :** "
17              + liniile[ iteratorul ]
18          );
19          return false;
20      }
21  }
22  return true;
23 }
24
25 function indexeazaLinia( intrarea, simbolurile ) {
26     var valoareaCurentaAregistruluiPC = asamblor
27         .asamblorulCitesteValoareaPCcurenta();
28
29     asamblor.asamblorulFaLinia(
30         intrarea, 0, simbolurile
31     );
32
33     if ( intrarea.match( /^\w+:/ ) ) {
34         var eticheta = intrarea
35             .replace( /(^\w+):.*$/ , "$1" );
36
37         if ( gaseste( eticheta ) ) {
38             return false;
39         }
40         if ( simbolurile.cautare( eticheta ) ) {
41             auxiliarAfiseazaMesajul(
42                 "**Eticheta "
43                 + eticheta
44                 + " este deja folosit&#259; ca simbol;"
45                 + "<br>redenumi&#355;i, v&#259; rog,"
46                 + " una din ele.**"
47             );
48             return false;
49         }
50         return (
51             urca(
52                 eticheta + "|" + valoareaCurentaAregistruluiPC
53             )
54         );
55     }
56     return true;
57 }
58

```

```
59  function urca( numele ) {
60      if ( gaseste( numele ) ) {
61          return false;
62      }
63      indexEtichete.push( numele + "|" );
64      return true;
65  }
66
67  function gaseste( numele ){
68      var numeleSiAdresa;
69      var iteratorul;
70
71      for ( iteratorul = 0; iteratorul < indexEtichete.length;
72          iteratorul++ ) {
73          numeleSiAdresa = indexEtichete[ iteratorul ]
74              .split( "|" );
75          if ( numele === numeleSiAdresa[0] ) {
76              return true;
77          }
78      }
79      return false;
80  }
81
82  function seteazaRegistrulPC( numele, adresa ) {
83      var numeleSiAdresa;
84      var iteratorul;
85
86      for( iteratorul = 0; iteratorul < indexEtichete.length;
87          iteratorul++ ) {
88          numeleSiAdresa = indexEtichete[ iteratorul ]
89              .split( "|" );
90          if ( numele === numeleSiAdresa[0] ) {
91              indexEtichete[ iteratorul ] = numele + "|" + adresa;
92              return true;
93          }
94      }
95      return false;
96  }
97
98  function citesteRegistrulPC( numele ) {
99      var numeleSiAdresa;
100     var iteratorul;
101
102     for ( iteratorul = 0; iteratorul < indexEtichete.length;
103         iteratorul++ ) {
104         numeleSiAdresa = indexEtichete[ iteratorul ]
105             .split( "|" );
106         if ( numele === numeleSiAdresa[0] ) {
107             return (numeleSiAdresa[1]);
108         }
109     }
110     return -1;
111 }
112
```

```

113  function afiseazaMesajulEtichete() {
114      var cateEtichete = indexEtichete.length;
115      var sirul        = "G&#259;sit "
116          + cateEtichete + " etichet";
117
118      if ( cateEtichete !== 1 ) {
119          sirul += "e";
120      } else {
121          sirul += "&#259;";
122      }
123      sirul += ".";
124      auxiliarAfiseazaMesajul( sirul );
125  }
126
127  function reseteazaEtichete() {
128      indexEtichete = [];
129  }
130
131  return {
132      eticheteleIndexeaza : indexeazaLiniile,
133      eticheteleGaseste   : gaseste,
134      eticheteleCitestePC : citesteRegistruPC,
135      eticheteleMesaj     : afiseazaMesajulEtichete,
136      eticheteleReseteaza : reseteazaEtichete
137  };
138 }

```

Etichetele au formula

“expresie\_alfanumerică:”.

Vezi rândul 33. Expresia alfanumerică poate conține (și) numai cifre zecimale însă trebuie să cuprindă *cel puțin un* caracter. Lungimea sa maximă<sup>13</sup> este limitată doar de restricțiile de memorie impuse căutării *grupului de captură* [25] de la rândul 35.

Funcția `seteazaRegistruPC()`, de la rândurile 82 și urm., nu este folosită în versiunea de față a aplicației Easy6502.

Valorile permise argumentului simbolurile, al funcției `indexeazaLinia()`, trebuie să fie obiecte care implementează *metoda* `cautare()` [59], vezi rândul 40.

În codul prezentat la pagina 36, Subsecțiunea 2.9.2, sunt introduse *simbolurile* [75, pag. 43] despre care discutăm în codul de față la rândurile 40–49. Codul împiedică asamblarea programului care urmează.

```

define  adresa $03
adresa:
        lda     #adresa

```

---

<sup>13</sup> Asamblarea cu MASM, pentru arhitectura x86, limitează lungimea expresiei la 247 de caractere [42]. Vezi și [70, pag. 29].

În anul 1975, compania americană MOS Technology lansează 6500 Series, o suită de zece microprocesoare compatibile la nivel *software* [70, pag. 15]. Acestea dispun de același *set de instrucțiuni* și de aceeași *arhitectură* de bază. Versiuni ale microprocesoarelor au fost realizate și de companiile americane Rockwell International (R6500) și Synertek (SY6500) [70, *ibid.*], [54].

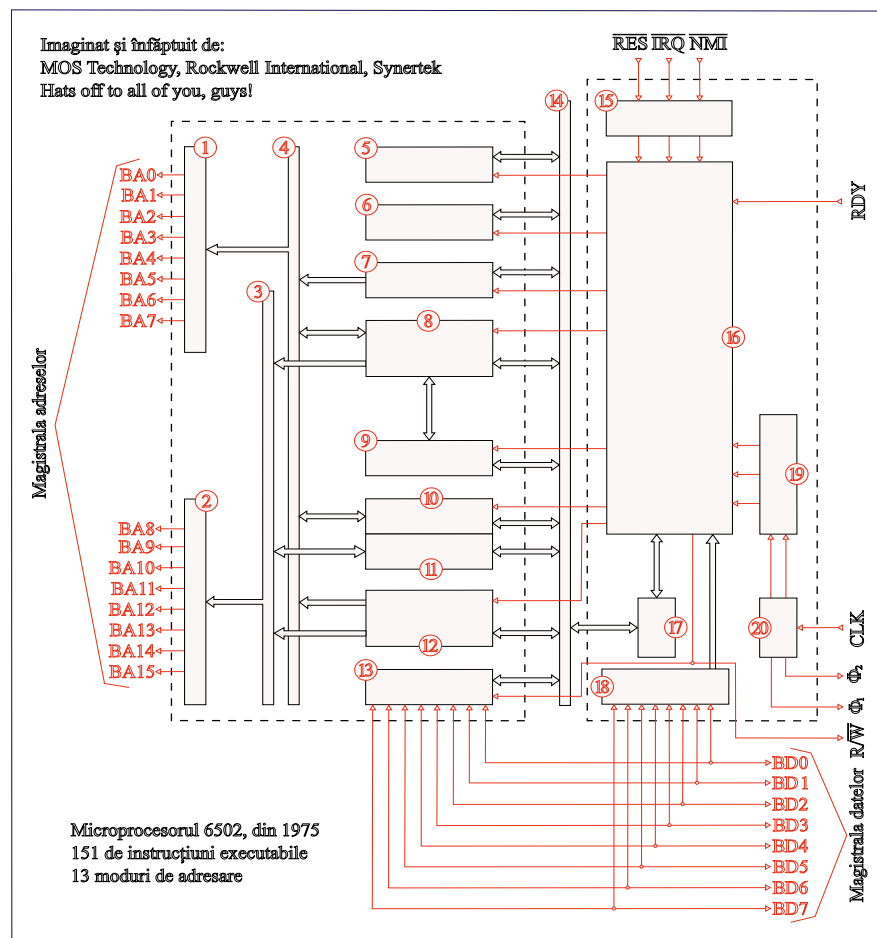


**Fig. 2.1** Schema-bloc a microprocesorului 6502 [70, pag. 16], [54].

În Figura 2.1 se găsește schema-bloc a microprocesorului 6502 din 6500 Series. Microprocesorul reprezintă *unitatea centrală de prelucrare* CPU a informațiilor.

ei de către (micro)calculator<sup>14</sup>. Ea este alcătuită, metodologic, din *dispozitivul aritmetic* și din *dispozitivul de comandă* [11, pp. 11, 12] — vezi, în ordinea anterioară, de la stânga la dreapta, cele două dreptunghiuri desenate cu linie întreruptă —.

Componentele numerotate sunt următoarele: **1** octetul inferior — *Low* — al adreselor de memorie, **2** octetul superior — *High* — al adreselor de memorie, **3** magistrala internă pentru octeții superiori de adresă, **4** magistrala internă pentru octeții inferiori de adresă, **5** registrul (de) index<sup>15</sup> *Y*, **6** registrul (de) index *X*, **7** registrul indicator de *stivă*<sup>16</sup> *SP* [60, pag. 200], **8** unitatea de calcul<sup>17</sup> aritmetic și logic *ALU*, **9** registrul acumulator *A*, **10** octetul inferior al registrului *PC*, **11** octetul superior al registrului *PC*, **12** circuitul (electronic) basculant *bistabil*<sup>18</sup> al introducerii datelor, **13** circuitul *tampon*<sup>19</sup> al *magistralei*<sup>20</sup> de date, **14** magistrala internă de date, **15** circuitele *logice* [11, pag. 16] ale *întreruperilor* — adică, opriri ale execuției în desfășurare a programului ca răspuns la *semnalele* electrice [51, pag. 14] trimise de dispozitive exterioare microprocesorului —.

Aici avem, vezi [70, pp. 20, 21], *not-RESet*, întreruperea care intervine (și) la începerea alimentării cu energie electromagnetică a circuitului de curent continuu al microprocesorului; *not-Interrupt ReQuest*, o întrerupere ce poate fi ignorată, *mascată*, în funcție de starea *fanionului*<sup>21</sup> *I* din registrul *P*; *not-Non-Maskable Interrupt*, o întrerupere ce nu poate fi mascată.

Întreruperile nu sunt implementate în această versiune a aplicației Easy6502. Astfel, putem folosi adresele de memorie ale întreruperilor: 0xffffa–0xffff [70, *ibid.*].

Mai departe, **16** decodificatorul de instrucțiuni [60, pag. 199], **17** registrul de *stare*<sup>22</sup> a procesorului *P*, **18** registrul cu *opcodul*<sup>23</sup> instrucțiunii *curente*<sup>24</sup> *IR* [70, pag. 18], **19** circuitul de control al *frecvenței de execuție*<sup>25</sup>, **20** generatorul de *tact*<sup>26</sup> [60, pag. 200] — produce *semnalul de ceas* [4, pag. 76] al microprocesorului —.

În *materializare*, variantele microprocesorului 6502 funcționează la frecvențe de 1, 2 sau 3 Mhz, ceea ce implică *durate* ale execuției instrucțiunilor de ordinul microsecundelor [70, pp. 18, 91]. Versiunea de față a aplicației Easy6502 nu implementează aceste durate, chiar dacă viteza de execuție a emulatorului poate fi modificată.

<sup>14</sup> Sau *microcomputer*.

<sup>15</sup> [4, pag. 7].

<sup>16</sup> În limba engleză, *stack*. Acesta memorează adresa *vârfului* stivei [4, *ibid.*]. Notat, în arhitectura 65xx, cu *S* [70, pag. 27].

<sup>17</sup> Adică, blocul logic pentru operații aritmetice [11, pag. 13].

<sup>18</sup> În limba engleză, *latch*.

<sup>19</sup> În limba engleză, *buffer*.

<sup>20</sup> În limba engleză, *bus*.

<sup>21</sup> În limba engleză, *flag*. Sau *indicatorului de condiții* [4, pag. 8] ori a *bistabilului de stare* [60, pag. 200].

<sup>22</sup> În limba engleză, *status*.

<sup>23</sup> Vezi pagina 60.

<sup>24</sup> În arhitectura x86 există registrul *IP* pentru *adresa* instrucțiunii *curente* [4, pag. 7].

<sup>25</sup> În limba engleză, *timing*.

<sup>26</sup> În limba engleză, *clock*.

Restul simbolurilor: **CLK** reprezintă *sursa*, exterioară microprocesorului, de *impulsuri* de tact — un *oscilator* de cristal [70, ibid.] —;  $\Phi_{1,2}$  furnizarea de semnale de *sincronizare* către dispozitivele periferice, folosind generatorul de tact cu două faze [60, pag. 176]; **RDY** semnalarea, către CPU, a *prezenței* datelor pe magistrala de date [11, pag. 15].

Legăturile între componente sunt făcute prin linii *subțiri* — cu *lățimea* de 1 bit — și *groase* — cu lățimea de 8 biți —. Liniile sunt încheiate fie printr-o singură *săgeată* — canale *unidirecționale* de transport de informații — fie prin două săgeți — canale bidirecționale —.

## 2.9 Funcția Asamblorul()

În această versiune a aplicației Easy6502, nu există posibilitatea să *salvăm* rezultatul asamblării prin apăsarea unui buton al interfeței cu utilizatorul: pe măsură ce sunt asamblate, liniile de cod sunt *încărcate*<sup>27</sup> în memorie. De asemenea, *dezasamblarea* codului-mașină hexazecimal [70, pag. 25] nu se realizează într-un obiect JavaScript separat.

### 2.9.1 Funcțiile *asambleazaCodul()* și *sanitizeazaLinia()*

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **200** al listării de cod din Secțiunea 2.1.

```

1 function asambleazaCodul () {
2   var ADRESA_BOOTSTRAP = 0x600;
3   var codul;
4   var liniile;
5   var simbolurile;
6   var iteratorul;
7   var sirul;
8
9   depasireaPlajeiRamificarii = false;
10  simulator.simulatorulReseteaza ();
11  etichete.eticheteleReseteaza ();
12  valoareaRegistruluiPCpentruCodObisnuit = ADRESA_BOOTSTRAP;
13  $nod.find ( ".mesajele code " ).empty ();
14
15  codul = $nod.find ( ".codul " ).val ();
16  codul += "\n\n";
17  liniile = codul.split ( "\n " );
18  codulAsamblatCorect = true;
19  auxiliarAfiseazaMesaajul ( "Preprocesez..." );

```

<sup>27</sup> În limba engleză, *loaded* [46], [4, pag. 83].

```

20 simbolurile = preprocesare( liniile );
21
22 auxiliarAfiseazaMesajul( "Indexez etichetele..." );
23 valoareaRegistruluiPCpentruCodObisnuit = ADRESA_BOOTSTRAP;
24 if ( !etichete.eticheteleIndexeaza(
25     liniile, simbolurile
26 )
27 ) {
28     return false;
29 }
30 etichete.eticheteleMesaj();
31
32 valoareaRegistruluiPCpentruCodObisnuit = ADRESA_BOOTSTRAP;
33 auxiliarAfiseazaMesajul( "Asamblez codul..." );
34 lungimeaCodului = 0;
35 for ( iteratorul = 0; iteratorul < liniile.length;
36     iteratorul++ ) {
37     if ( !asambleazaLinia(
38         liniile[ iteratorul ], iteratorul, simbolurile
39     )
40     ) {
41         codulAsamblatCorect = false;
42         break;
43     }
44 }
45 if ( lungimeaCodului === 0 ) {
46     codulAsamblatCorect = false;
47     auxiliarAfiseazaMesajul( "Nu am cod de executat." );
48 }
49 if ( codulAsamblatCorect ) {
50     interfata.interfataAsamblareReusita();
51     memorie.memoriaSeteaza(
52         valoareaRegistruluiPCpentruCodObisnuit, 0x00
53     );
54 } else {
55     if ( liniile[ iteratorul ] ) {
56         sirul = liniile[ iteratorul ]
57             .replace( "<", "&lt;" )
58             .replace( ">", "&gt;" );
59         if ( !depasireaPlajeiRamificarii ) {
60             auxiliarAfiseazaMesajul(
61                 "***Eroare de sintax&#259; la linia "
62                 + ( iteratorul + 1 )
63                 + ": "
64                 + sirul
65                 + "***"
66             );
67         } else {
68             auxiliarAfiseazaMesajul(
69                 "**S-a dep&#259;&#351;it "
70                 + " plaja de ramificare"
71                 + " la linia "
72                 + ( iteratorul + 1 )
73                 + "<br>(ramific&#259;rile sunt "

```

```

74         + " limitate de la -128 la +127): "
75         + sirul
76         + "**"
77     );
78 }
79 }
80 interfata.interfataInitializare();
81 return false;
82 }
83
84 auxiliarAfiseazaMesajul (
85     "Codul a fost asamblat cu succes, &#238;n "
86     + lungimeaCodului
87     + " octe"
88     + (( lungimeaCodului === 1 )? "t" : "&#355;i")
89     + ". "
90 );
91 return true;
92 }

```

Instrucțiunea de la rândul **20** produce implementarea metodei `cautare()` — vezi observația de la pagina 30 —, respectiv implică modificarea valorii variabilei `valoareaRegistruluiPCpentruCodObisnuit`.

La rândurile **57, 58**, sunt *inactivate* — dacă există — *parantezele unghiulare*<sup>28</sup> din textul de asamblat prin înlocuirea lor cu cod HTML [66].

Memoria *de utilizator* începe de la adresa `0x600`, conform rândurilor **2** și **12**. De la această adresă *în sus* [70, pag. 40] — mergând până la `0xffff`, conform discuției de pe pagina 32 — întâlnim programele asamblate în Easy6502, diversele date temporare ș.a.m.d. Deversarea hexazecimală a conținutului memoriei, respectiv dezasamblarea programelor se referă numai la acest spațiu de adrese.

*Ramificările* — vezi rândul **73** — sunt introduse la pagina 44.

Program:> În codul următor, construim funcția care se găsește la rândul **201** al listării de cod din Secțiunea 2.1.

```

1 function sanitizeazaLinia( linia ) {
2     var faraComentariiSiSpatiiGoale
3     = linia.replace( /^(.*?);.*$/, "$1" );
4
5     return (
6         faraComentariiSiSpatiiGoale
7         .replace( /\s+/, " " )
8         .replace( /\s+$/, " " )
9     );
10 }

```

Comentariile *pe un singur rând*, în textul de asamblat, se scriu la dreapta caracterului “;” [70, pag. 32], vezi rândul **3**. Prezența lui “?” în grupul de captură ne permite să folosim *de mai multe ori* [22] caracterul “;” pe același rând al textului de asamblat.

<sup>28</sup> În limba engleză, *angle bracket* (sing.).

### 2.9.2 Funcția *preprocesare()*

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **202** al listării de cod din Secțiunea 2.1.

