ARHITECTURA SISTEMELOR DE CALCUL
LUCRAREA DE LABORATOR NR. 3

Sisteme multiprocesor. Conectarea şi sincronizarea procesoarelor sistemului multiprocesor

Scopul lucrării

a) Principii de interconectare a procesoerelor în sisteme multiprocesor.
b) Descrierea unui sistem multiprocesor cu două procesoare interconectate prin memorie comună.
c) Sincronizarea software a activităţilor procesosrelor sistemului multiprocesor.

1. Arhitecturi de sisteme multiprocesor

Există două categorii de sisteme multiprocesor : arhitectură *centralizată* şi arhitectură *distribuită* .Cele două soluţii constructive sînt ilustrate în figura 1.

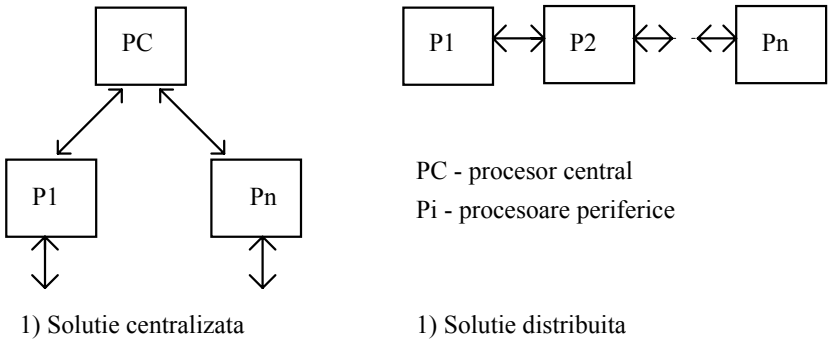


Figura 1. Arhitecturi de sisteme multiprocesor

Soluţia centralizată este utilizată pentru sisteme complexe. Este necesar un mecanism de intercomunicare între procesoare (realizat software sau hardware) care limitează performanţele sistemului.

Pentru soluţia distribuită deciziile se iau local de către procesoarele periferice. Mecanismul de intercomunicare (uzual realizat software) este mai simplu. Este necesară divizarea funcţiilor sistemului în subfuncţii bine determinate care sînt atribuite procesoarelor locale.

În practică se utilizează şi soluţii mixte , cu un procesor central şi mai multe procesoare locale.

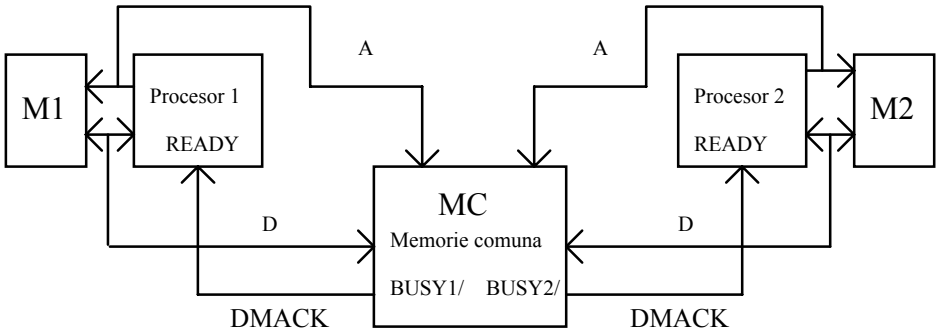
1.1. Soluţii de interconectare

Principalele moduri de conectare între două procesoare sînt :

- cu memorie comună
- cu împărţirea busurilor

1.1.1. Sistem multiprocesor cu 2 procesoare cu memorie comună

Schema sistemului este prezentată în figura 2.



Semnalele de adresa si control pentru memoria comuna nu s-au figurat

MC - (IDT7132 - 2k x 8 dual cu circuitul de arbitrare inclus)

M1, M2 - memorii privata A , D - adrese , date

DMACK - Data Acknowledge (trebuie sa fie 1 pentru functionare normala

Figura 2. Sistem multiprocesor cu memorie comună

Memoria comună este un circuit mai complex ce include memoria propriu-zisă şi un controler care arbitrează conflictele de acces ce pot apare dacă cele două procesoare cer accesul "simultan" la memoria comună.

Conflictele de acces pot apare dacă un procesor citeşte iar celălalt scrie aceeaşi locaţie de memorie sau dacă ambele procesoare scriu în aceaşi locaţie de memorie.

Circuitul de arbitrare a conflictelor de acces are următoarele funcţii :

- compară adresele generate de cele două procesoare pentru memoria comună
- generează semnalele de control pentru memoria comună (de exemplu WE/)
- generează semnale pentru blocarea unui ciclu de acces (cel sosit puţin mai tîrziu); se generează de exemplu un semnal BUSY/ .