```

1  function preprocesare( liniile ) {
2    var tabloul          = [];
3    var PREFIXUL         = "__";
4    var iteratorul;
5    var dataSePotriveste;
6
7    function cautare( cheia ) {
8      var nouaCheie = PREFIXUL + cheia;
9
10     if ( tabloul.hasOwnProperty( nouaCheie ) ) {
11       return (tabloul[ nouaCheie ]);
12     } else {
13       return undefined;
14     }
15   }
16
17   function adauga( cheia, valoarea ) {
18     var nouaCheie      = PREFIXUL + cheia;
19     var valoareaExistaDeja = tabloul
20       .hasOwnProperty( nouaCheie );
21
22     if ( !valoareaExistaDeja ) {
23       tabloul[ nouaCheie ] = valoarea;
24     }
25   }
26
27   for ( iteratorul = 0; iteratorul < liniile.length;
28     iteratorul++ ) {
29     liniile[ iteratorul ]
30       = sanitizeazaLinia( liniile[ iteratorul ] );
31     dataSePotriveste
32       = liniile[ iteratorul ]
33       .match( /^define\s+(\w+)\s+(\S+)/ );
34     if ( dataSePotriveste ) {
35       adauga(
36         dataSePotriveste[1],
37         sanitizeazaLinia( dataSePotriveste[2] )
38       );
39     liniile[ iteratorul ] = "";
40   }
41 }
42
43 return {
44   cautare : cautare
45 }
46 }
```

Asamblorul Easy6502 permite introducerea, în textul de asamblat, a unor *denumiri*<sup>29</sup> de constante, cu expresia “define numele valoarea”.

Vezi rândul **33**. Astfel, programul — nerecomandat! —

```
define lda $2
      lda #lda
```

va produce codul-mașină a9 02. Instrucțiunea de la rândul **22** face ca numele, odată atribuit, să nu poată fi redefinit. Orice redefinire va fi ștearsă, conform rândului **39**.

### 2.9.3 Funcțiile *asambleazaLinia()* și *DCB()*

Program:> În codul următor, implementăm funcția care se găsește la rândul **203** al listării de cod din Secțiunea 2.1.

```
1 function asambleazaLinia( ceIntra, numarulLiniei, simboluri ) {
2   var eticheta;
3   var comanda;
4   var parametrul;
5   var adresa;
6   var operatia;
7
8   if ( ceIntra.match( /^w+:/ ) ) {
9     eticheta = ceIntra.replace( /^(w+):.*$/ , "$1" );
10
11     if ( ceIntra.match( /^w+:[\s]*w+.*$/ ) ) {
12       ceIntra = ceIntra.replace( /^w+:[\s]*(.*)$/ , "$1" );
13       comanda = ceIntra.replace( /^(w+).*/ , "$1" );
14     } else {
15       comanda = "";
16     }
17   } else {
18     comanda = ceIntra.replace( /^(w+).*/ , "$1" );
19   }
20
21   if ( comanda === "" ) {
22     return true;
23   }
24
25   comanda = comanda.toUpperCase();
26
27   if ( ceIntra.match( /^*\s*=\s*\$?[0-9a-f]*$/ ) ) {
28     parametrul = ceIntra.replace( /^*\s*\s*=\s*/ , "" );
29
```

---

<sup>29</sup> Adică, a unor *macro*-uri [50].

```

30     if ( parametrul[0] === "$" ) {
31         parametrul = parametrul.replace( /\$/ , "" );
32         adresa = parseInt( parametrul, 16 );
33     } else {
34         adresa = parseInt( parametrul, 10 );
35     }
36
37     if ( (adresa < 0) || (adresa > 0xffff) ) {
38         auxiliarAfiseazaMesajul(
39             "Nu pot reloca niciun cod &#238;n"
40             + " afara celor 64k de memorie."
41         );
42         return false;
43     }
44
45     valoareaRegistruluiPCpentruCodObisnuit = adresa;
46     return true;
47 }
48
49 if ( ceIntra.match( /\w+\s+.*$/ ) ) {
50     parametrul = ceIntra.replace( /\w+\s+(.*)/, "$1" );
51 } else if ( ceIntra.match( /\w+$/ ) ) {
52     parametrul = "";
53 } else {
54     return false;
55 }
56
57 parametrul = parametrul.replace( /[ ]/g, "" );
58
59 if ( comanda === "DCB" ) {
60     return DCB( parametrul );
61 }
62
63 for ( operatia = 0;
64     operatia < codurileOperationale.length;
65     operatia++
66 ) {
67     if ( codurileOperationale[ operatia ][0] === comanda ) {
68         if ( verificaUnSingurOctet(
69             parametrul, codurileOperationale[ operatia ][1]
70         ) ) {
71             return true;
72         }
73         if ( verificaImediat(
74             parametrul, codurileOperationale[ operatia ][1],
75             simboluri
76         ) ) {
77             return true;
78         }
79     }
80     if ( verificaPaginaZero(
81         parametrul, codurileOperationale[ operatia ][2],
82         simboluri
83     ) ) {

```

```
84         )
85     ) {
86         return true;
87     }
88     if ( verificaPaginaZeroX(
89         parametrul, codurileOperationale[ operatia ][3],
90         simboluri
91     )
92     ) {
93         return true;
94     }
95     if ( verificaPaginaZeroY(
96         parametrul, codurileOperationale[ operatia ][4],
97         simboluri
98     )
99     ) {
100         return true;
101     }
102     if ( verificaAdresareaAbsolutaX(
103         parametrul, codurileOperationale[ operatia ][6],
104         simboluri
105     )
106     ) {
107         return true;
108     }
109     if ( verificaAdresareaAbsolutaY(
110         parametrul, codurileOperationale[ operatia ][7],
111         simboluri
112     )
113     ) {
114         return true;
115     }
116     if ( verificaIndirect(
117         parametrul, codurileOperationale[ operatia ][8],
118         simboluri
119     )
120     ) {
121         return true;
122     }
123     if ( verificaIndirectX(
124         parametrul, codurileOperationale[ operatia ][9],
125         simboluri
126     )
127     ) {
128         return true;
129     }
130     if ( verificaIndirectY(
131         parametrul, codurileOperationale[ operatia ][10],
132         simboluri
133     )
134     ) {
135         return true;
136     }
137     if ( verificaAdresareaAbsoluta(
```

```

138     parametrul, codurileOperationale[ operatia ][5],
139     simboluri
140 )
141 ) {
142     return true;
143 }
144 if ( verificaRamificare (
145     parametrul,
146     codurileOperationale[ operatia ][12]
147 )
148 ) {
149     return true;
150 }
151 }
152 }
153 return false;
154 }

```

La rândul **27**, testăm *pseudo-operația*<sup>30</sup> [49, pag. 3-109]

`*=adresă` care permite *relocarea*<sup>31</sup> codului-mașină al programului asamblat (începând de) la adresa `adresă`, exprimată într-una din bazele de numerație 16, 10. Programul *cu originea în 0x202*, de mai jos,

```

*= $202
lda #3
sta $207

```

este asamblat drept `a9 03 8d 07 02` dar nu va fi nici deversat hexazecimal nici dezasamblat corect în versiunea de față a aplicației Easy6502. Pentru a-i vedea codul-mașină hexazecimal și rezultatul execuției putem folosi *monitorizarea* execuției, respectiv butonul **17** — *Sari la adresa* —, din Figura 1.2, al *depanatorului* asamblării.

Fanionul de *căutare globală* `g` [24], de la rândul **57**, elimină *toate* spațiile goale din date. Astfel, programul<sup>32</sup>

```
lda # % 001 111 00
```

se transformă în `a9 3c`. Memorarea *pe grupuri de cifre* a numerelor mari este o practică întâlnită frecvent.

Argumentul `numarulLiniei`, de pe rândul **1**, al funcției `asambleazaLinia()` îi este *rezervat* unei versiuni viitoare a aplicației Easy6502.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **205** al listării de cod din Secțiunea 2.1.

```
1 function DCB( parametrul ) {
```

<sup>30</sup> Sau *pseudo-instrucțiunea* [4, pag. 9].

<sup>31</sup> În limba engleză, *relocation* [67].

<sup>32</sup> Pentru prefixul `%`, vezi pagina 42.

```

2  var valorile;
3  var numarul;
4  var sirul;
5  var caracterul;
6  var iteratorul;
7
8  valorile = parametrul.split( "," );
9  if ( valorile.length === 0 ) {
10     return false;
11 }
12
13 for ( iteratorul = 0; iteratorul < valorile.length;
14     iteratorul++ ){
15     sirul = valorile[ iteratorul ];
16     if ( sirul ) {
17         caracterul = sirul.substring( 0, 1 );
18         if ( caracterul === "$" ) {
19             numarul = parseInt (
20                 sirul.replace( /\$/ , "" ), 16
21             );
22             urcaOctetul( numarul );
23         } else if ( caracterul === "%" ) {
24             numarul = parseInt (
25                 sirul.replace( /^%/, "" ), 2
26             );
27             urcaOctetul( numarul );
28         } else if (
29             (caracterul >= "0")
30             && (caracterul <= "9")
31         ) {
32             numarul = parseInt ( sirul, 10 );
33             urcaOctetul( numarul );
34         } else {
35             return false;
36         }
37     }
38 }
39 return true;
40 }

```

Cifrele “0”–“9”, ale bazei de numerație 10, pot fi scrise *binar* folosind cvartete. Codificarea zecimal-binară naturală<sup>33</sup> BCD a numerelor întregi fără semn [44, pag. 6-11] presupune traducerea *fiecăreia* din cifrele acestor numere în cvartete și *ordonarea* cvartetelor după ordinea originală a cifrelor zecimale. Aplicația Easy-6502 dispune de pseudo-operația DCB — conform rândului **59** al codului funcției `asambleazaLinia()` — pentru încărcarea în memorie a mărimilor codificate zecimal-binar natural. Vezi programul de mai jos.

```

* = $608
dcb 1, $2, %11 ; alte numere intregi, fara semn,

```

<sup>33</sup> În limba engleză, *binary-coded decimal* [70, pag. 43]. Astfel, rezultatul codificării este un *cod zecimal codificat binar natural* [55, pag. 47].

; cu VIRGULA între ele...

Aici, *prefixele* \$ și % desemnează bazele de numerație 16, respectiv 2 [70, pag. 32], la rândurile **18** și **23**. Octeții 0x01, 0x02 și 0x03 ai valorilor celor trei argumente ale pseudo-operației vor fi poziționați în memoria de utilizator începând de la adresa 0x608.

### 2.9.4 Funcțiile *incearcaParsarea...()*

Program:> În codul următor, implementăm funcția care se găsește la rândul **206** al listării de cod din Secțiunea 2.1.

```

1  function incearcaParsareaOperanduluiOctet (
2    parametrul, simboluri
3  ) {
4    var valoareaCautarii;
5    var valoarea;
6    var dataSePotriveste;
7
8    if ( parametrul.match( /^\w+$/ ) ) {
9      valoareaCautarii = simboluri.cautare( parametrul );
10     if ( valoareaCautarii ) {
11       parametrul = valoareaCautarii;
12     }
13   }
14
15   dataSePotriveste = parametrul.match( /^([0-9]{1,3})$/ );
16   if ( dataSePotriveste ) {
17     valoarea = parseInt( dataSePotriveste[1], 10 );
18   }
19
20   dataSePotriveste = parametrul.match( /^\[0-9a-f]{1,2}\$/i );
21   if ( dataSePotriveste ) {
22     valoarea = parseInt( dataSePotriveste[1], 16 );
23   }
24
25   dataSePotriveste = parametrul.match( /^%([0-1]{1,8})$/ );
26   if ( dataSePotriveste ) {
27     valoarea = parseInt( dataSePotriveste[1], 2 );
28   }
29
30   if ( (valoarea >= 0) && (valoarea <= 0xff) ) {
31     return valoarea;
32   } else {
33     return -1;
34   }
35 }

```

Cuantificatorul  $\{1, 3\}$  [22], de la rândul **15**, testează dacă valoarea argumentului parametrul este un *șir* format din 1–3 caractere din setul de caractere “0”–“9”.

Fanionul `i` [24], de la rândul **20**, lasă la alegerea noastră scrierea *cu majusculă* a caracterelor din setul “a”–“f”.

*Parsarea*<sup>34</sup> unui șir de caractere ASCII preluate din textul de asamblat înseamnă<sup>35</sup> efectuarea *analizei lexicale* [2] a acestuia: asamblorul verifică dacă șirul este *numele* dat de utilizator unei constante — rândul **9** —, respectiv dacă este un *număr întreg fără semn*, scris într-una din bazele de numerație 10, 16, 2 și *reprezentabil* printr-un singur octet — vezi rândul **30** —.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **208** al listării de cod din Secțiunea 2.1.

```

1 function incelearcaParsareaOperanduluiCuvant (
2   parametrul, simboluri
3 ) {
4   var valoareaCautarii;
5   var valoarea;
6   var dataSePotriveste;
7
8   if ( parametrul.match( /^\w+$/ ) ) {
9     valoareaCautarii = simboluri.cautare( parametrul );
10    if ( valoareaCautarii ) {
11      parametrul = valoareaCautarii;
12    }
13  }
14
15  dataSePotriveste = parametrul.match( /^\${[0-9a-f]{3,4}}$/i );
16  if ( dataSePotriveste ) {
17    valoarea = parseInt( dataSePotriveste[1], 16 );
18  } else {
19    dataSePotriveste = parametrul.match( /^[0-9]{1,5}$/ );
20    if ( dataSePotriveste ) {
21      valoarea = parseInt( dataSePotriveste[1], 10 );
22    }
23  }
24
25  if ( (valoarea >= 0) && (valoarea <= 0xffff) ) {
26    return valoarea;
27  } else {
28    return -1;
29  }
30 }

```

<sup>34</sup> În limba engleză, *parsing*.

<sup>35</sup> A se citi *începe cu*. Vezi [75, pag. 153].

### 2.9.5 Funcțiile verifica...()

Program:> În codul următor, implementăm funcția care se găsește la rândul **210** al listării de cod din Secțiunea 2.1.

```

1 function verificaRamificare( parametrul, codOperational ) {
2   var adresa;
3   var distanta;
4
5   if ( codOperational === null ) {
6     return false;
7   }
8
9   adresa = -1;
10  if ( parametrul.match( /\w+/ ) ) {
11    adresa = etichete.eticheteleCitestePC( parametrul );
12  }
13  if ( adresa === -1 ){
14    urcaCuvantul( 0x00 );
15    return false;
16  }
17
18  urcaOctetul( codOperational );
19
20  distanta = adresa - 1
21    - valoareaRegistruluiPCpentruCodObisnuit;
22  if ( (distanta < -128) || (distanta > 127) ) {
23    depasireaPlajeiRamificarii = true;
24    return false;
25  }
26
27  urcaOctetul( distanta );
28  return true;
29 }
```

*Salturile, absolute*<sup>36</sup> — JMP — sau *relative*<sup>37</sup> [4, pag. 40] — *ramificări*<sup>38</sup> BRA —, sunt abateri de la execuția secvențială [70, pag. 22] a instrucțiunilor programului de asamblat. Salturile relative se realizează cu *deplasamente*<sup>39</sup> reprezentabile printr-un octet — vezi rândul **22** — și sunt considerate *adresări ale memoriei*<sup>40</sup> [70, pag. 40]. În exemplul dat în continuare, saltul este produs de instrucțiunea BCC și este *condiționat* [70, pag. 55] de resetarea fanionului *de transport*<sup>41</sup> C.

---

bcc salt

<sup>36</sup> În limba engleză, *jump* (sing.). Vezi pagina 47.

<sup>37</sup> În limba engleză, *branch* (sing.).

<sup>38</sup> Cele opt ramificări din arhitectura de instrucțiuni 6502 sunt BCC, BCS, BEQ, BMI, BNE, BPL, BVC și BVS [70, pag. 28].

<sup>39</sup> În limba engleză, *displacement* (sing.). Sau *pe distanțe*.

<sup>40</sup> Vezi pagina 46.

<sup>41</sup> În limba engleză, *carry* [70, pag. 19], [11, pag. 69].

```

fara_salt:
    lda #$3
salt:
    ldx #$2

```

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **212** al listării de cod din Secțiunea 2.1.

```

1 function verificaImediat (
2   parametrul, codOperational, simboluri
3 ) {
4   var eticheta;
5   var supInf;
6   var adresa;
7   var dataSePotriveste;
8   var operandul;
9
10  if ( codOperational === null ) {
11    return false;
12  }
13
14  dataSePotriveste = parametrul.match( /^#([\w\%]+)$/i );
15  if ( dataSePotriveste ) {
16    operandul = incercaParsareaOperanduluiOctet (
17      dataSePotriveste[1], simboluri
18    );
19    if ( operandul >= 0 ) {
20      urcaOctetul( codOperational );
21      urcaOctetul( operandul );
22      return true;
23    }
24  }
25
26  if ( parametrul.match( /^#[<>]\w+$/ ) ) {
27    eticheta = parametrul.replace( /^#[<>](\w+)$/ , "$1" );
28    supInf = parametrul.replace( /^#([<>]).*$/ , "$1" );
29    urcaOctetul( codOperational );
30    if ( etichete.eticheteleGaseste( eticheta ) ) {
31      adresa = etichete.eticheteleCitestePC( eticheta );
32      switch ( supInf ) {
33        case ">":
34          urcaOctetul( (adresa >> 8) & 0xff );
35          return true;
36        case "<":
37          urcaOctetul( adresa & 0xff );
38          return true;
39        default:
40          return false;
41      }
42    } else {
43      urcaOctetul( 0x00 );
44      return true;

```

```

45     }
46 }
47 return false;
48 }

```

Datele folosite în programul de executat sunt deținute de (diversele tipuri de) *memorie* [4, pag. 9]. Pentru a le localiza, ne *adresăm* acesteia [4, pag. 7]. *Adresarea memoriei*<sup>42</sup> este, așadar, precizarea în textul de asamblat a modalității de localizare a datelor programului.

Prin adresarea *imediată*<sup>43</sup> a datelor `Imm` înțelegem inserarea în cod a însăși valorii operanzilor. Datele adresabile imediat au lungimea de 1 octet și sunt precedate de caracterul `#`. Conform rândului **14**, valorile numerice pot fi scrise în bazele de numerație 10, 16 și 2.

Deoarece adresele se reprezintă prin *cuvinte* de memorie, caracterele “<” și “>” sunt folosite pentru a desemna octetul *inferior* `L` al reprezentării, respectiv octetul *superior* `H` al acesteia [26, pag. 318]. Vezi rândul **28**. Un exemplu de utilizare a unei astfel de localizări a datelor este listat în continuare.

```

nop
nop
incarcare:
    lda #2
    ldx #>incarcare
    ldy #<incarcare

```

Program:> În codul următor, implementăm funcția care se găsește la rândul **214** al listării de cod din Secțiunea 2.1.