În figura 3. este ilustrată o soluţie constructivă fără controler integrat (memoria comună este o memorie RAM uzuală).

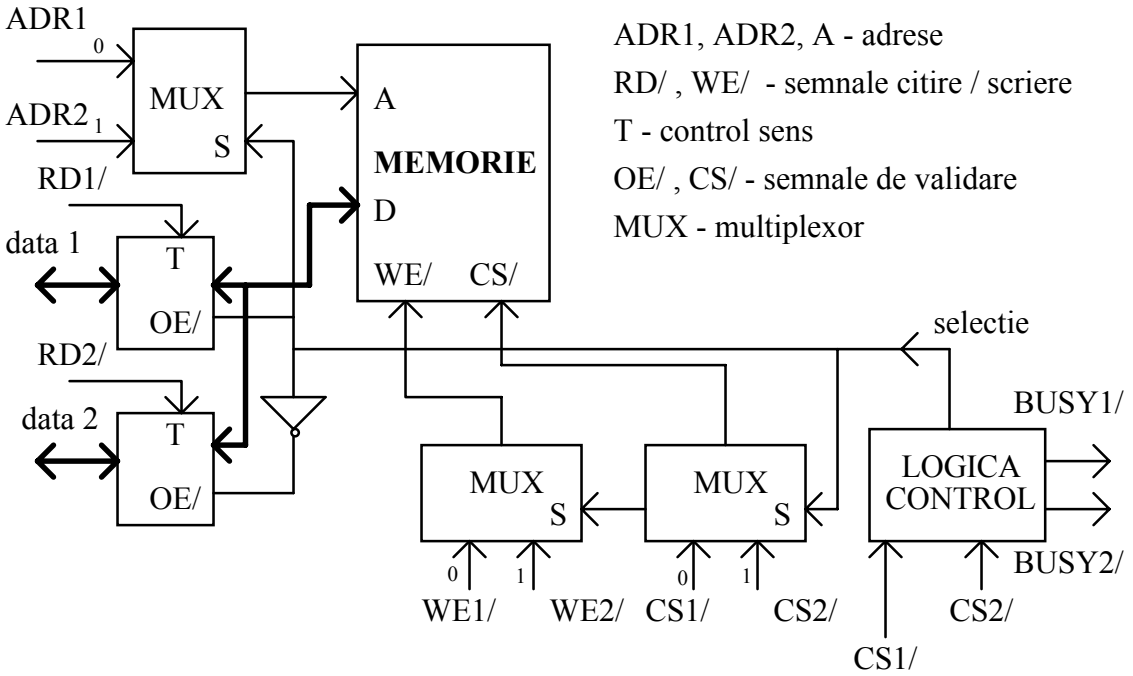


Figura 3. Interconectarea procesoarelor prin memorie comună (fără controler integrat)

Dezavantajul soluției din figura 3 este complexitatea hardware în raport cu soluția integrată din figura 2.

1.1.2. Sistem multiprocesor cu împărțirea bus-urilor (*bus shareing*)

În figura 4.. se prezintă o soluție de interconectare ce utilizează principiul cererii - achitării de magistrală (*bus request - bus grant*).