```

1 function verificaIndirect (
2   parametrul, codOperational, simboluri
3 ) {
4   var dataSePotriveste;
5   var operandul;
6
7   if ( codOperational === null ) {
8     return false;
9   }
10
11   dataSePotriveste = parametrul.match( /^\\(((\\w\\$)+)\\)\\$/i );
12   if ( dataSePotriveste ) {
13     operandul = incearcaParsareaOperanduluiCuvant (
14       dataSePotriveste[1], simboluri
15     );
16     if ( operandul >= 0 ) {
17       urcaOctetul( codOperational );
18       urcaCuvantul( operandul );
19       return true;

```

<sup>42</sup> În limba engleză, *addressing*.

<sup>43</sup> În limba engleză, *immediate*. Vezi [70, pag. 33], [4, pag. 22].

```

20     }
21 }
22 return false;
23 }

```

Adresarea *absolută indirectă* IND este folosită de instrucțiunea JMP [70, pag. 35] pentru a citi octetul inferior L al unei adrese de memorie. Acest octet se găsește la adresa dată de operandul instrucțiunii JMP. Cel de-al doilea octet necesar, octetul H, va fi preluat de la adresa care *îi urmează* adresei din operand. Concatenând cei doi octeți, obținem o adresă de memorie care va fi depozitată în registrul PC. În exemplul de mai jos, microprocesorul va executa instrucțiunea 0x0a, stocată la adresa 0x000a.

```

; opcodul pentru ASL A este 0x0a
lda #$0a      ; in A: 00001010 (0x0a)
sta $000a
jmp ($000a)   ; in A: 00010100 (0x14)

```

La rândul 11, caracterele speciale “(”, “\$”, “)” sunt *dezactivate*<sup>44</sup>.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul 216 al listării de cod din Secțiunea 2.1.

```

1 function verificaIndirectX(
2   parametrul, codOperational, simboluri
3 ) {
4   var dataSePotriveste;
5   var operandul;
6
7   if (codOperational === null ) {
8     return false;
9   }
10
11  dataSePotriveste = parametrul.match( /^\\(((\\w\\$]+),X\\)$\\i );
12  if ( dataSePotriveste ) {
13    operandul = incercareParsareaOperanduluiOctet(
14      dataSePotriveste[1], simboluri
15    );
16    if ( operandul >= 0 ) {
17      urcaOctetul( codOperational );
18      urcaOctetul( operandul );
19      return true;
20    }
21  }
22  return false;
23 }

```

Adresarea *indirectă indexată* a memoriei [70, pag. 38] INDX combină adresarea indirectă — vezi pagina 47 — cu adresarea indexată în raport cu registrul X a paginii zero de memorie — vezi pagina 52 —. În exemplul dat în continuare, operandul

<sup>44</sup> În limba engleză, *escape* [23].

instrucțiunii LDA este o adresă de pagina zero. Valoarea depozitată la această adresă va fi încărcată în registrul acumulator A.

```
ldx #$b4
ldy #$12
stx $75
sty $76
sty $12b4
ldx #$4b
lda ($2a,X)
```

Program:> În codul următor, implementăm funcția care se găsește la rândul **218** al listării de cod din Secțiunea 2.1.

```
1 function verificaIndirectY(
2   parametrul, codOperational, simboluri
3 ) {
4   var dataSePotriveste;
5   var operandul;
6
7   if ( codOperational === null ) {
8     return false;
9   }
10
11   dataSePotriveste = parametrul.match( /^\\((\\w\\$)+\\),Y$/i );
12   if ( dataSePotriveste ) {
13     operandul = incercaParsareaOperanduluiOctet(
14       dataSePotriveste[1], simboluri
15     );
16     if ( operandul >= 0 ) {
17       urcaOctetul( codOperational );
18       urcaOctetul( operandul );
19       return true;
20     }
21   }
22   return false;
23 }
```

Adresarea *indexată indirectă* a memoriei [70, pag. 39] INDY procedează *invers* ca adresarea indirectă indexată — vezi pagina 47 —:

```
ldx #$b4
stx $2a
ldx #$12
stx $2b
ldx #$ea
stx $12ff
ldy #$4b
lda ($2a),Y
```

Aici, operandul instrucțiunii LDA conține o adresă de pagina zero.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul 220 al listării de cod din Secțiunea 2.1.

```

1 function verificaUnSingurOctet (
2   parametrul, codOperational
3 ) {
4   if ( codOperational === null ) {
5     return false;
6   }
7   if (
8     (parametrul) && (parametrul !== "")
9     && (String ( parametrul ) .toUpperCase () !== "A")
10  ) {
11    return false;
12  }
13
14  urcaOctetul ( codOperational );
15  return true;
16 }

```

Adresarea *implicită*<sup>45</sup> și adresarea *la acumulator* [70, pag. 40] [4, pag. 22] SNGL produc 1 octet de cod-mașină. Prima nu necesită operanzi, vezi instrucțiunea TAX din exemplul de mai jos, cealaltă — ASL — are drept operand numele registrului acumulator A.

```

lda #6 ; cod binar: 0110
asl a  ; cod binar: 1100
tax

```

Program:> În codul următor, implementăm funcția care se găsește la rândul 222 al listării de cod din Secțiunea 2.1.

```

1 function verificaPaginaZero (
2   parametrul, codOperational, simboluri
3 ) {
4   var operandul;
5
6   if ( codOperational === null ) {
7     return false;
8   }
9   operandul = incearcaParsareaOperanduluiOctet (
10    parametrul, simboluri
11  );
12  if ( operandul >= 0 ) {
13    urcaOctetul ( codOperational );
14    urcaOctetul ( operandul );
15    return true;

```

---

<sup>45</sup> În limba engleză, *implied* [70, pag. 34].

```

16 }
17 return false;
18 }

```

Adresarea *paginii zero* a memoriei ZP se produce atunci când operandul-adresă are lungimea de 1 octet [70, pag. 34]. Constituie un caz particular de adresare absolută — vezi pagina 54 —. Iată un exemplu, mai jos, la instrucțiunea STA.

```

define adresa $aa
    lda    #$5
    sta    adresa

```

Pagina zero a memoriei este spațiul adreselor 0x0000–0x00ff. Formula acestora, 0x00hh — unde h desemnează o cifră hexazecimală —, ne permite *economisirea* unui octet de date [70, ibid.]: operandul este octetul 0xhh.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **224** al listării de cod din Secțiunea 2.1.

```

1 function verificaAdresareaAbsolutaX(
2   parametrul, codOperational, simboluri
3 ) {
4   var adresa;
5   var dataSePotriveste;
6   var operandul;
7
8   if ( codOperational === null ) {
9     return false;
10  }
11
12  dataSePotriveste = parametrul.match( /^([\w\$\$]+),X$/i );
13  if ( dataSePotriveste ) {
14    operandul = incercaParsareaOperanduluiCuvant(
15      dataSePotriveste[1], simboluri
16    );
17    if ( operandul >= 0 ) {
18      urcaOctetul( codOperational );
19      urcaCuvantul( operandul );
20      return true;
21    }
22  }
23
24  if ( parametrul.match( /^[\w\$,X$/i ] ) ) {
25    parametrul = parametrul.replace( /,X$/i, " " );
26    urcaOctetul( codOperational );
27    if ( etichete.eticheteleGaseste( parametrul ) ) {
28      adresa = etichete.eticheteleCitestePC( parametrul );
29      if ( (adresa < 0) || (adresa > 0xffff) ) {
30        return false;
31      }
32      urcaCuvantul( adresa );
33      return true;
34    } else {

```

```

35     urcaCuvantul( 0xffff );
36     return true;
37 }
38 }
39 return false;
40 }

```

Adresarea *indexată* — cu registrul X — *absolută* a memoriei [70, pag. 36] ABSX modifică adresarea absolută — vezi pagina 54 —: adresei din operand i se adaugă valoarea depozitată în registrul X, ca în exemplul dat în continuare.

```

define adresa $607
    ldX    #$03
    stX    $60a
    lda    adresa,X

```

Program:> În codul următor, implementăm funcția care se găsește la rândul **226** al listării de cod din Secțiunea 2.1.

```

1  function verificaAdresareaAbsolutaY(
2  parametrul, codOperational, simboluri
3  ) {
4      var adresa;
5      var dataSePotriveste;
6      var operandul;
7
8      if ( codOperational === null ) {
9          return false;
10     }
11
12     dataSePotriveste = parametrul.match( /^([\w\$\$]+),Y$/i );
13     if ( dataSePotriveste ) {
14         operandul = incearcaParsareaOperanduluiCuvant(
15             dataSePotriveste[1], simboluri
16         );
17         if ( operandul >= 0 ) {
18             urcaOctetul( codOperational );
19             urcaCuvantul( operandul );
20             return true;
21         }
22     }
23
24     if ( parametrul.match( /^[\w\$\$]+,Y$/i ) ) {
25         parametrul = parametrul.replace( /,Y$/i, "" );
26         urcaOctetul( codOperational );
27         if ( etichete.eticheteleGaseste( parametrul ) ) {
28             adresa = etichete.eticheteleCitestePC( parametrul );
29             if ( (adresa < 0) || (adresa > 0xffff) ) {
30                 return false;
31             }
32             urcaCuvantul( adresa );
33             return true;

```

```

34     } else {
35         urcaCuvantul( 0xffff );
36         return true;
37     }
38 }
39 return false;
40 }

```

Aici este implementată adresarea *indexată absolută*, față de registrul Y, a memoriei ABSY.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **228** al listării de cod din Secțiunea 2.1.

```

1 function verificaPaginaZeroX(
2     parametrul, codOperational, simboluri
3 ) {
4     var dataSePotriveste;
5     var operandul;
6
7     if ( codOperational === null ) {
8         return false;
9     }
10
11     dataSePotriveste = parametrul.match( /^([\w\$\$]+),X$/i );
12     if ( dataSePotriveste ) {
13         operandul = incearcaParsareaOperanduluiOctet(
14             dataSePotriveste[1], simboluri
15         );
16         if ( operandul >= 0 ) {
17             urcaOctetul( codOperational );
18             urcaOctetul( operandul );
19             return true;
20         }
21     }
22     return false;
23 }

```

Adresarea *indexată* — cu registrul X — a *paginii zero* de memorie [70, pag. 37] ZPX modifică adresarea paginii zero — vezi pagina 50 —: adresei din operand i se adaugă valoarea depozitată în registrul X, ca în exemplul dat mai jos. *Nu poate fi depășită pagina zero*, deci adresele sunt calculate *modulo 256*.

```

ldx #6
stx $05
lda $ff,X

```

Program:> În codul următor, implementăm funcția care se găsește la rândul **230** al listării de cod din Secțiunea 2.1.

```

1 function verificaPaginaZeroY(

```

```

2 parametrul, codOperational, simboluri
3 ) {
4     var dataSePotriveste;
5     var operandul;
6
7     if ( codOperational === null ) {
8         return false;
9     }
10
11     dataSePotriveste = parametrul.match( /^([\w\$\$]+),Y$/i );
12     if ( dataSePotriveste ) {
13         operandul = inearcaParsareaOperanduluiOctet(
14             dataSePotriveste[1], simboluri
15         );
16         if ( operandul >= 0 ) {
17             urcaOctetul( codOperational );
18             urcaOctetul( operandul );
19             return true;
20         }
21     }
22     return false;
23 }

```

Aici este implementată adresarea *indexată a paginii zero* de memorie, față de registrul Y, ZPY.

Program:> În codul care urmează, implementăm funcția care se găsește la rândul **232** al listării de cod din Secțiunea 2.1.

```

1 function verificaAdresareaAbsoluta(
2     parametrul, codOperational, simboluri
3 ) {
4     var adresa;
5     var dataSePotriveste;
6     var operandul;
7
8     if ( codOperational === null ) {
9         return false;
10    }
11
12    dataSePotriveste = parametrul.match( /^([\w\$\$]+)$/i );
13    if ( dataSePotriveste ) {
14        operandul = inearcaParsareaOperanduluiCuvant(
15            dataSePotriveste[1], simboluri
16        );
17        if ( operandul >= 0 ) {
18            urcaOctetul( codOperational );
19            urcaCuvantul( operandul );
20            return true;
21        }
22    }
23
24    if ( parametrul.match( /^~\w+$/ ) ) {
25        urcaOctetul( codOperational );

```

```

26     if ( etichete.eticheteleGaseste( parametrul ) ) {
27         adresa = etichete.eticheteleCitestePC( parametrul );
28         if ( (adresa < 0) || (adresa > 0xffff) ) {
29             return false;
30         }
31         urcaCuvantul( adresa );
32         return true;
33     } else {
34         urcaCuvantul( 0xffff );
35         return true;
36     }
37 }
38 return false;
39 }

```

Adresarea *absolută*<sup>46</sup> a memoriei ABS se face inserând adresa celulei de memorie care ne interesează, din plaja de valori 0x0000–0xffff — adică, 65536 de poziții, spațiu la care ne referim drept *cei 64K de memorie* [70, pag. 17] —, în textul de asamblat. Ea este folosită de cele două instrucțiuni STA din exemplul dat mai jos.

```

        lda #$2
        nop
adresa:
        nop
        nop
        sta adresa
        sta 608      ; codul hexa: $0260, cea de-a
                    ; 609-a celula de memorie...

```

### 2.9.6 Funcțiile *urca...()*, *...FereastraPopUp()* și *hexDumpAsamblorului()*

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile 234–238 ale listării de cod din Secțiunea 2.1.

```

1 function urcaOctetul( valoarea ) {
2     memorie.memoriaSeteaza(
3         valoareaRegistruluiPCpentruCodObisnuit,
4         (valoarea & 0xff)
5     );
6     valoareaRegistruluiPCpentruCodObisnuit++;
7     lungimeaCodului++;
8 }
9

```

---

<sup>46</sup> În limba engleză, *absolute* [70, pag. 33]. Sau *directă* [4, pag. 22].

```

10 function urcaCuvantul( valoarea ) {
11     urcaOctetul( valoarea & 0xff );
12     urcaOctetul( (valoarea >> 8) & 0xff );
13 }
14
15 function deschideFereastrăPopUp(
16     continutul, titlul
17 ) {
18     var fereastră;
19     var textulHTML;
20
21     fereastră = window
22     .open(
23         "",
24         titlul,
25         (
26             "width=500,"
27             + "height=300,"
28             + "resizable=yes,"
29             + "scrollbars=yes,"
30             + "toolbar=no,"
31             + "location=no,"
32             + "menubar=no,"
33             + "status=no"
34         )
35     );
36     textulHTML = "<html><head>"
37     + "<link href=\"fisiere/stilul/stil.css\" \"
38     + \" rel=\"stylesheet\" type=\"text/css\">"
39     + "<title >"
40     + titlul
41     + "</title></head><body>"
42     + "<pre><code>"
43     + continutul
44     + "</code></pre></body></html>";
45     fereastră.document.write( textulHTML );
46     fereastră.document.close();
47 }
48
49 function hexDumpAsamblorul() {
50     deschideFereastrăPopUp(
51         (memorie
52             .memoriaFormateaza(
53                 0x600, lungimeaCodului
54             )
55         ),
56         "Hexdump"
57     );
58 }

```

Numărul *magic* [56] 0x600, de la rândul **53**, a intervenit în textul programului de față și la paginile 22 — linia **2** de cod —, respectiv 33 — tot linia **2** —. El repre-

zintă începutul spațiului *contiguu*<sup>47</sup> al memoriei de utilizator, 0x0600–0xffff. Vezi rândul 6 și comentariul de la pagina 35. Raportându-ne la arhitectura x86, putem considera că avem un *model de memorie* tiny și că microprocesorul 6502 funcționează în *modul* real [4, pp. 18, 25].

Funcția `hexDumpAsamblorul()` va produce, la acționarea butonului 6 din Figura 1.2, *afișarea* conținutului acestui spațiu de memorie în cadrul unei ferestre de *browser* separate de fereastra principală a aplicației Easy6502. Vezi Figura 0.5.

### 2.9.7 Funcțiile *citesteModulSiCodul()* și *dezasambleazaCodul()*

Program:> În codul care urmează, implementăm funcțiile care se găsesc la rândurile 239 și 241 ale listării de cod din Secțiunea 2.1.

```

1  function citesteModulSiCodul( octetul ) {
2    var indexul;
3    var linia    = codurileOperationale
4      .filter (
5        function ( linia ) {
6          var indexulPosibil = linia.indexOf( octetul );
7
8          if ( indexulPosibil > -1 ) {
9            indexul = indexulPosibil;
10           return true;
11         }
12       }
13     )[0];
14
15     if ( !(linia) ) {
16       return {
17         gasitCodulOperational : "???",
18         gasitModul            : "SNGL"
19       };
20     } else {
21       return {
22         gasitCodulOperational : linia[0],
23         gasitModul            : modurileDeAdresare[ indexul ]
24       };
25     }
26   }
27
28   function dezassembleazaCodul() {
29     var adresaDeStart      = 0x600;
30     var adresaCurenta      = adresaDeStart;
31     var adresaFinala       = adresaDeStart
32       + lungimeaCodului;
33     var instructiuniDezasamblare = [];

```

---

<sup>47</sup> Sau *liniar* [4, pag. 50].

```

34  var lungimeDezasamblare;
35  var instructiuneDezasamblare;
36  var octetDezasamblare;
37  var modSiCodDezasamblare;
38  var textHTML;
39  var iteratorul;
40
41  while (adresaCurenta < adresaFinala ) {
42      instructiuneDezasamblare
43      = creeazaInstructiunea( adresaCurenta );
44      octetDezasamblare = memorie
45      .memoriaCiteste( adresaCurenta );
46      instructiuneDezasamblare
47      .adaugaOctetul( octetDezasamblare );
48      modSiCodDezasamblare
49      = citesteModulSiCodul( octetDezasamblare );
50      lungimeDezasamblare = lungimileInstructiunilor[
51      ( modSiCodDezasamblare.gasitModul )
52      ];
53      instructiuneDezasamblare
54      .seteazaModulSiCodul( modSiCodDezasamblare );
55
56      for ( iteratorul = 1; iteratorul < lungimeDezasamblare;
57          iteratorul++
58      ) {
59          adresaCurenta++;
60          octetDezasamblare = memorie
61          .memoriaCiteste( adresaCurenta );
62          instructiuneDezasamblare
63          .adaugaOctetul( octetDezasamblare );
64          instructiuneDezasamblare
65          .adaugaArgumentul( octetDezasamblare );
66      }
67
68      instructiuniDezasamblare
69      .push( instructiuneDezasamblare );
70      adresaCurenta++;
71  }
72
73  textHTML = "Adresa  Hexdump  Dezasamblat\n"
74  + "-----\n"
75  + (instructiuniDezasamblare.join( "\n" ));
76  deschideFereastrăPopUp( textHTML, "Dezasambleaz&#259;" );
77  }

```

La rândurile 4–13, folosind funcția `filter()` [36] a bibliotecii jQuery, găsim — dacă există — *mnemonica*<sup>48</sup> [70, pag. 23] din arhitectura 65xx la care se referă codul-mașină hexazecimal dat de valoarea argumentului `octetul`. Dacă *mnemonica* nu este găsită, atunci în textul *dezasamblat* va apărea trigramă<sup>49</sup> “???”, vezi rândul 17.

<sup>48</sup> Aici, o abreviere de trei litere.

<sup>49</sup> În limba engleză, *trigraph* [13].

### 2.9.8 Funcția creeazaInstructiunea()

Program:> În codul următor, implementăm funcția care se găsește la rândul **240** al listării de cod din Secțiunea 2.1.

```

1  function creeazaInstructiunea( adresa ) {
2      var octetii          = [];
3      var codOperational;
4      var argumentele      = [];
5      var modul;
6
7      function esteInstructiuneDeAcumulator() {
8          var octetiiDeAcumulator = [0x0a, 0x2a, 0x4a, 0x6a];
9
10         if ( octetiiDeAcumulator.indexOf( octetii[0] ) > -1 ) {
11             return true;
12         }
13     }
14
15     function esteInstructiuneDeRamificare() {
16         return (
17             ( codOperational.match( /^B/ ) )
18             && ( !(
19                 (codOperational === "BIT")
20                 || (codOperational === "BRK")
21             ) )
22         );
23     }
24
25     function formateazaArgumentele() {
26         var sirulArgumentelor = argumentele
27             .map( auxiliarConvertescNumarLaHexa )
28             .reverse()
29             .join( " " );
30
31         if ( esteInstructiuneDeRamificare() ) {
32             var destinatia = adresa + 2;
33
34             if ( argumentele[0] > 0x7f ) {
35                 destinatia -= 0x100 - argumentele[0];
36             } else {
37                 destinatia += argumentele[0];
38             }
39
40             sirulArgumentelor
41                 = auxiliarConvertescAdresaLaHexa( destinatia );
42         }
43
44         if ( sirulArgumentelor ) {
45             sirulArgumentelor = "$" + sirulArgumentelor;
46         }
47     }

```

```

48     if ( modul === "Imm" ) {
49         sirulArgumentelor = "#" + sirulArgumentelor;
50     }
51
52     if ( modul.match( /X$/ ) ) {
53         sirulArgumentelor += ",X";
54     }
55
56     if ( modul.match( /^IND/ ) ) {
57         sirulArgumentelor = "(" + sirulArgumentelor + ")";
58     }
59
60     if ( modul.match( /Y$/ ) ) {
61         sirulArgumentelor += ",Y";
62     }
63
64     if ( esteInstructiuneDeAcumulator() ) {
65         sirulArgumentelor = "A";
66     }
67
68     return sirulArgumentelor;
69 }
70
71 return {
72     adaugaOctetul      :
73     function ( octetul ) {
74         octetii.push( octetul );
75     },
76     seteazaModulSiCodul :
77     function ( modulSiCodul ) {
78         codOperational = modulSiCodul.gasitCodulOperational;
79         modul = modulSiCodul.gasitModul;
80     },
81     adaugaArgumentul    :
82     function ( argumentul ) {
83         argumentele.push( argumentul );
84     },
85     toString           :
86     function () {
87         var sirulOctetilor = octetii
88             .map( auxiliarConvertescNumarLaHexa )
89             .join( " " );
90         var departajare    = new Array(
91             11 - (sirulOctetilor.length)
92         )
93             .join( " " );
94
95         return (
96             "$"
97             + auxiliarConvertescAdresaLaHexa( adresa )
98             + " "
99             + sirulOctetilor
100             + departajare
101             + codOperational

```

```

102         + " "
103         + formateazaArgumentele ()
104     );
105 }
106 };
107 }

```

La rândul **8** se găsesc codurile-mașină hexazecimale ale unor instrucțiuni executabile care sunt listate în Subsecțiunea 2.10.10 cu formatul “*ihh*”,

unde *h* desemnează o cifră hexazecimală. Ne referim la aceste coduri-mașină cu apelativul *opcoduri*<sup>50</sup>.

La rândurile **17–21** sunt căutate cele opt mnemonice referitoare la salturi relative. Ele încep cu caracterul *B*, vezi pagina 44.

La rândurile **34–35**, forțăm *rămânerea*<sup>51</sup> deplasamentului în plaja de valori zecimale  $[-128, 127]$ . În arhitectura 6502, numerele negative sunt reprezentate folosind *complementul lui 2*, adică [70, pag. 42]

$$-n \equiv 256 - n \pmod{256}, \quad n \in [0, 128].$$

La rândurile **85–105** realizăm *suprascrierea*<sup>52</sup> metodei JavaScript `toString()` [33]. Metoda este apelată la rândul **103**, atunci când se solicită o reprezentare de tip *șir de caractere alfanumerice* a obiectului JavaScript omonim, respectiv la rândul **75** al codului-sursă comun al funcțiilor `citesteModulSiCodul()` și `dezasambleazaCodul()`.

### 2.9.9 Codurile operaționale

Program:> În codul care urmează, *inițializăm* variabila `codurileOperationale` de la rândul **181** al listării de cod din Secțiunea 2.1. Tablourile componente — 56 la număr [70, pag. 27] —, cu câte 13 elemente, cuprind șirul de trei majuscule ASCII al mnemonicelor arhitecturii 6502, respectiv codurile-mașină hexazecimale care corespund *asocierii* dintre mnemonica așezată pe prima poziție și fiecare<sup>53</sup> din cele 13 tipuri de adresare a memoriei. Evident, nicio mnemonică nu poate fi asociată cu toate tipurile de adresare.

Penultima mnemonică, *WDM*, nu face parte din arhitectura 65xx, fiind o instrucțiune *ajutătoare*, vezi comentariul de la pagina 99.

<sup>50</sup> În limba engleză, *operation code* (sing.) [70, pp. 23, 28]. Sau *coduri operaționale*, respectiv *coduri de operație* [60, pag. 201].

<sup>51</sup> În limba engleză, *wrapping around*  $0 \times 100$  [12].

<sup>52</sup> În limba engleză, *overriding* [79].

<sup>53</sup> De fapt, doar 12 poziții îi urmează mnemonice în tabloul component, deoarece modurile de adresare implicit și cel de adresare la acumulator au fost reunite sub titulatura de *SNGL*.

| Nr. | Mnemonică | Semnificație                    |
|-----|-----------|---------------------------------|
| 01  | ADC       | Aduna la A cu fanionul C        |
| 02  | AND       | SI între Memorie și A           |
| 03  | ASL       | Deplasare la Stanga în A        |
| 04  | BCC       | Salt relativ la resetarea lui C |
| 05  | BCS       | Salt relativ la setarea lui C   |
| 06  | BEQ       | Salt relativ când obțin Zero    |
| 07  | BIT       | Testează bitul de Memorie cu A  |
| 08  | BMI       | Salt relativ când obțin Minus   |
| 09  | BNE       | Salt relativ când nu obțin Zero |
| 10  | BPL       | Salt relativ când obțin Plus    |
| 11  | BRK       | Oprire forțată                  |
| 12  | BVC       | Salt relativ la resetarea lui V |
| 13  | BVS       | Salt relativ la setarea lui V   |
| 14  | CLC       | Resetează fanionul C            |
| 15  | CLD       | Resetează Modul Zecimal-Binar   |
| 16  | CLI       | Resetează fanionul I            |
| 17  | CLV       | Resetează fanionul V            |
| 18  | CMP       | Compara Memorie cu A            |
| 19  | CPX       | Compara Memorie cu X            |
| 20  | CPY       | Compara Memorie cu Y            |
| 21  | DEC       | Descrește Memorie cu Unu        |
| 22  | DEX       | Descrește X cu Unu              |
| 23  | DEY       | Descrește Y cu Unu              |
| 24  | EOR       | XOR între Memorie și A          |
| 25  | INC       | Creste Memorie cu Unu           |
| 26  | INX       | Creste X cu Unu                 |
| 27  | INY       | Creste Y cu Unu                 |
| 28  | JMP       | Salt absolut                    |
| 29  | JSR       | Salt absolut la Subrutina       |
| 30  | LDA       | Incarcă A cu Memorie            |
| 31  | LDX       | Incarcă X cu Memorie            |
| 32  | LDY       | Incarcă Y cu Memorie            |

|         |     |                             |         |
|---------|-----|-----------------------------|---------|
| 33      | LSR | Deplasare logica la Dreapta |         |
| +-----+ |     |                             | +-----+ |
| 34      | NOP | Nicio operatie              |         |
| +-----+ |     |                             | +-----+ |
| 35      | ORA | SAU intre Memorie si A      |         |
| +-----+ |     |                             | +-----+ |
| 36      | PHA | Urca A pe Stiva             |         |
| +-----+ |     |                             | +-----+ |
| 37      | PHP | Urca registrul P pe Stiva   |         |
| +-----+ |     |                             | +-----+ |
| 38      | PLA | Coboara A de pe Stiva       |         |
| +-----+ |     |                             | +-----+ |
| 39      | PLP | Coboara P de pe Stiva       |         |
| +-----+ |     |                             | +-----+ |
| 40      | ROL | Roteste la Stanga           |         |
| +-----+ |     |                             | +-----+ |
| 41      | ROR | Roteste la Dreapta          |         |
| +-----+ |     |                             | +-----+ |
| 42      | RTI | Revino din Intrerupere      |         |
| +-----+ |     |                             | +-----+ |
| 43      | RTS | Revino din Subrutina        |         |
| +-----+ |     |                             | +-----+ |
| 44      | SBC | Scazi din A cu fanionul C   |         |
| +-----+ |     |                             | +-----+ |
| 45      | SEC | Seteaza C                   |         |
| +-----+ |     |                             | +-----+ |
| 46      | SED | Seteaza Modul Zecimal-Binar |         |
| +-----+ |     |                             | +-----+ |
| 47      | SEI | Seteaza fanionul I          |         |
| +-----+ |     |                             | +-----+ |
| 48      | STA | Scrie A in Memorie          |         |
| +-----+ |     |                             | +-----+ |
| 49      | STX | Scrie X in Memorie          |         |
| +-----+ |     |                             | +-----+ |
| 50      | STY | Scrie Y in Memorie          |         |
| +-----+ |     |                             | +-----+ |
| 51      | TAX | Scrie A in X                |         |
| +-----+ |     |                             | +-----+ |
| 52      | TAY | Scrie A in Y                |         |
| +-----+ |     |                             | +-----+ |
| 53      | TSX | Scrie registrul SP in X     |         |
| +-----+ |     |                             | +-----+ |
| 54      | TXA | Scrie X in A                |         |
| +-----+ |     |                             | +-----+ |
| 55      | TXS | Scrie X in SP               |         |
| +-----+ |     |                             | +-----+ |
| 56      | TYA | Scrie Y in A                |         |
| +-----+ |     |                             | +-----+ |
| 57      | WDM | Specifica Easy6502          |         |
| +-----+ |     |                             | +-----+ |
| 58      | --- | Rezervat in Easy6502        |         |
| +-----+ |     |                             | +-----+ |

```

1 var codurileOperationale = [
2
3  /*****
4  /* LEGENDA tabloului: */
5  /* 57 de mnemonici (Nume) si 1 mnemonica de rezerva. */
6  /* [
7  /* Nume, Imediat, ZP, ZPX, ZPY, ABS, ABSX, ABSY,
8  /* IND, INDX, INDY, SNGL, BRA
9  /* ]
10 /*****

```

```

11
12 [ /* 01 */
13     "ADC", 0x69, 0x65, 0x75, null, 0x6d, 0x7d, 0x79,
14     null, 0x61, 0x71, null, null
15 ],
16 [ /* 02 */
17     "AND", 0x29, 0x25, 0x35, null, 0x2d, 0x3d, 0x39,
18     null, 0x21, 0x31, null, null
19 ],
20 [ /* 03 */
21     "ASL", null, 0x06, 0x16, null, 0x0e, 0x1e, null,
22     null, null, null, 0x0a, null
23 ],
24 [ /* 04 */
25     "BCC", null, null, null, null, null, null, null,
26     null, null, null, null, 0x90
27 ],
28 [ /* 05 */
29     "BCS", null, null, null, null, null, null, null,
30     null, null, null, null, 0xb0
31 ],
32 [ /* 06 */
33     "BEQ", null, null, null, null, null, null, null,
34     null, null, null, null, 0xf0
35 ],
36 [ /* 07 */
37     "BIT", null, 0x24, null, null, 0x2c, null, null,
38     null, null, null, null, null
39 ],
40 [ /* 08 */
41     "BMI", null, null, null, null, null, null, null,
42     null, null, null, null, 0x30
43 ],
44 [ /* 09 */
45     "BNE", null, null, null, null, null, null, null,
46     null, null, null, null, 0xd0
47 ],
48 [ /* 10 */
49     "BPL", null, null, null, null, null, null, null,
50     null, null, null, null, 0x10
51 ],
52 [ /* 11 */
53     "BRK", null, null, null, null, null, null, null,
54     null, null, null, null, 0x00, null
55 ],
56 [ /* 12 */
57     "BVC", null, null, null, null, null, null, null,
58     null, null, null, null, 0x50
59 ],
60 [ /* 13 */
61     "BVS", null, null, null, null, null, null, null,
62     null, null, null, null, 0x70
63 ],
64 [ /* 14 */

```

```

65     "CLC", null, null, null, null, null, null, null,
66         null, null, null, 0x18, null
67 ],
68 [ /* 15 */
69     "CLD", null, null, null, null, null, null, null,
70         null, null, null, 0xd8, null
71 ],
72 [ /* 16 */
73     "CLI", null, null, null, null, null, null, null,
74         null, null, null, 0x58, null
75 ],
76 [ /* 17 */
77     "CLV", null, null, null, null, null, null, null,
78         null, null, null, 0xb8, null
79 ],
80 [ /* 18 */
81     "CMP", 0xc9, 0xc5, 0xd5, null, 0xcd, 0xdd, 0xd9,
82         null, 0xc1, 0xd1, null, null
83 ],
84 [ /* 19 */
85     "CPX", 0xe0, 0xe4, null, null, 0xec, null, null,
86         null, null, null, null, null
87 ],
88 [ /* 20 */
89     "CPY", 0xc0, 0xc4, null, null, 0xcc, null, null,
90         null, null, null, null, null
91 ],
92 [ /* 21 */
93     "DEC", null, 0xc6, 0xd6, null, 0xce, 0xde, null,
94         null, null, null, null, null
95 ],
96 [ /* 22 */
97     "DEX", null, null, null, null, null, null, null,
98         null, null, null, 0xca, null
99 ],
100 [ /* 23 */
101     "DEY", null, null, null, null, null, null, null,
102         null, null, null, 0x88, null
103 ],
104 [ /* 24 */
105     "EOR", 0x49, 0x45, 0x55, null, 0x4d, 0x5d, 0x59,
106         null, 0x41, 0x51, null, null
107 ],
108 [ /* 25 */
109     "INC", null, 0xe6, 0xf6, null, 0xee, 0xfe, null,
110         null, null, null, null, null
111 ],
112 [ /* 26 */
113     "INX", null, null, null, null, null, null, null,
114         null, null, null, 0xe8, null
115 ],
116 [ /* 27 */
117     "INY", null, null, null, null, null, null, null,
118         null, null, null, 0xc8, null

```

```

119 ],
120 [ /* 28 */
121     "JMP", null, null, null, null, 0x4c, null, null,
122     0x6c, null, null, null, null
123 ],
124 [ /* 29 */
125     "JSR", null, null, null, null, 0x20, null, null,
126     null, null, null, null, null
127 ],
128 [ /* 30 */
129     "LDA", 0xa9, 0xa5, 0xb5, null, 0xad, 0xbd, 0xb9,
130     null, 0xa1, 0xb1, null, null
131 ],
132 [ /* 31 */
133     "LDX", 0xa2, 0xa6, null, 0xb6, 0xae, null, 0xbe,
134     null, null, null, null, null
135 ],
136 [ /* 32 */
137     "LDY", 0xa0, 0xa4, 0xb4, null, 0xac, 0xbc, null,
138     null, null, null, null, null
139 ],
140 [ /* 33 */
141     "LSR", null, 0x46, 0x56, null, 0x4e, 0x5e, null,
142     null, null, null, 0x4a, null
143 ],
144 [ /* 34 */
145     "NOP", null, null, null, null, null, null, null,
146     null, null, null, 0xea, null
147 ],
148 [ /* 35 */
149     "ORA", 0x09, 0x05, 0x15, null, 0x0d, 0x1d, 0x19,
150     null, 0x01, 0x11, null, null
151 ],
152 [ /* 36 */
153     "PHA", null, null, null, null, null, null, null,
154     null, null, null, 0x48, null
155 ],
156 [ /* 37 */
157     "PHP", null, null, null, null, null, null, null,
158     null, null, null, 0x08, null
159 ],
160 [ /* 38 */
161     "PLA", null, null, null, null, null, null, null,
162     null, null, null, 0x68, null
163 ],
164 [ /* 39 */
165     "PLP", null, null, null, null, null, null, null,
166     null, null, null, 0x28, null
167 ],
168 [ /* 40 */
169     "ROL", null, 0x26, 0x36, null, 0x2e, 0x3e, null,
170     null, null, null, 0x2a, null
171 ],
172 [ /* 41 */

```

```

173     "ROR", null, 0x66, 0x76, null, 0x6e, 0x7e, null,
174     null, null, null, 0x6a, null
175 ],
176 [ /* 42 */
177     "RTI", null, null, null, null, null, null, null,
178     null, null, null, 0x40, null
179 ],
180 [ /* 43 */
181     "RTS", null, null, null, null, null, null, null,
182     null, null, null, 0x60, null
183 ],
184 [ /* 44 */
185     "SBC", 0xe9, 0xe5, 0xf5, null, 0xed, 0xfd, 0xf9,
186     null, 0xe1, 0xf1, null, null
187 ],
188 [ /* 45 */
189     "SEC", null, null, null, null, null, null, null,
190     null, null, null, 0x38, null
191 ],
192 [ /* 46 */
193     "SED", null, null, null, null, null, null, null,
194     null, null, null, 0xf8, null
195 ],
196 [ /* 47 */
197     "SEI", null, null, null, null, null, null, null,
198     null, null, null, 0x78, null
199 ],
200 [ /* 48 */
201     "STA", null, 0x85, 0x95, null, 0x8d, 0x9d, 0x99,
202     null, 0x81, 0x91, null, null
203 ],
204 [ /* 49 */
205     "STX", null, 0x86, null, 0x96, 0x8e, null, null,
206     null, null, null, null, null
207 ],
208 [ /* 50 */
209     "STY", null, 0x84, 0x94, null, 0x8c, null, null,
210     null, null, null, null, null
211 ],
212 [ /* 51 */
213     "TAX", null, null, null, null, null, null, null,
214     null, null, null, 0xaa, null
215 ],
216 [ /* 52 */
217     "TAY", null, null, null, null, null, null, null,
218     null, null, null, 0xa8, null
219 ],
220 [ /* 53 */
221     "TSX", null, null, null, null, null, null, null,
222     null, null, null, 0xba, null
223 ],
224 [ /* 54 */
225     "TXA", null, null, null, null, null, null, null,
226     null, null, null, 0x8a, null

```