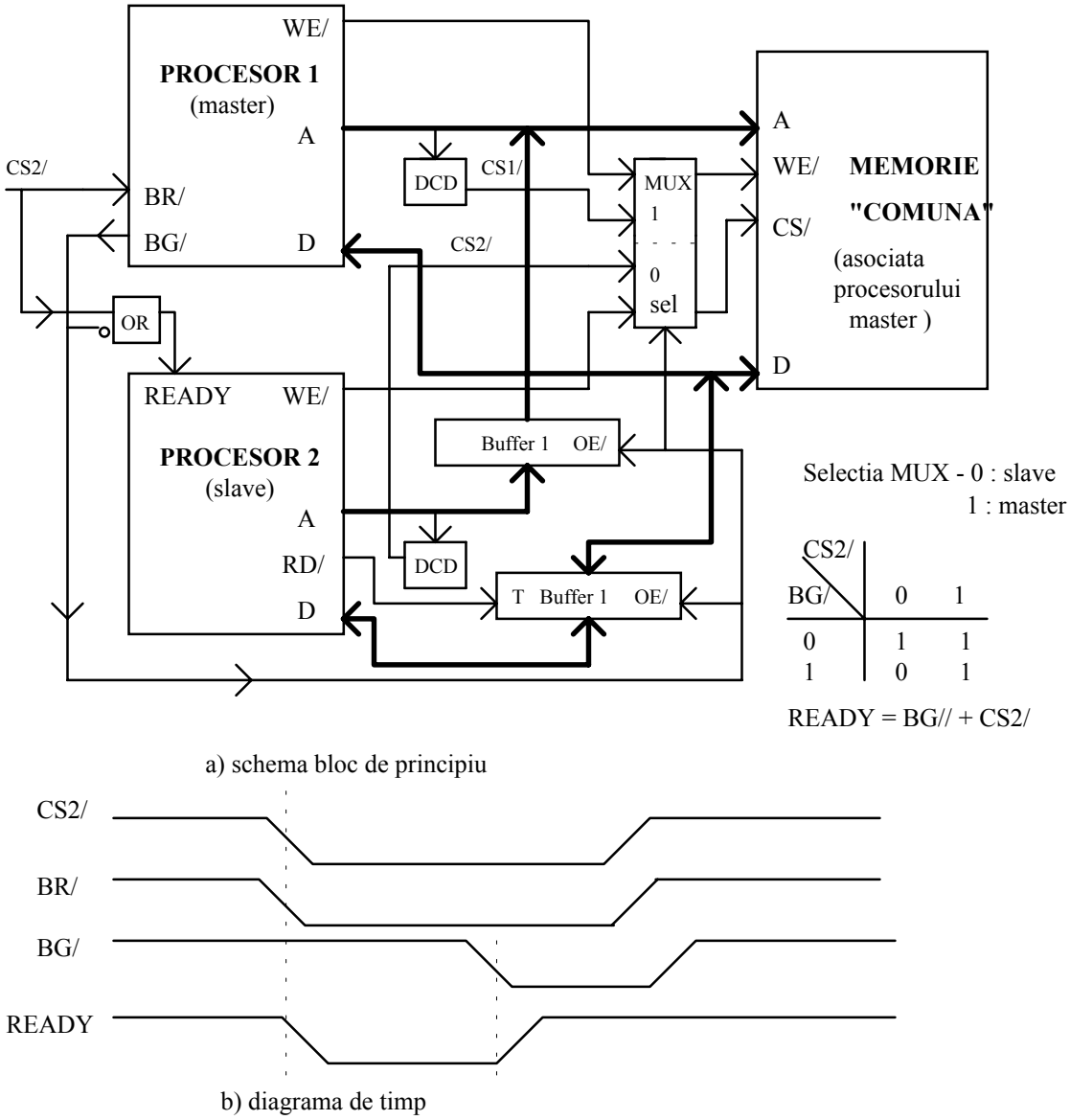
Comunicarea între procesoare se realizează tot printr-o memorie comună; cele 2 procesoare utilizează aceleaşi bus-uri pentru accesul la memoria comună.

Unul dintre procesoare va fi *master* pînă cînd primeşte o cerere de bus; acestă cerere va fi generată atunci cînd procesorul *slave* doreşte să acceseze memoria comună.

Ciclul de bus al procesorului *slave* este întîrziat (prin trecerea acestui procesor în starea WAIT) pînă la activarea semnalului ce semnifică acceptarea cererii de magistrală.

Soluția din figura 4. este funcțională numai dacă procesorul *slave* posedă o intrare ce permite trecerea sa în starea WAIT (de exemplu READY la I80x86).

În caz contrar se pot insera stări de WAIT pentru ciclurile de acces la memoria comună pentru a aştepta eliberarea magistralelor de către *master* (ca în figura 5.) .