```

227 ],
228 [ /* 55 */
229     "TXS", null, null, null, null, null, null, null,
230     null, null, null, 0x9a, null
231 ],
232 [ /* 56 */
233     "TYA", null, null, null, null, null, null, null,
234     null, null, null, 0x98, null
235 ],
236 [ /* 57 */
237     "WDM", 0x42, 0x42, null, null, null, null, null,
238     null, null, null, null, null
239 ],
240 [ /* 58 */
241     "---", null, null, null, null, null, null, null,
242     null, null, null, null, null
243 ]
244 ];

```

## 2.10 Funcția Simulatorul()

La această versiune a aplicației Easy6502, *imitarea* funcționării microprocesorului 6502 trebuie socotită drept *aproximativă*, adică de folosit în scop didactic. Există *diferențe semnificative* între rezultatele execuțiilor de programe demonstrative<sup>54</sup> pe Easy6502 și cele realizate pe emulatorul original al dlui. Stian Søreng.

### 2.10.1 Funcțiile setare...()

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile **101–107** ale listării de cod din Secțiunea 2.1. Cele 7 fanioane ale arhitecturii 6502 sunt localizate în registrul P, de stare a procesorului [70, pp. 19, 20]: 5 biți *de stare* — B, C, N, V, Z — și 2 biți *de control* — D, I —. Vezi blocul **14** din Figura 2.1.

| +-----+<br>  Registrul P: NV-BDIZC  <br>+-----+ |          |                      |  |
|-------------------------------------------------|----------|----------------------|--|
| Bitul                                           | Fanionul | Semnificatia lui 1   |  |
| 7                                               | N        | Negativ              |  |
| 6                                               | V        | Depasire             |  |
| 5                                               | -        | Rezervat             |  |
| 4                                               | B        | Oprire fortata (BRK) |  |
| 3                                               | D        | Modul zecimal-binar  |  |

<sup>54</sup> Din setul depozitat pe platforma Wayback Machine.

|   |   |                |  |
|---|---|----------------|--|
| 2 | I | IRQ dezactivat |  |
| 1 | Z | Rezultat Zero  |  |
| 0 | C | Cu transport   |  |

```

1 function setareFanioaneleNZV( valoarea ) {
2   if ( valoarea ) {
3     registrulP &= 0xfd;
4   } else {
5     registrulP |= 0x02;
6   }
7   if ( valoarea & 0x80 ) {
8     registrulP |= 0x80;
9   } else {
10    registrulP &= 0x7f;
11  }
12 }
13
14 function setareFanionulCdeLaBitul0( valoarea ) {
15   registrulP = (registrulP & 0xfe)
16     | (valoarea & 0x01);
17 }
18
19 function setareFanionulCdeLaBitul7( valoarea ) {
20   registrulP = (registrulP & 0xfe)
21     | ((valoarea >> 7) & 0x01);
22 }
23
24 function setareFanioaneleNZVpentruRegistrulA() {
25   setareFanioaneleNZV( registrulA );
26 }
27
28 function setareFanioaneleNZVpentruRegistrulX() {
29   setareFanioaneleNZV( registrulX );
30 }
31
32 function setareFanioaneleNZVpentruRegistrulY() {
33   setareFanioaneleNZV( registrulY );
34 }
35
36 function setareFanionulV() {
37   registrulP |= 0x40;
38 }

```

Măștile de biți de la rândurile **3** — 11111101 — și **5** — 00000010 — ne permit setarea fanionului Z iar măștile de la rândurile **8** — 10000000 — și **10** — 01111111 — setarea fanionului N. Fanionul V este setat cu masca de la rândul **37** — 01000000 —.

Masca de biți de la rândurile **15** și **20** — 11111110 — ne asigură că setarea fanionului C este decisă de bitul 0 din mărimea valoarea, la rândul **16**, respectiv de bitul 7 la rândul **21**.

Într-o versiune viitoare a aplicației Easy6502, funcțiile setareFanioanele-NZV() și setareFanionulV() vor fi comasate.

### 2.10.2 Funcțiile BIT(), CLC(), SEC(), CLV(), DEC(), INC() și saltRamificație()

Program:> În codul care urmează, implementăm funcțiile care se găsesc la rândurile **108–114** ale listării de cod din Secțiunea 2.1.

```

1  function BIT( valoarea ) {
2    if ( valoarea & 0x80 ) {
3      registrulP |= 0x80;
4    } else {
5      registrulP &= 0x7f;
6    }
7    if ( valoarea & 0x40 ) {
8      registrulP |= 0x40;
9    } else {
10     registrulP &= ( ~0x40 );
11   }
12   if ( registrulA & valoarea ) {
13     registrulP &= 0xfd;
14   } else {
15     registrulP |= 0x02;
16   }
17 }
18
19 function CLC() {
20   registrulP &= 0xfe;
21 }
22
23 function SEC() {
24   registrulP |= 0x01;
25 }
26
27 function CLV() {
28   registrulP &= 0xbf;
29 }
30
31 function DEC( adresa ) {
32   var valoarea = memorie.memoriaCiteste( adresa );
33
34   valoarea--;
35   valoarea &= 0xff;
36   memorie.memoriaStocheazaOctetul( adresa, valoarea );

```

```

37  setareFanioaneleNZV( valoarea );
38  }
39
40  function INC( adresa ) {
41      var valoarea = memorie.memoriaCiteste( adresa );
42
43      valoarea++;
44      valoarea &= 0xff;
45      memorie.memoriaStocheazaOctetul( adresa, valoarea );
46      setareFanioaneleNZV( valoarea );
47  }
48
49  function saltRamificatie( deplasamentul ) {
50      if ( deplasamentul > 0x7f ) {
51          registrulPC -= (0x100 - deplasamentul);
52      } else {
53          registrulPC += deplasamentul;
54      }
55  }

```

Funcția `BIT()` permite testarea *biților individuali* ai unei celule de memorie cu ajutorul fanioanelor N, V, Z — rândurile **3, 8, 15** —, ceea ce face posibilă utilizarea microprocesorului 6502 la *comanda* echipamentelor electrice [70, pag. 62]. Vezi exemplul dat în continuare. Aici, *masca* de biți *masca* are activat bitul de pe poziția de testat. Testul este *trecut* dacă bitul de la aceeași poziție al octetului stocat în celula de memorie este 0 [70, *ibid.*].

```

define masca          $04 ; bitul 2
define valoareaTestata $13 ; bitul 2 este 0?
define adresaTestata  $02
bucla: ldx             #valoareaTestata
      stx             adresaTestata
      lda             #masca
      bit             adresaTestata
      bne             bucla
      ldy             #valoareaTestata

```

### 2.10.3 Funcțiile *fanionul...()*, *modulZecimal()* și *faComparatia()*

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile **115–120** ale listării de cod din Secțiunea 2.1.

```

1  function fanionulVsetat() {
2      return (registrulP & 0x40);
3  }
4
5  function fanionulCsetat() {

```

```

6   return (registrulP & 0x01);
7 }
8
9 function fanionulNsetat() {
10  return (registrulP & 0x80);
11 }
12
13 function fanionulZsetat() {
14  return (registrulP & 0x02);
15 }
16
17 function modulZecimal() {
18  return (registrulP & 0x08);
19 }
20
21 function faComparatia( registrul, valoarea ) {
22  if ( registrul >= valoarea ) {
23      SEC();
24  } else {
25      CLC();
26  }
27  valoarea = registrul - valoarea;
28  setareFanioaneleNZV( valoarea );
29 }

```

La rândul **18**, funcția `modulZecimal()` setează fanionul D.

#### 2.10.4 Funcțiile testeaza...()

Program:> În codul care urmează, implementăm funcțiile care se găsesc la rândurile **121** și **122** ale listării de cod din Secțiunea 2.1.

```

1 function testeazaSBC( valoarea ) {
2   var temporar;
3   var cuvantul;
4
5   if ( (registrulA ^ valoarea) & 0x80 ) {
6       setareFanionulV();
7   } else {
8       CLV();
9   }
10  if ( modulZecimal() ) {
11      temporar = 0xf + (registrulA & 0xf)
12      - (valoarea & 0xf) + fanionulCsetat();
13      if ( temporar < 0x10 ) {
14          cuvantul = 0;
15          temporar -= 0x06;
16      } else {
17          cuvantul = 0x10;
18          temporar -= 0x10;

```

```

19     }
20     cuvantul += 0xf0 + (registrulA & 0xf0)
21     - (valoarea & 0xf0);
22     if (cuvantul < 0x100) {
23         CLC();
24         if ( (fanionulVsetat()) && (cuvantul < 0x80) ) {
25             CLV();
26         }
27         cuvantul -= 0x60;
28     } else {
29         SEC();
30         if ( (fanionulVsetat()) && (cuvantul >= 0x180) ) {
31             CLV();
32         }
33     }
34     cuvantul += temporar;
35 } else {
36     cuvantul = 0xff + registrulA - valoarea
37     + fanionulCsetat();
38     if (cuvantul < 0x100) {
39         CLC();
40         if ( (fanionulVsetat()) && (cuvantul < 0x80) ) {
41             CLV();
42         }
43     } else {
44         SEC();
45         if ( (fanionulVsetat()) && (cuvantul >= 0x180) ) {
46             CLV();
47         }
48     }
49 }
50 registrulA = cuvantul & 0xff;
51 setareFanioaneleNZVpentruRegistrulA();
52 }
53
54 function testeazaADC( valoarea ) {
55     var temporar;
56
57     if ( (registrulA ^ valoarea) & 0x80 ) {
58         CLV();
59     } else {
60         setareFanionulV();
61     }
62     if ( modulZecimal() ) {
63         temporar = (registrulA & 0xf)
64         + (valoarea & 0xf) + fanionulCsetat();
65         if ( temporar >= 0x0a ) {
66             temporar = 0x10
67             | ((temporar + 0x06) & 0xf);
68         }
69         temporar += (registrulA & 0xf0)
70         + (valoarea & 0xf0);
71         if ( temporar >= 0xa0 ) {
72             SEC();

```

```

73     if ( (fanionulVsetat()) && (temporar >= 0x180) ) {
74         CLV();
75     }
76     temporar += 0x60;
77 } else {
78     CLC();
79     if ( (fanionulVsetat()) && (temporar < 0x80) ) {
80         CLV();
81     }
82 }
83 } else {
84     temporar = registrula + valoarea + fanionulCsetat();
85     if ( temporar >= 0x100 ) {
86         SEC();
87         if ( (fanionulVsetat()) && (temporar >= 0x180) ) {
88             CLV();
89         }
90     } else {
91         CLC();
92         if ( (fanionulVsetat()) && (temporar < 0x80) ) {
93             CLV();
94         }
95     }
96 }
97 registrula = temporar & 0xff;
98 setareFanioaneleNZVpentruRegistrula();
99 }

```

În arhitectura 6502, *adunarea* ADC a numerelor întregi, respectiv *scăderea* SBC a lor sunt implementate *cu fanionul C* [70, pp. 44, 46]. Operațiile pot fi efectuate atât în modul zecimal cât și în cel zecimal-binar — rândurile **10, 62** — [70, pag. 43]. Vezi exemplul de mai jos.

```

lda #12
clc
adc #3 ; A = A + M + C, M = memorie
sec
sbc #3 ; A = A - M - (not C)

```

La rândul **5** este folosit operatorul logic *SAU exclusiv* — XOR — [69]. Adică, un operator *not-egal* [44, pag. 9-4]. Codul de la rândurile **5–9** și cel de la rândurile **57–61** se referă la *invalidarea* rezultatului — vezi rândurile **50, 97** —, conform [70, pag. 48]: pentru SBC atunci când avem operanzi de semne *diferite* iar pentru ADC operanzi de *același* semn.

### 2.10.5 Funcțiile *urca...()*, *coboara...()*

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile **123–126** ale listării de cod din Secțiunea 2.1.

```

1 function urcaPeStiva ( valoarea ) {
2     memorie.memoriaSeteaza(
3         ((registrulSP & 0xff) + 0x100), (valoarea & 0xff)
4     );
5     registrulSP--;
6     if ( registrulSP < 0 ) {
7         registrulSP &= 0xff;
8         auxiliarAfiseazaMesajul(
9             "Stiva microprocesorului 6502 este plin&#259;!"
10            + "<br>O reset&#259;m..."
11        );
12    }
13 }
14
15 function coboaraDinStiva(){
16     registrulSP++;
17     if ( registrulSP >= 0x100 ) {
18         registrulSP &= 0xff;
19         auxiliarAfiseazaMesajul(
20             "Stiva microprocesorului 6502 este goal&#259;!"
21            + "<br>O reset&#259;m..."
22        );
23    }
24     return (
25         memorie.memoriaCiteste( registrulSP + 0x100 )
26     );
27 }
28
29 function coboaraUnOctetDinMemorie() {
30     return (
31         (memorie.memoriaCiteste( registrulPC++ )) & 0xff
32     );
33 }
34
35 function coboaraUnCuvantDinMemorie() {
36     return (
37         coboaraUnOctetDinMemorie()
38         + (coboaraUnOctetDinMemorie() << 8)
39     );
40 }

```

*Stiva*<sup>55</sup> este un spațiu *contiguu* de adrese de memorie, care vor fi ocupate în *ordine inversă* — de sus în jos, vezi rândul 5 — și care acoperă plaja de valori 0x0100–0x01ff, formând *pagina unu* [70, pp. 21, 70]. Vârful stivei *goale* are adresa 0x01ff. Pe măsură ce stiva se umple, vârful său coboară către adresa 0x0100. *Forma* acestor adrese, 0x01hh — unde h desemnează o cifră hexazecimală —, face posibilă stocarea vârfului stivei în registrul SP, cu formatul “\$hh” — conform rândului 3 —.

---

<sup>55</sup> Vezi pagina 32.

Extragerea octeților din stivă se realizează urmând regula *ultimul venit este primul plecat*<sup>56</sup> — rândul **16** —.

În exemplul listat în continuare este implementată o *subrutină*<sup>57</sup>, adică un bloc de cod care poate fi *apelat*<sup>58</sup> — de câte ori dorim — din programul aflat în execuție. La apelul subrutinei, valoarea registrului PC va fi *urcată*<sup>59</sup> pe stivă. Astfel, la *revenirea*<sup>60</sup> din subrutină, emulatorul *va ști* ce are de executat. Subrutina din exemplu afișează mesajul “Vă salut !” în fereastra *de mesagerie* a aplicației Easy6502 — elementul HTML **26** din Figura 1.2 —. Codul trebuie executat *de două ori* pentru a vizualiza mesajul — vezi *valoarea* registrului PC din *minidepanator*, elementul **14** din Figura 1.2 —.

```
; a9 56 42 00 a9 e3 42 00 a9 20 42 00 a9 73 42 00
lda #$a9
sta $00
lda #$56
sta $01
lda #$42
sta $02
lda #$00
sta $03
lda #$a9
sta $04
lda #$e3
sta $05
lda #$42
sta $06
lda #$00
sta $07
lda #$a9
sta $08
lda #$20
sta $09
lda #$42
sta $0a
lda #$00
sta $0b
lda #$a9
sta $0c
lda #$73
sta $0d
lda #$42
sta $0e
lda #$00
sta $0f
; a9 61 42 00 a9 6c 42 00 a9 75 42 00 a9 74 42 00
lda #$a9
sta $10
lda #$61
sta $11
lda #$42
sta $12
lda #$00
sta $13
lda #$a9
```

<sup>56</sup> În limba engleză, *last in first out* — LIFO — [70, pag. ibid.].

<sup>57</sup> În limba engleză, *subroutine* [70, pag. 78].

<sup>58</sup> În limba engleză, *called* [70, pag. 75].

<sup>59</sup> În limba engleză, *pushed*.

<sup>60</sup> În limba engleză, *returning*.

```

sta $14
lda #$6c
sta $15
lda #$42
sta $16
lda #$00
sta $17
lda #$a9
sta $18
lda #$75
sta $19
lda #$42
sta $1a
lda #$00
sta $1b
lda #$a9
sta $1c
lda #$74
sta $1d
lda #$42
sta $1e
lda #$00
sta $1f
; a9 21 42 00
lda #$a9
sta $20
lda #$21
sta $21
lda #$42
sta $22
lda #$00
sta $23
lda #$60 ; RTS
sta $24

jsr $0000
ldx #$03

```

### 2.10.6 Funcțiile *executa..()*, *multiExecuta()*

Program:> În codul care urmează, implementăm funcțiile care se găsesc la rândurile **127–130** ale listării de cod din Secțiunea 2.1.

```

1 function executaUrmatoareaInstructiune() {
2   var numeleInstructiunii = coboaraUnOctetDinMemorie()
3   .toString(16)
4   .toLowerCase();
5   var instructiunea;
6
7   if ( numeleInstructiunii.length === 1 ) {
8     numeleInstructiunii = "0" + numeleInstructiunii;
9   }
10  instructiunea = instructiunile[ "i" + numeleInstructiunii ];
11  if ( instructiunea ) {
12    instructiunea();
13  } else {
14    instructiunile.ieroare();

```

```

15     }
16 }
17
18 function executa( codInDepanare ) {
19     if ( (!codInExecutie) && (!codInDepanare) ) {
20         return;
21     }
22
23     auxiliarSeteazaAleatoriuOctetul();
24     executaUrmatoareaInstructiune();
25     if (
26         (registrulPC === 0)
27         || ((!codInExecutie) && (!codInDepanare))
28     ) {
29         opresteSimulatorul();
30         auxiliarAfiseazaMesajul(
31             "Programul s-a oprit la PC=$"
32             + auxiliarConvertescAdresaLaHexa( registrulPC - 1 )
33             + "."
34         );
35         interfata.interfataOpreste();
36     }
37 }
38
39 function executaBinarul() {
40     if ( codInExecutie ) {
41         opresteSimulatorul();
42         interfata.interfataOpreste();
43     } else {
44         interfata.interfataPorneste();
45         codInExecutie = true;
46         identificatorulInstructiuniiDeExecutat
47             = setInterval( multiExecuta, 15 );
48     }
49 }
50
51 function multiExecuta() {
52     var iteratorul1;
53
54     if ( !depanareSimulator ) {
55         for ( iteratorul1 = 0; iteratorul1 < 97; iteratorul1++ ) {
56             executa();
57         }
58     }
59     actualizeazaInformatiaDepanatorului();
60 }

```

La execuția *oricărei* instrucțiuni este calculat — vezi rândul **23** — câte un octet pseudo-aleatoriu<sup>61</sup>.

Execuția programului se oprește (și) atunci când registrul PC este 0x0000, caz în care în fereastra de mesagerie se afișează valoarea \$ffff, conform rândului **32**.

---

<sup>61</sup> Vezi pagina 18.

*Viteza de execuție* a emulatorului poate fi modificată dacă schimbăm numărul magic 15 — de la rândul 47 — care definește *pauza* [53], măsurată în *milisecunde*, dintre apelurile funcției `multiExecuta()`.

### 2.10.7 Funcțiile `actualizeazaMonitorul()` și `manevreLaSchimbareaPlajeiMonitorului()`

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile 131 și 132 ale listării de cod din Secțiunea 2.1.

```

1  function actualizeazaMonitorul() {
2      var deUndeIncep;
3      var lungimea;
4      var undeInchei;
5      var nodulMonitorului;
6
7      if ( monitorizareSimulator ) {
8          deUndeIncep
9              = parseInt( $nod.find( ".inceputul" ).val(), 16 );
10         lungimea
11             = parseInt( $nod.find( ".lungimea" ).val(), 16 );
12         undeInchei = deUndeIncep + lungimea - 1;
13         nodulMonitorului = $nod.find( ".monitorul code" );
14
15         if (
16             (!isNaN( deUndeIncep )) && (!isNaN( lungimea ))
17             && (deUndeIncep >= 0) && (lungimea > 0)
18             && (undeInchei <= 0xffff)
19         ) {
20             nodulMonitorului.html(
21                 memorie.memoriaFormateaza( deUndeIncep, lungimea )
22             );
23         } else {
24             nodulMonitorului.html(
25                 "Nu putem monitoriza aceste&#259; plaj&#259; de adrese."
26                 + "<br>Adresele permise sunt cele cuprinse &#238;ntre "
27                 + "<br>&#36;0000 &#351;i &#36;ffff, inclusiv."
28             );
29         }
30     }
31 }
32
33 function manevreLaSchimbareaPlajeiMonitorului() {
34     var $inceput = $nod.find( ".inceputul" );
35     var $lungime = $nod.find( '.lungimea' );
36     var startul = parseInt( $inceput.val(), 16 );
37     var lungimea = parseInt( $lungime.val(), 16 );
38     var finalul = startul + lungimea - 1;
39

```

```

40 $inceput.removeClass( "monitorulInvalidat" );
41 $lungime.removeClass( "monitorulInvalidat" );
42 if (
43     (isNaN( startul )) || (startul < 0) || (startul > 0xffff)
44 ) {
45     $inceput.addClass( "monitorulInvalidat" );
46 } else if ( (isNaN( lungimea )) || (finalul > 0xffff) ) {
47     $lungime.addClass( "monitorulInvalidat" );
48 }
49 }

```

Fereastra de mesagerie a monitorului execuției este elementul HTML **25** din Figura 1.2. Vezi rândurile **13** și **24**.

În elementele HTML **22** și **24** trebuie introduse *numere întregi fără semn*, din plaja de valori 0x0000–0xffff. În caz contrar, elementele vor fi *dezactivate* — rândurile **45**, **47** —.

### 2.10.8 Funcțiile *depaneazaExecutia()*, *actualizeazaInformatiaDepanatorului()* și *adresaUndeMergi()*

Program:> În codul care urmează, implementăm funcțiile care se găsesc la rândurile **133–135** ale listării de cod din Secțiunea 2.1.

```

1 function depaneazaExecutia() {
2     executa( true );
3     actualizeazaInformatiaDepanatorului();
4 }
5
6 function actualizeazaInformatiaDepanatorului() {
7     var textulHTML = "Regi&#351;trii A=$"
8     + auxiliarConvertescNumarLaHexa( registrulA )
9     + ",X=$"
10    + auxiliarConvertescNumarLaHexa( registrulX )
11    + ",Y=$"
12    + auxiliarConvertescNumarLaHexa( registrulY )
13    + "<br>SP=$"
14    + auxiliarConvertescNumarLaHexa( registrulSP )
15    + ",PC=$"
16    + auxiliarConvertescAdresaLaHexa( registrulPC )
17    + "<br>";
18    var fanioanele = "NV-BDIZC";
19    var iteratorul;
20
21    textulHTML
22    += "Registrul P, fanioanele NV-BDIZC:<br>";
23    for ( iteratorul = 0; iteratorul <= 7; iteratorul++ ) {
24        textulHTML += fanioanele

```

```

25     .substring( iteratorul, iteratorul + 1 )
26     + "=";
27     textulHTML
28     += ((registrulP >> (7 - iteratorul)) & 0x01)
29     + ((iteratorul === 7) ? "." : ",");
30 }
31 $nod.find( ".minidepanatorul" ).html( textulHTML );
32 actualizeazaMonitorul();
33 }
34
35 function adresaUndeMergi() {
36     var textulInserat = prompt(
37         "Introduce\u021bi fie o adres\u0103"
38         + " fie o etichet\u0103.",
39         ""
40     );
41     var adresa = 0;
42
43     if (
44         etichete.eticheteleGaseste( textulInserat )
45     ) {
46         adresa = etichete
47             .eticheteleCitestePC( textulInserat );
48     } else {
49         if (
50             textulInserat.match( /^0x[0-9a-f]{1,4}$/i )
51         ) {
52             textulInserat = textulInserat
53                 .replace( /^0x/, "" );
54             adresa = parseInt( textulInserat, 16 );
55         } else if (
56             textulInserat.match( /^\[0-9a-f]{1,4}$/i )
57         ) {
58             textulInserat = textulInserat
59                 .replace( /\[/, "" );
60             adresa = parseInt( textulInserat, 16 );
61         }
62     }
63     if ( adresa === 0 ) {
64         auxiliarAfiseazaMesajul(
65             "Nu pot g\u0103si/parsa adresa/eticheta"
66             + " dat\u0103;."
67         );
68     } else {
69         registrulPC = adresa;
70     }
71     actualizeazaInformatiaDepanatorului();
72 }

```

Funcția `depaneazaExecutia()` este *funcția de reacție* a butonului **16** din Figura 1.2 iar funcția `adresaUndeMergi()` cea a butonului **17**. Vezi Secțiunea 2.3.

**2.10.9 Funcțiile ...Depanatorul(),...Simulatorul()**

Program:> În codul următor, implementăm funcțiile care se găsesc la rândurile **136–140** ale listării de cod din Secțiunea 2.1.

```

1  function opresteDepanatorul() {
2    depanareSimulator = false;
3  }
4
5  function activeazaDepanatorul() {
6    depanareSimulator = true;
7    if ( codInExecutie ) {
8      actualizeazaInformatiaDepanatorului();
9    }
10 }
11
12 function reseteazaSimulatorul() {
13   var iteratorul;
14
15   afisor.ecranulResetare();
16   for ( iteratorul = 0; iteratorul < 0x600; iteratorul++ ) {
17     memorie.memoriaSeteaza( iteratorul, 0x00 );
18   }
19   registrulA = registrulX = registrulY = 0x00;
20   registrulPC = 0x600;
21   registrulSP = 0xff;
22   registrulP = 0x30;
23   actualizeazaInformatiaDepanatorului();
24 }
25
26 function opresteSimulatorul( e ) {
27   if ( (!e) || (e && (e.code !== "Tab")) ) {
28     codInExecutie = false;
29     clearInterval( identificatorulInstructiuniiDeExecutat );
30     auxiliarAfiseazaMesajul( "\nAm oprit emulatorul.\n" );
31   }
32 }
33
34 function monitorizareSchimbariSimulatorul( starea ) {
35   monitorizareSimulator = starea;
36 }

```

Conform rândului **27**, execuția programului în emulator *nu se oprește* dacă inserăm Tab-uri în fereastra editorului aplicației Easy6502.

### 2.10.10 Instrucțiunile executabile

Program:> În codul care urmează, *inițializăm* variabila instrucțiunile de la rândul 99 al listării de cod din Secțiunea 2.1. *Opcodurile* instrucțiunilor se găsesc în tablourile componente din Subsecțiunea 2.9.9.

```

1  var instructiunile = {
2
3  /* ***** */
4  /*  LEGENDA tablei: 152 de instructiuni + 1 instructiune  */
5  /*                de eroare                                */
6  /*  --citesc instructiunea:  i[nr. liniei][nr. coloanei]  */
7  /*  --exista instructiunea:  +                            */
8  /*  --nu exista instructiunea: *                          */
9  /*  ----- */
10 /*  |i|0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f| */
11 /*  |-----| */
12 /*  |0|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
13 /*  |1|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
14 /*  |2|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
15 /*  |3|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
16 /*  |4|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
17 /*  |5|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
18 /*  |6|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
19 /*  |7|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
20 /*  |8|*|+|*|*|+|+|*|+|+|*|*|+|+|*| */
21 /*  |9|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
22 /*  |a|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
23 /*  |b|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
24 /*  |c|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
25 /*  |d|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
26 /*  |e|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
27 /*  |f|+|+|*|*|*|+|+|*|+|+|*|*|+|+|*| */
28 /*  ----- */
29 /* ***** */
30
31 i00 : function () {                                /* BRK */
32     codInExecutie = false;
33 },
34 i01 : function () {
35     var paginaZero
36         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
37     var adresa
38         = memorie.memoriaCitesteCuvantul( paginaZero );
39     var valoarea = memorie.memoriaCiteste( adresa );
40     registrulA |= valoarea;
41     ORA();
42 },
43 i05 : function () {
44     var paginaZero = coboaraUnOctetDinMemorie();
45     var valoarea = memorie.memoriaCiteste( paginaZero );
46     registrulA |= valoarea;

```

```

47         ORA();
48     },
49     i06 : function() {
50         var paginaZero = coboaraUnOctetDinMemorie();
51         var valoarea = memorie.memoriaCiteste( paginaZero );
52         setareFanionulCdeLaBitul7( valoarea );
53         valoarea = (valoarea << 1) & 0xff;
54         memorie
55             .memoriaStocheazaOctetul( paginaZero, valoarea );
56         ASL( valoarea );
57     },
58     i08 : function() {                                     /* PHP */
59         urcaPeStiva( registrulP | 0x30 );
60     },
61     i09 : function() {
62         registrulA |= coboaraUnOctetDinMemorie();
63         ORA();
64     },
65     i0a : function() {
66         setareFanionulCdeLaBitul7( registrulA );
67         registrulA = (registrulA << 1) & 0xff;
68         ASL( registrulA );
69     },
70     i0d : function() {
71         var adresa = coboaraUnCuvantDinMemorie();
72         var valoarea = memorie.memoriaCiteste( adresa );
73         registrulA |= valoarea;
74         ORA();
75     },
76     i0e : function() {
77         var adresa = coboaraUnCuvantDinMemorie();
78         var valoarea = memorie.memoriaCiteste( adresa );
79         setareFanionulCdeLaBitul7( valoarea );
80         valoarea = (valoarea << 1) & 0xff;
81         memorie
82             .memoriaStocheazaOctetul( adresa, valoarea );
83         ASL( valoarea );
84     },
85     i10 : function() {                                     /* BPL */
86         var deplasamentul = coboaraUnOctetDinMemorie();
87         var fanionSetat = fanionulNsetat();
88         if ( !fanionSetat ) {
89             saltRamificatie( deplasamentul );
90         }
91     },
92     i11 : function() {
93         var paginaZero = coboaraUnOctetDinMemorie();
94         var valoarea = memorie
95             .memoriaCitesteCuvantul( paginaZero )
96             + registrulY;
97         registrulA |= memorie.memoriaCiteste( valoarea );
98         ORA();
99     },
100    i15 : function() {

```

```

101     var adresa
102         = (coboaraUnOctetDinMemorie() + registrulX)
103         & 0xff;
104     registrulA |= memorie.memoriaCiteste( adresa );
105     ORA();
106 },
107 i16 : function() {
108     var adresa
109         = (coboaraUnOctetDinMemorie() + registrulX)
110         & 0xff;
111     var valoarea = memorie.memoriaCiteste( adresa );
112     setareFanionulCdeLaBitul7( valoarea );
113     valoarea = (valoarea << 1) & 0xff;
114     memorie
115         .memoriaStocheazaOctetul( adresa, valoarea );
116     ASL(valoarea);
117 },
118 i18 : function() {
119     CLC();
120 },
121 i19 : function() {
122     var adresa
123         = coboaraUnCuvantDinMemorie() + registrulY;
124     registrulA |= memorie.memoriaCiteste( adresa );
125     ORA();
126 },
127 i1d : function() {
128     var adresa
129         = coboaraUnCuvantDinMemorie() + registrulX;
130     registrulA |= memorie.memoriaCiteste( adresa );
131     ORA();
132 },
133 ile : function() {
134     var adresa
135         = coboaraUnCuvantDinMemorie() + registrulX;
136     var valoarea = memorie.memoriaCiteste( adresa );
137     setareFanionulCdeLaBitul7( valoarea );
138     valoarea = (valoarea << 1) & 0xff;
139     memorie
140         .memoriaStocheazaOctetul( adresa, valoarea );
141     ASL( valoarea );
142 },
143 i20 : function() { /* JSR */
144     var adresa = coboaraUnCuvantDinMemorie();
145     var adresaCurenta = registrulPC - 1;
146     urcaPeStiva( (adresaCurenta >> 8) & 0xff );
147     urcaPeStiva( adresaCurenta & 0xff );
148     registrulPC = adresa;
149 },
150 i21 : function() {
151     var paginaZero
152         = (coboaraUnOctetDinMemorie() + registrulX)
153         & 0xff;
154     var adresa

```

```

155         = memorie.memoriaCitesteCuvantul( paginaZero );
156     var valoarea = memorie.memoriaCiteste( adresa );
157     registrulA &= valoarea;
158     AND();
159 },
160 i24 : function() {
161     var paginaZero = coboaraUnOctetDinMemorie();
162     var valoarea = memorie.memoriaCiteste( paginaZero );
163     BIT( valoarea );
164 },
165 i25 : function() {
166     var paginaZero = coboaraUnOctetDinMemorie();
167     var valoarea = memorie.memoriaCiteste( paginaZero );
168     registrulA &= valoarea;
169     AND();
170 },
171 i26 : function() {
172     var fanionSetat = fanionulCsetat();
173     var adresa = coboaraUnOctetDinMemorie();
174     var valoarea = memorie.memoriaCiteste( adresa );
175     setareFanionulCdeLaBitul7( valoarea );
176     valoarea = (valoarea << 1) & 0xff;
177     valoarea |= fanionSetat;
178     memorie.memoriaStocheazaOctetul( adresa, valoarea );
179     ROL( valoarea );
180 },
181 i28 : function() { /* PLP */
182     var valoarea = coboaraDinStiva();
183     registrulP = valoarea | 0x30;
184 },
185 i29 : function() {
186     var valoarea = coboaraUnOctetDinMemorie();
187     registrulA &= valoarea;
188     AND();
189 },
190 i2a : function() {
191     var fanionSetat = fanionulCsetat();
192     setareFanionulCdeLaBitul7( registrulA );
193     registrulA = (registrulA << 1) & 0xff;
194     registrulA |= fanionSetat;
195     ROL( registrulA );
196 },
197 i2c : function() {
198     var adresa = coboaraUnCuvantDinMemorie();
199     var valoarea = memorie.memoriaCiteste( adresa );
200     BIT( valoarea );
201 },
202 i2d : function() {
203     var adresa = coboaraUnCuvantDinMemorie();
204     var valoarea = memorie.memoriaCiteste( adresa );
205     registrulA &= valoarea;
206     AND();
207 },
208 i2e : function() {

```

```

209     var fanionSetat = fanionulCsetat();
210     var adresa      = coboaraUnCuvantDinMemorie();
211     var valoarea    = memorie.memoriaCiteste( adresa );
212     setareFanionulCdeLaBitul7( valoarea );
213     valoarea = (valoarea << 1) & 0xff;
214     valoarea |= fanionSetat;
215     memorie.memoriaStocheazaOctetul( adresa, valoarea );
216     ROL( valoarea );
217 },
218 i30 : function() { /* BMI */
219     var deplasamentul = coboaraUnOctetDinMemorie();
220     var fanionSetat    = fanionulNsetat();
221     if ( fanionSetat ) {
222         saltRamificatie( deplasamentul );
223     }
224 },
225 i31 : function() {
226     var paginaZero = coboaraUnOctetDinMemorie();
227     var valoarea    = memorie
228         .memoriaCitesteCuvantul( paginaZero );
229     var adresa      = valoarea + registrulY;
230     registrulA &= memorie.memoriaCiteste( adresa );
231     AND();
232 },
233 i35 : function() {
234     var adresa
235         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
236     registrulA &= memorie.memoriaCiteste( adresa );
237     AND();
238 },
239 i36 : function() {
240     var fanionSetat = fanionulCsetat();
241     var adresa
242         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
243     var valoarea    = memorie.memoriaCiteste( adresa );
244     setareFanionulCdeLaBitul7( valoarea );
245     valoarea = (valoarea << 1) & 0xff;
246     valoarea |= fanionSetat;
247     memorie.memoriaStocheazaOctetul( adresa, valoarea );
248     ROL( valoarea );
249 },
250 i38 : function() {
251     SEC();
252 },
253 i39 : function() {
254     var adresa = coboaraUnCuvantDinMemorie() + registrulY;
255     var valoarea = memorie.memoriaCiteste( adresa );
256     registrulA &= valoarea;
257     AND();
258 },
259 i3d : function() {
260     var adresa = coboaraUnCuvantDinMemorie() + registrulX;
261     var valoarea = memorie.memoriaCiteste( adresa );
262     registrulA &= valoarea;

```

```

263         AND ();
264     },
265     i3e : function () {
266         var fanionSetat = fanionulCsetat();
267         var adresa = coboaraUnCuvantDinMemorie() + registrulX;
268         var valoarea = memorie.memoriaCiteste( adresa );
269         setareFanionulCdeLaBitul7( valoarea );
270         valoarea = (valoarea << 1) & 0xff;
271         valoarea |= fanionSetat;
272         memorie.memoriaStocheazaOctetul( adresa, valoarea );
273         ROL( valoarea );
274     },
275     i40 : function () { /* RTI */
276         var valoarea1 = coboaraDinStiva();
277         var valoarea2 = coboaraDinStiva();
278         var valoarea3 = (coboaraDinStiva()) << 8;
279         registrulP = valoarea1 | 0x30;
280         registrulPC = valoarea2 | valoarea3;
281     },
282     i41 : function () {
283         var paginaZero
284             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
285         var valoarea
286             = memorie.memoriaCitesteCuvantul( paginaZero );
287         registrulA ^= memorie.memoriaCiteste( valoarea );
288         EOR();
289     },
290     i42 : function () { /* WDM */
291         var valoarea = coboaraUnOctetDinMemorie();
292         if ( valoarea === 0 ) {
293             auxiliarAfiseazaMesajul(
294                 String.fromCharCode( registrulA )
295             );
296         }
297     },
298     i45 : function () {
299         var adresa = coboaraUnOctetDinMemorie() & 0xff;
300         var valoarea = memorie.memoriaCiteste( adresa );
301         registrulA ^= valoarea;
302         EOR();
303     },
304     i46 : function () {
305         var adresa = coboaraUnOctetDinMemorie() & 0xff;
306         var valoarea = memorie.memoriaCiteste( adresa );
307         setareFanionulCdeLaBitul0( valoarea );
308         valoarea >>= 1;
309         memorie.memoriaStocheazaOctetul( adresa, valoarea );
310         LSR( valoarea );
311     },
312     i48 : function () { /* PHA */
313         urcaPeStiva( registrulA );
314     },
315     i49 : function () {
316         registrulA ^= coboaraUnOctetDinMemorie();

```

```

317         EOR();
318     },
319     i4a : function() {
320         setareFanionulCdeLaBitul0( registrulA );
321         registrulA >>= 1;
322         LSR( registrulA );
323     },
324     i4c : function() {                                     /* JMP */
325         registrulPC = coboaraUnCuvantDinMemorie();
326     },
327     i4d : function() {
328         var adresa = coboaraUnCuvantDinMemorie();
329         var valoarea = memorie.memoriaCiteste( adresa );
330         registrulA ^= valoarea;
331         EOR();
332     },
333     i4e : function() {
334         var adresa = coboaraUnCuvantDinMemorie();
335         var valoarea = memorie.memoriaCiteste( adresa );
336         setareFanionulCdeLaBitul0( valoarea );
337         valoarea >>= 1;
338         memorie.memoriaStocheazaOctetul( adresa, valoarea );
339         LSR( valoarea );
340     },
341     i50 : function() {                                     /* BVC */
342         var deplasamentul = coboaraUnOctetDinMemorie();
343         var fanionSetat = fanionulVsetat();
344         if ( !fanionSetat ) {
345             saltRamificatie( deplasamentul );
346         }
347     },
348     i51 : function() {
349         var paginaZero = coboaraUnOctetDinMemorie();
350         var valoarea = memorie
351             .memoriaCitesteCuvantul( paginaZero )
352             + registrulY;
353         registrulA ^= memorie.memoriaCiteste( valoarea );
354         EOR();
355     },
356     i55 : function() {
357         var adresa
358             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
359         var valoarea = memorie.memoriaCiteste( adresa );
360         registrulA ^= valoarea;
361         EOR();
362     },
363     i56 : function() {
364         var adresa
365             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
366         var valoarea = memorie.memoriaCiteste( adresa );
367         setareFanionulCdeLaBitul0( valoarea );
368         valoarea >>= 1;
369         memorie.memoriaStocheazaOctetul( adresa, valoarea );
370         LSR( valoarea );

```

```

371     },
372     i58 : function () {                                     /* CLI */
373         registrulP &= ~0x04;
374         throw new Error (
375             "&#206;ntreruperi neimplementate!"
376         );
377     },
378     i59 : function () {
379         var adresa
380             = coboaraUnCuvantDinMemorie () + registrulY;
381         var valoarea = memorie.memoriaCiteste ( adresa );
382         registrulA ^= valoarea;
383         EOR ();
384     },
385     i5d : function () {
386         var adresa
387             = coboaraUnCuvantDinMemorie () + registrulX;
388         var valoarea = memorie.memoriaCiteste ( adresa );
389         registrulA ^= valoarea;
390         EOR ();
391     },
392     i5e : function () {
393         var adresa
394             = coboaraUnCuvantDinMemorie () + registrulX;
395         var valoarea = memorie.memoriaCiteste ( adresa );
396         setareFanionulCdeLaBitul0 ( valoarea );
397         valoarea >>= 1;
398         memorie.memoriaStocheazaOctetul ( adresa, valoarea );
399         LSR ( valoarea );
400     },
401     i60 : function () {                                     /* RTS */
402         var valoarea1 = coboaraDinStiva ();
403         var valoarea2 = (coboaraDinStiva ()) << 8;
404         registrulPC = (valoarea1 | valoarea2) + 1;
405     },
406     i61 : function () {                                     /* ADC */
407         var paginaZero
408             = (coboaraUnOctetDinMemorie () + registrulX) & 0xff;
409         var adresa
410             = memorie.memoriaCitesteCuvantul ( paginaZero );
411         var valoarea = memorie.memoriaCiteste ( adresa );
412         testeazaADC ( valoarea );
413     },
414     i65 : function () {                                     /* ADC */
415         var adresa = coboaraUnOctetDinMemorie ();
416         var valoarea = memorie.memoriaCiteste ( adresa );
417         testeazaADC ( valoarea );
418     },
419     i66 : function () {
420         var fanionSetat = fanionulCsetat ();
421         var adresa = coboaraUnOctetDinMemorie ();
422         var valoarea = memorie.memoriaCiteste ( adresa );
423         setareFanionulCdeLaBitul0 ( valoarea );
424         valoarea >>= 1;

```

```

425         if ( fanionSetat ) {
426             valoarea |= 0x80;
427         }
428         memorie.memoriaStocheazaOctetul( adresa, valoarea );
429         ROR( valoarea );
430     },
431     i68 : function() {                                     /* PLA */
432         registrula = coboaraDinStiva();
433         setareFanioaneleNZVpentruRegistrula();
434     },
435     i69 : function() {                                     /* ADC */
436         var valoarea = coboaraUnOctetDinMemorie();
437         testeazaADC( valoarea );
438     },
439     i6a : function() {
440         var fanionSetat = fanionulCsetat();
441         setareFanionulCdeLaBitul0( registrula );
442         registrula >>= 1;
443         if ( fanionSetat ) {
444             registrula |= 0x80;
445         }
446         ROR( registrula );
447     },
448     i6c : function() {                                     /* JMP */
449         var valoarea = coboaraUnCuvantDinMemorie();
450         registrulPC = memorie
451             .memoriaCitesteCuvantul( valoarea );
452     },
453     i6d : function() {                                     /* ADC */
454         var adresa = coboaraUnCuvantDinMemorie();
455         var valoarea = memorie.memoriaCiteste( adresa );
456         testeazaADC( valoarea );
457     },
458     i6e : function() {
459         var fanionSetat = fanionulCsetat();
460         var adresa = coboaraUnCuvantDinMemorie();
461         var valoarea = memorie.memoriaCiteste( adresa );
462         setareFanionulCdeLaBitul0( valoarea );
463         valoarea >>= 1;
464         if ( fanionSetat ) {
465             valoarea |= 0x80;
466         }
467         memorie.memoriaStocheazaOctetul( adresa, valoarea );
468         ROR( valoarea );
469     },
470     i70 : function() {                                     /* BVS */
471         var deplasamentul = coboaraUnOctetDinMemorie();
472         var fanionSetat = fanionulVsetat();
473         if ( fanionSetat ) {
474             saltRamificatie( deplasamentul );
475         }
476     },
477     i71 : function() {                                     /* ADC */
478         var paginaZero = coboaraUnOctetDinMemorie();

```

```

479         var adresa      = memorie
480         .memoriaCitesteCuvantul( paginaZero );
481         var valoarea = memorie
482         .memoriaCiteste( adresa + registrulY );
483         testeazaADC( valoarea );
484     },
485     i75 : function() {                                /* ADC */
486         var adresa
487         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
488         var valoarea = memorie.memoriaCiteste( adresa );
489         testeazaADC( valoarea );
490     },
491     i76 : function() {
492         var fanionSetat = fanionulCsetat();
493         var adresa
494         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
495         var valoarea = memorie.memoriaCiteste( adresa );
496         setareFanionulCdeLaBitul0( valoarea );
497         valoarea >>= 1;
498         if ( fanionSetat ) {
499             valoarea |= 0x80;
500         }
501         memorie.memoriaStocheazaOctetul( adresa, valoarea );
502         ROR( valoarea );
503     },
504     i78 : function() {                                /* SEI */
505         registrulP |= 0x04;
506         throw new Error(
507             "&#206;ntreruperi neimplementate!"
508         );
509     },
510     i79 : function() {                                /* ADC */
511         var adresa      = coboaraUnCuvantDinMemorie();
512         var valoarea = memorie
513         .memoriaCiteste( adresa + registrulY );
514         testeazaADC( valoarea );
515     },
516     i7d : function() {                                /* ADC */
517         var adresa      = coboaraUnCuvantDinMemorie();
518         var valoarea = memorie
519         .citeste( adresa + registrulX );
520         testeazaADC( valoarea );
521     },
522     i7e : function() {
523         var fanionSetat = fanionulCsetat();
524         var adresa
525         = coboaraUnCuvantDinMemorie() + registrulX;
526         var valoarea      = memorie.memoriaCiteste( adresa );
527         setareFanionulCdeLaBitul0( valoarea );
528         valoarea >>= 1;
529         if ( fanionSetat ) {
530             valoarea |= 0x80;
531         }
532         memorie.memoriaStocheazaOctetul( adresa, valoarea );

```

```

533         ROR( valoarea );
534     },
535     i81 : function() {                                /* STA */
536         var paginaZero
537         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
538         var adresa = memorie
539         .memoriaCitesteCuvantul( paginaZero );
540         memorie.memoriaStocheazaOctetul( adresa, registrula );
541     },
542     i84 : function() {                                /* STY */
543         var adresa = coboaraUnOctetDinMemorie();
544         memorie.memoriaStocheazaOctetul( adresa, registruly );
545     },
546     i85 : function() {                                /* STA */
547         var adresa = coboaraUnOctetDinMemorie();
548         memorie.memoriaStocheazaOctetul( adresa, registrula );
549     },
550     i86 : function() {                                /* STX */
551         var adresa = coboaraUnOctetDinMemorie();
552         memorie.memoriaStocheazaOctetul( adresa, registrulx );
553     },
554     i88 : function() {                                /* DEY */
555         var valoarea = registruly - 1;
556         registruly = valoarea & 0xff;
557         setareFanioaneleNZVpentruRegistruly();
558     },
559     i8a : function() {                                /* TXA */
560         registrula = registrulx & 0xff;
561         setareFanioaneleNZVpentruRegistrula();
562     },
563     i8c : function() {                                /* STY */
564         var adresa = coboaraUnCuvantDinMemorie();
565         memorie.memoriaStocheazaOctetul( adresa, registruly );
566     },
567     i8d : function() {                                /* STA */
568         var adresa = coboaraUnCuvantDinMemorie();
569         memorie.memoriaStocheazaOctetul( adresa, registrula );
570     },
571     i8e : function() {                                /* STX */
572         var adresa = coboaraUnCuvantDinMemorie();
573         memorie.memoriaStocheazaOctetul( adresa, registrulx );
574     },
575     i90 : function() {                                /* BCC */
576         var deplasamentul = coboaraUnOctetDinMemorie();
577         var fanionSetat = fanionulCsetat();
578         if ( !fanionSetat ) {
579             saltRamificatie( deplasamentul );
580         }
581     },
582     i91 : function() {                                /* STA */
583         var paginaZero = coboaraUnOctetDinMemorie();
584         var adresa = memorie
585         .memoriaCitesteCuvantul( paginaZero )
586         + registruly;

```

```

587         memorie.memoriaStocheazaOctetul( adresa, registrula );
588     },
589     i94 : function() {                                /* STY */
590         var adresa
591             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
592         memorie.memoriaStocheazaOctetul( adresa, registruly );
593     },
594     i95 : function() {                                /* STA */
595         var adresa
596             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
597         memorie.memoriaStocheazaOctetul( adresa, registrula );
598     },
599     i96 : function() {                                /* STX */
600         var adresa
601             = (coboaraUnOctetDinMemorie() + registruly) & 0xff;
602         memorie.memoriaStocheazaOctetul( adresa, registrulX );
603     },
604     i98 : function() {                                /* TYA */
605         registrula = registruly & 0xff;
606         setareFanioaneleNZVpentruRegistrula();
607     },
608     i99 : function() {                                /* STA */
609         var adresa = coboaraUnCuvantDinMemorie() + registruly;
610         memorie.memoriaStocheazaOctetul( adresa, registrula );
611     },
612     i9a : function() {                                /* TXS */
613         registrulSP = registrulX & 0xff;
614     },
615     i9d : function() {                                /* STA */
616         var adresa = coboaraUnCuvantDinMemorie();
617         memorie.memoriaStocheazaOctetul(
618             adresa + registrulX, registrula
619         );
620     },
621     ia0 : function() {
622         registruly = coboaraUnOctetDinMemorie();
623         LDY();
624     },
625     ia1 : function() {
626         var paginaZero
627             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
628         var adresa
629             = memorie.memoriaCitesteCuvantul( paginaZero );
630         registrula = memorie.memoriaCiteste( adresa );
631         LDA();
632     },
633     ia2 : function() {
634         registrulX = coboaraUnOctetDinMemorie();
635         LDX();
636     },
637     ia4 : function() {
638         var adresa = coboaraUnOctetDinMemorie();
639         registruly = memorie.memoriaCiteste( adresa );
640         LDY();

```

```

641     },
642     ia5 : function () {
643         var adresa = coboaraUnOctetDinMemorie();
644         registrulA = memorie.memoriaCiteste( adresa );
645         LDA();
646     },
647     ia6 : function () {
648         var adresa = coboaraUnOctetDinMemorie();
649         registrulX = memorie.memoriaCiteste( adresa );
650         LDH();
651     },
652     ia8 : function () {                                     /* TAY */
653         registrulY = registrulA & 0xff;
654         setareFanioaneleNZVpentruRegistrulY();
655     },
656     ia9 : function () {
657         registrulA = coboaraUnOctetDinMemorie();
658         LDA();
659     },
660     iaa : function () {                                     /* TAX */
661         registrulX = registrulA & 0xff;
662         setareFanioaneleNZVpentruRegistrulX();
663     },
664     iac : function () {
665         var adresa = coboaraUnCuvantDinMemorie();
666         registrulY = memorie.memoriaCiteste( adresa );
667         LDY();
668     },
669     iad : function () {
670         var adresa = coboaraUnCuvantDinMemorie();
671         registrulA = memorie.memoriaCiteste( adresa );
672         LDA();
673     },
674     iae : function () {
675         var adresa = coboaraUnCuvantDinMemorie();
676         registrulX = memorie.memoriaCiteste( adresa );
677         LDH();
678     },
679     ib0 : function () {                                     /* BCS */
680         var deplasamentul = coboaraUnOctetDinMemorie();
681         var fanionSetat = fanionulCsetat();
682         if ( fanionSetat ) {
683             saltRamificatie( deplasamentul );
684         }
685     },
686     ib1 : function () {
687         var paginaZero = coboaraUnOctetDinMemorie();
688         var adresa = memorie
689             .memoriaCitesteCuvantul( paginaZero )
690             + registrulY;
691         registrulA = memorie.memoriaCiteste( adresa );
692         LDA();
693     },
694     ib4 : function () {

```

[illegible]

```

749         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
750     var adresa = memorie
751     .memoriaCitesteCuvantul( paginaZero );
752     var valoarea = memorie.memoriaCiteste( adresa );
753     faComparatia( registrula, valoarea );
754 },
755 ic4 : function() {                                /* CPY */
756     var adresa = coboaraUnOctetDinMemorie();
757     var valoarea = memorie.memoriaCiteste( adresa );
758     faComparatia( registruly, valoarea );
759 },
760 ic5 : function() {                                /* CPA */
761     var adresa = coboaraUnOctetDinMemorie();
762     var valoarea = memorie.memoriaCiteste( adresa );
763     faComparatia( registrula, valoarea );
764 },
765 ic6 : function() {
766     var paginaZero = coboaraUnOctetDinMemorie();
767     DEC( paginaZero );
768 },
769 ic8 : function() {                                /* INY */
770     registruly = (registruly + 1) & 0xff;
771     setareFanioaneleNZVpentruRegistruly();
772 },
773 ic9 : function() {                                /* CMP */
774     var valoarea = coboaraUnOctetDinMemorie();
775     faComparatia( registrula, valoarea );
776 },
777 ica : function() {                                /* DEX */
778     registrulX = (registrulX - 1) & 0xff;
779     setareFanioaneleNZVpentruRegistrulX();
780 },
781 icc : function() {                                /* CPY */
782     var adresa = coboaraUnCuvantDinMemorie();
783     var valoarea = memorie.memoriaCiteste( adresa );
784     faComparatia( registruly, valoarea );
785 },
786 icd : function() {                                /* CPA */
787     var adresa = coboaraUnCuvantDinMemorie();
788     var valoarea = memorie.memoriaCiteste( adresa );
789     faComparatia( registrula, valoarea );
790 },
791 ice : function() {
792     var adresa = coboaraUnCuvantDinMemorie();
793     DEC( adresa );
794 },
795 id0 : function() {                                /* BNE */
796     var deplasamentul = coboaraUnOctetDinMemorie();
797     var fanionSetat = fanionulZsetat();
798     if ( !fanionSetat ) {
799         saltRamificatie( deplasamentul );
800     }
801 },
802 id1 : function() {                                /* CMP */

```

```

803         var paginaZero = coboaraUnOctetDinMemorie();
804         var adresa      = memorie
805             .memoriaCitesteCuvantul( paginaZero ) + registrulY;
806         var valoarea     = memorie.memoriaCiteste( adresa );
807         faComparatia( registrulA, valoarea );
808     },
809     id5 : function() {                                     /* CMP */
810         var adresa
811             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
812         var valoarea = memorie.memoriaCiteste( adresa );
813         faComparatia( registrulA, valoarea );
814     },
815     id6 : function() {
816         var adresa
817             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
818         DEC( adresa );
819     },
820     id8 : function() {                                     /* CLD */
821         registrulP &= 0xf7;
822     },
823     id9 : function() {                                     /* CMP */
824         var adresa
825             = coboaraUnCuvantDinMemorie() + registrulY;
826         var valoarea = memorie.memoriaCiteste( adresa );
827         faComparatia( registrulA, valoarea );
828     },
829     idd : function() {                                     /* CMP */
830         var adresa
831             = coboaraUnCuvantDinMemorie() + registrulX;
832         var valoarea = memorie.memoriaCiteste( adresa );
833         faComparatia( registrulA, valoarea );
834     },
835     ide : function() {
836         var adresa
837             = coboaraUnCuvantDinMemorie() + registrulX;
838         DEC( adresa );
839     },
840     ie0 : function() {                                     /* CPX */
841         var valoarea = coboaraUnOctetDinMemorie();
842         faComparatia( registrulX, valoarea );
843     },
844     ie1 : function() {                                     /* SBC */
845         var paginaZero
846             = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
847         var adresa
848             = memorie.memoriaCitesteCuvantul( paginaZero );
849         var valoarea = memorie.memoriaCiteste( adresa );
850         testeazaSBC( valoarea );
851     },
852     ie4 : function() {                                     /* CPX */
853         var adresa = coboaraUnOctetDinMemorie();
854         var valoarea = memorie.memoriaCiteste( adresa );
855         faComparatia( registrulX, valoarea );
856     },

```

```

857 ie5 : function() {                                     /* SBC */
858     var adresa = coboaraUnOctetDinMemorie();
859     var valoarea = memorie.memoriaCiteste( adresa );
860     testeazaSBC( valoarea );
861 },
862 ie6 : function() {
863     var paginaZero = coboaraUnOctetDinMemorie();
864     INC( paginaZero );
865 },
866 ie8 : function() {                                     /* INX */
867     registrulX = (registrulX + 1) & 0xff;
868     setareFanioaneleNZVpentruRegistrulX();
869 },
870 ie9 : function() {                                     /* SBC */
871     var valoarea = coboaraUnOctetDinMemorie();
872     testeazaSBC( valoarea );
873 },
874 iea : function() {                                     /* NOP */
875 },
876 iec : function() {                                     /* CPX */
877     var adresa = coboaraUnCuvantDinMemorie();
878     var valoarea = memorie.memoriaCiteste( adresa );
879     faComparatia( registrulX, valoarea );
880 },
881 ied : function() {                                     /* SBC */
882     var adresa = coboaraUnCuvantDinMemorie();
883     var valoarea = memorie.memoriaCiteste( adresa );
884     testeazaSBC( valoarea );
885 },
886 iee : function() {
887     var adresa = coboaraUnCuvantDinMemorie();
888     INC( adresa );
889 },
890 if0 : function() {                                     /* BEQ */
891     var deplasamentul = coboaraUnOctetDinMemorie();
892     var fanionSetat = fanionulZsetat();
893     if ( fanionSetat ) {
894         saltRamificatie( deplasamentul );
895     }
896 },
897 if1 : function() {                                     /* SBC */
898     var paginaZero = coboaraUnOctetDinMemorie();
899     var adresa = memorie
900         .memoriaCitesteCuvantul( paginaZero );
901     var valoarea = memorie
902         .memoriaCiteste( adresa + registrulY );
903     testeazaSBC( valoarea );
904 },
905 if5 : function() {                                     /* SBC */
906     var adresa
907         = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
908     var valoarea = memorie.memoriaCiteste( adresa );
909     testeazaSBC( valoarea );
910 },

```

```

911  if6 : function () {
912      var adresa
913          = (coboaraUnOctetDinMemorie() + registrulX) & 0xff;
914      INC( adresa );
915  },
916  if8 : function () {                                /* SED */
917      registrulP |= 0x08;
918  },
919  if9 : function () {                                /* SBC */
920      var adresa      = coboaraUnCuvantDinMemorie();
921      var valoarea    = memorie
922          .memoriaCiteste( adresa + registrulY );
923      testeazaSBC( valoarea );
924  },
925  ifd : function () {                                /* SBC */
926      var adresa      = coboaraUnCuvantDinMemorie();
927      var valoarea    = memorie
928          .memoriaCiteste( adresa + registrulX );
929      testeazaSBC( valoarea );
930  },
931  ife : function () {
932      var adresa = coboaraUnCuvantDinMemorie() + registrulX;
933      INC( adresa );
934  },
935  ieroare : function () {
936      auxiliarAfiseazaMesajul (
937          "Adresa $"
938          + auxiliarConvertescAdresaLaHexa( registrulPC )
939          + " - cod operațional necunoscut."
940      );
941      codInExecutie = false;
942  }
943 };

```

Instrucțiunea WDM este o pseudo-operatie în emulatorul Easy6502. Vezi exemplul dat în continuare.

```

lda #$be ; afiseaza "3/4"
          ; (Unicode: \u00be, UTF-8: &#190;)
          ; in mesagerie
wdm #0   ; opcodul 0x42

```



## Referințe Bibliografice

1. `$ ( . . . )`: *Launching Code on Document Ready*.  
[https://learn.jquery.com/about-jquery/how-jquery-works/  
#launching-code-on-document-ready](https://learn.jquery.com/about-jquery/how-jquery-works/#launching-code-on-document-ready)
2. Analiza lexicală: *Lexical analysis*.  
[https://en.wikipedia.org/wiki/Lexical\\_analysis](https://en.wikipedia.org/wiki/Lexical_analysis)
3. Arhitectura setului de instrucțiuni: *Instruction set architecture*.  
[https://en.wikipedia.org/wiki/Instruction\\_set\\_architecture](https://en.wikipedia.org/wiki/Instruction_set_architecture)
4. Athanasiu, I.; Pănoiu, A.: *Microprocesoarele 8086/80286/80386. Programarea în limbaj de asamblare*. Editura TEORA, București (1992)
5. Atribute în DOM: *Interface Element*.  
<https://dom.spec.whatwg.org/#interface-element>
6. Blocuri de conținut: *Block-level content*.  
[https://developer.mozilla.org/en-US/docs/Glossary/  
Block-level\\_content](https://developer.mozilla.org/en-US/docs/Glossary/Block-level_content)
7. `br`: *The br element*.  
[https://html.spec.whatwg.org/multipage/  
text-level-semantics.html#the-br-element](https://html.spec.whatwg.org/multipage/text-level-semantics.html#the-br-element)
8. Codurile RGB hexazecimale ale culorilor în CSS: *<named-color>*.  
<https://developer.mozilla.org/en-US/docs/Web/CSS/named-color>
9. Creative Commons. CC BY 4.0: *ATtribution 4.0 INTERNATIONAL Deed*.  
<https://creativecommons.org/licenses/by/4.0/>
10. CSS: *Cascading Style Sheet*.  
<https://developer.mozilla.org/en-US/docs/Web/CSS>
11. Dancea I.: *Dispozitivele aritmetice ale calculatoarelor numerice*. Editura tehnică, București (1971)
12. Depășirea plajei de valori: *Integer overflow*.  
[https://en.wikipedia.org/wiki/Integer\\_overflow](https://en.wikipedia.org/wiki/Integer_overflow)
13. Digramă, trigramă: *Digraphs and trigraphs (programming)*.  
[https://en.wikipedia.org/wiki/Digraphs\\_and\\_trigraphs\\_\(programming\)](https://en.wikipedia.org/wiki/Digraphs_and_trigraphs_(programming))
14. DOM: *Introduction to "The DOM"*.  
<https://dom.spec.whatwg.org/#introduction-to-the-dom>
15. Elemente HTML: *HTML elements reference*.  
<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>
16. Evenimente în DOM: *blur*.  
<https://w3c.github.io/uievents/#event-type-blur>
17. Evenimente în DOM: *keydown*.  
<https://w3c.github.io/uievents/#event-type-keydown>

18. Evenimente în DOM: *keypress*.  
<https://w3c.github.io/uievents/#event-type-keypress>
19. Evenimente în DOM: *keypress event*.  
<https://api.jquery.com/keypress/>
20. Evenimente în DOM via jQuery: *Event*.  
<https://api.jquery.com/Types/#Event>
21. Expresii regulate: *Regular expressions*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_expressions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions)
22. Expresii regulate. Cuantificatori: *Quantifier*: ...  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular\\_expressions/Quantifier](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions/Quantifier)
23. Expresii regulate. Dezactivarea caracterelor speciale: *Character escape*: ...  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular\\_expressions/Character\\_escape](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions/Character_escape)
24. Expresii regulate. Fanieane: *Regular expressions. Advanced searching with flags*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_expressions#advanced\\_searching\\_with\\_flags](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions#advanced_searching_with_flags)
25. Expresii regulate. Grupuri de captură: *Capturing group*: (...).  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular\\_expressions/Capturing\\_group](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions/Capturing_group)
26. Eyes, D.; Lichty, R.: *Programming the 65816 Including the 6502, 65C02, and 65802*. Prentice Hall Press, New York (1986)
27. Firefox: *Firefox Browsers*.  
<https://www.mozilla.org/en-US/firefox/>
28. Funcții anonime: *Static semantics: IsAnonymousFunctionDefinition(expr)*.  
<https://tc39.es/ecma262/#sec-isanonymousfunctiondefinition>
29. Funcții de prima clasă: *First-class function*.  
[https://developer.mozilla.org/en-US/docs/Glossary/First-class\\_Function](https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function)
30. Funcții imbricate: *Nested functions and closures*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#nested\\_functions\\_and\\_closures](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#nested_functions_and_closures)
31. Funcții interioare: *Block-Level Function Declarations Web Legacy Compatibility Semantics*.  
<https://tc39.es/ecma262/#sec-block-level-function-declarations-web-legacy-compatibility-semantics>
32. Funcții JavaScript: *Math.random()*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Math/random](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random)
33. Funcții JavaScript: *Object.prototype.toString()*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object/toString](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString)
34. Funcții JavaScript: *String.prototype.replace()*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/String/replace](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace)
35. Funcții jQuery: *.each( function )*.  
<https://api.jquery.com/each/#each-function>
36. Funcții jQuery: *.filter( function )*.  
<https://api.jquery.com/filter/#filter-function>
37. Funcții jQuery: *.find( selector )*.  
<https://api.jquery.com/each/#each-function>
38. Funcții jQuery: *.on( events [, selector] [, data], handler )*.  
<https://api.jquery.com/on/#on-events-selector-data-handler>
39. Funcții jQuery: *.ready( handler )*.  
<https://api.jquery.com/ready/#ready-handler>

40. Funcții jQuery: `.val()`, `.val( value )`.  
<https://api.jquery.com/val/>
41. Funcții săgeată: *Arrow Function Definitions*.  
<https://tc39.es/ecma262/#sec-arrow-function-definitions>
42. Gramatică folosită de MASM în format BNF. Restricții privind *non-terminalul* LabelDef: *Microsoft Macro Assembler BNF Grammar*.  
<https://learn.microsoft.com/en-us/cpp/assembly/masm/masm-bnf-grammar?view=msvc-170>
43. HTML: *HyperText Markup Language*.  
<https://developer.mozilla.org/en-US/docs/Web/HTML>
44. Hyde, R.: *Using 6502 Assembly Language. How to Program the Apple II*. DataMost, Inc., Chatsworth (1982). Disponibilă online la adresa:  
<https://www.randallhyde.com/AssemblyLanguage/A2%20Hyde%206502%20Asm%20Lang.pdf>
45. input: *The input element*.  
<https://html.spec.whatwg.org/multipage/input.html>
46. Încărcător, program încărcător: *Loader (computing)*.  
[https://en.wikipedia.org/wiki/Loader\\_\(computing\)](https://en.wikipedia.org/wiki/Loader_(computing))
47. JavaScript, JS: *JavaScript*.  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>
48. jQuery: *jQuery: write less, do more*.  
<https://jquery.com/>
49. Leventhal, L.A.: *6502 Assembly Language Programming, Second Edition*. Osborne/McGraw-Hill, Berkeley (1986)
50. Macro-instrucțiuni, macro-uri: *Macro (computer science)*.  
[https://en.wikipedia.org/wiki/Macro\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Macro_(computer_science))
51. Mateescu, Ad.: *Analiza și sinteza circuitelor electrice*. Editura didactică și pedagogică, București (1975)
52. Mărimi CSS: *CSS Values and Units Module Level 4*.  
<https://drafts.csswg.org/css-values/>
53. Metode în DOM: *Window: setInterval() method*.  
<https://developer.mozilla.org/en-US/docs/Web/API/Window/setInterval>
54. Microprocesoarele din seria 6500, Synertek: *SY6500. 8-Bit Microprocessor Family*.  
<https://www.princeton.edu/~mae412/HANDOUTS/Datasheets/6502.pdf>
55. Muntean, I.: *Sinteza automatelor finite*. Editura tehnică, București (1977)
56. Numere magice: *Magic number (programming)*.  
[https://en.wikipedia.org/wiki/Magic\\_number\\_\(programming\)](https://en.wikipedia.org/wiki/Magic_number_(programming))
57. Obiecte funcție: *Function Objects*.  
<https://tc39.es/ecma262/#sec-function-objects>
58. Obiecte JavaScript: *Introducing JavaScript objects*.  
<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects>
59. Obiecte JavaScript. Definirea metodelor: *Method definitions*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method\\_definitions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method_definitions)
60. Pop, V.; Popovici, V.: *Circuite de comutare aplicate în calculatoarele electronice*. Editura Facla, Cluj (1976)
61. Pozițiile hexazecimale ale literelor românești în Unicode: *Romanian alphabet. Unicode and HTML*.  
[https://en.wikipedia.org/wiki/Romanian\\_alphabet#Unicode\\_and\\_HTML](https://en.wikipedia.org/wiki/Romanian_alphabet#Unicode_and_HTML)
62. pre: *The pre element*.  
<https://html.spec.whatwg.org/multipage/grouping-content.html#the-pre-element>

63. Programarea orientată-obiect. Interfețe: *Object-oriented programming*.  
[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented\\_programming](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming)
64. Proprietăți dezafectate ale unor evenimente: *UIEvent: which property*.  
<https://developer.mozilla.org/en-US/docs/Web/API/UIEvent/which>
65. Redarea unei pagini Web: *Populating the page: how browsers work*.  
[https://developer.mozilla.org/en-US/docs/Web/Performance/How\\_browsers\\_work](https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work)
66. Referințe HTML cu nume ale *unicodurilor* de caractere: *Named character references*.  
<https://html.spec.whatwg.org/multipage/named-characters.html#named-character-references>
67. Relocarea codului-mașină în memorie: *Relocation (computing)*.  
[https://en.wikipedia.org/wiki/Relocation\\_\(computing\)](https://en.wikipedia.org/wiki/Relocation_(computing))
68. Rezoluția: *Window: devicePixelRatio property*.  
<https://developer.mozilla.org/en-US/docs/Web/API/Window/devicePixelRatio>
69. SAU exclusiv, XOR: *Exclusive or*.  
[https://en.wikipedia.org/wiki/Exclusive\\_or](https://en.wikipedia.org/wiki/Exclusive_or)
70. Scanlon, L.J.: *6502 Software Design*. Howard W. Sams & Co., Inc., Indianapolis (1980)
71. Scrierea cămilă: *Camel case*.  
[https://developer.mozilla.org/en-US/docs/Glossary/Camel\\_case](https://developer.mozilla.org/en-US/docs/Glossary/Camel_case)
72. script: *The script element*.  
<https://html.spec.whatwg.org/multipage/scripting.html#the-script-element>
73. Selectoare jQuery: *Category: Selectors*.  
<https://api.jquery.com/category/selectors/>
74. Setul de unelte al dezvoltatorului Firefox: *Toolbox*.  
[https://firefox-source-docs.mozilla.org/devtools-user/tools\\_toolbox/](https://firefox-source-docs.mozilla.org/devtools-user/tools_toolbox/)
75. Simovici, D.: *Limbaje formale și tehnici de compilare*. Editura didactică și pedagogică, București (1978)
76. Spațiile goale: *How whitespace is handled by HTML, CSS, and in the DOM*.  
[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Whitespace](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Whitespace)
77. spellcheck: *Spelling and grammar checking*.  
<https://html.spec.whatwg.org/multipage/interaction.html#spelling-and-grammar-checking>
78. spellcheck în Firefox: *spellcheck*.  
[https://developer.mozilla.org/en-US/docs/Web/HTML/Global\\_attributes/spellcheck](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/spellcheck)
79. Suprascrierea metodelor: *Inheritance and the prototype chain. Inheriting "methods"*.  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance\\_and\\_the\\_prototype\\_chain#inheriting\\_methods](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain#inheriting_methods)
80. Tipuri de date în JavaScript: *ECMAScript Data Types and Values*.  
<https://tc39.es/ecma262/#sec-ecmascript-data-types-and-values>
81. Verificator de standard CSS: *The W3C CSS Validation Service*.  
[https://jigsaw.w3.org/css-validator/#validate\\_by\\_upload+with\\_options](https://jigsaw.w3.org/css-validator/#validate_by_upload+with_options)
82. Verificator de standard HTML: *The W3C Markup Validation Service*.  
[https://validator.w3.org/#validate\\_by\\_upload+with\\_options](https://validator.w3.org/#validate_by_upload+with_options)

# Index

- § ( . . . ), 16
- 0x600, 55
- 64K, 54
- adresarea memoriei, 44, 46
- ALU, 32
- arhitectura, 22, 30–32, 44, 56, 57, 60, 67, 73
  
- cod-mașină hexazecimal, 33
- codificarea zecimal-binară, 41
- compilator, 8
- complementul lui 2, 60
- contiguu, 56
- CPU, 31
- cvartet, 18
  
- debugging*, 2
- deversare hexazecimală, 22, 35, 40
- dezafectat, 20
  
- element HTML, 8
- expresie regulată, 17
- extensie, 1
  
- fanion, 32, 40, 43, 44, 67–71, 73
- formatul *little endian*, 24
- funcție *anonimă*, 16
- funcție de reacție, 20
- funcție *săgeată*, 16
  
- grup de captură, 30, 35
  
- input*, 8
- interfață, 24
  
- limbaj de asamblare, vii
  
- macro*, 37
  
- mască de biți, 17
- memoria *de utilizator*, 35, 42, 56
- memoria *grafică*, 22
- microcomputer, viii, 32
- mnemonică, 57
- modulo*, *mod*, 52, 60
  
- număr *magic*, 55
  
- obiect funcție, 11
- obiect JavaScript, 11
- opcod, 32, 60, 82
  
- pagina unu, 74
- pagina zero, 47–50, 52, 53
- parsare, 43
- pixel, 21
- plaja de valori, 18, 54, 60, 74, 79
- platforma *Wayback Machine*, ix, 67
- prima clasă, 11
- pseudo-operație, 40, 41, 99
  
- ramificare, 44
- registru, 27
  
- salt, 44
- scrierea *cămilă*, 11
- standard, 8, 10
- stivă, 74
- subrutină, 75
  
- terminal grafic, 8
- tetradă, 18
  
- unicoduri*, 27
  
- Web Developer Tools, 8