a) schema bloc de principiu

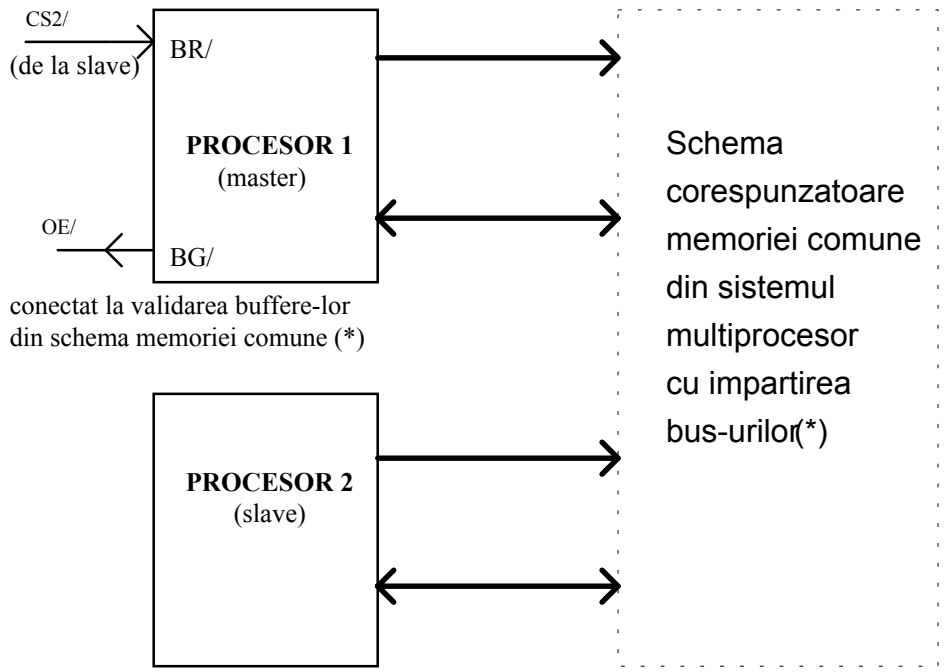
b) diagrama de timp

Figura 4. Sistem multiprocesor cu împărțirea busurilor

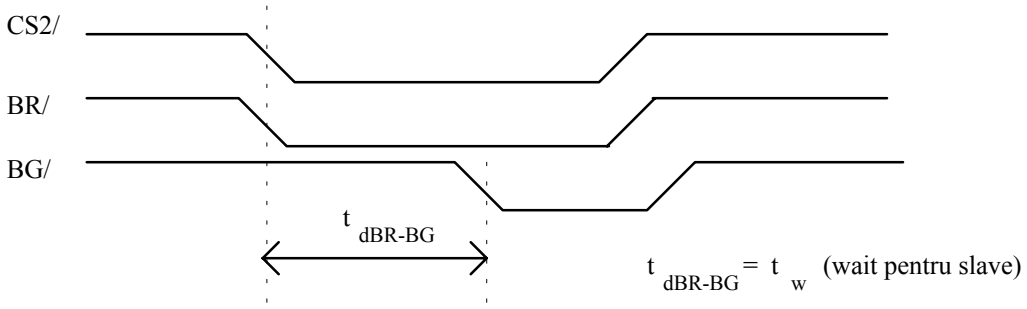
2. Descrierea unui sistem multiprocesor - I80x86 - ADSP210x

Pentru exemplificarea sistemului multiprocesor cu împărțirea busurilor se consideră un sistem cu procesor specializat (procesor DSP - Digital Signal Processing - ADSP210x) interconectat cu procesorul I80x86 din structura unui calculator PC.

Conectarea între aceste două subsisteme se realizează prin memorie comună (situată pe placa cu procesor DSP). Memoria comună este memoria procesorului DSP (considerat master) , dar poate fi accesată de către PC prin mecanismul *bus reguest - bus grant* , descris în figurile 4 sau 5.



a) schema bloc de principiu



b) diagrama de timp

Figura 5. Varianta a unui sistem multiprocesor cu împărțirea busurilor

Zona de memorie comună (de interes în acest caz) este "văzută" în felul următor de către procesorul DSP sau de către procesorul I80x86 (PC) :

- pentru DSP : memoria comună este pe 16 biți între adresele 0 - 1FFF
- pentru PC : memoria comună pe 8 biți atfel :

D000:6000 - D000:7FFF - octetul cel mai puțin semnificativ
D000:8000 - D000:9FFF - octetul cel mai semnificativ

În mod suplimentar sistemul cu procesor DSP poate genera o întrerupere către PC (utilă pentru sincronizare).

Se consideră că sistemul multiprocesor astfel constituit trebuie să îndeplinească următoarele funcțiuni :

PC - stabileşte doi operanzi şi operația ce se efectuează asupra acestora (în memoria comună)
- citeşte rezultatul operației (din memoria comună) şi îl afişează pe display

DSP - citeşte (din memoria comună operanzii şi tipul operației
- efectuează operația
- stochează (tot în memoria comună) rezultatul

Se utilizează următoarea structură de date (de finită în memoria comună) (figura 6):

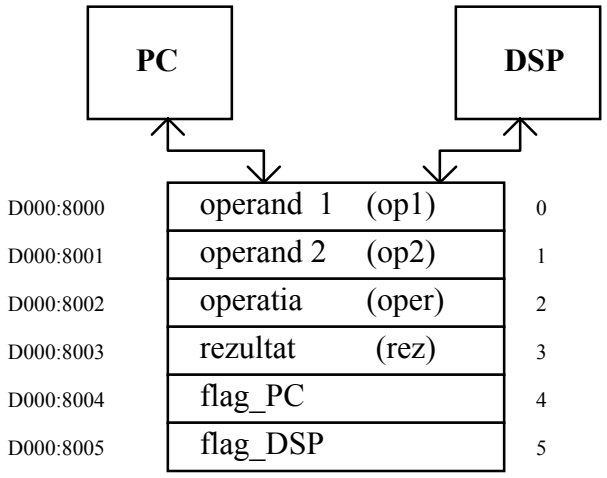


Figura 6. Structura de date asociată sistemului multiprocesor

Se consideră operanzii pe 8 biți (octeții cei mai puțin semnificativi pentru operanzii DSP sînt egali cu zero).

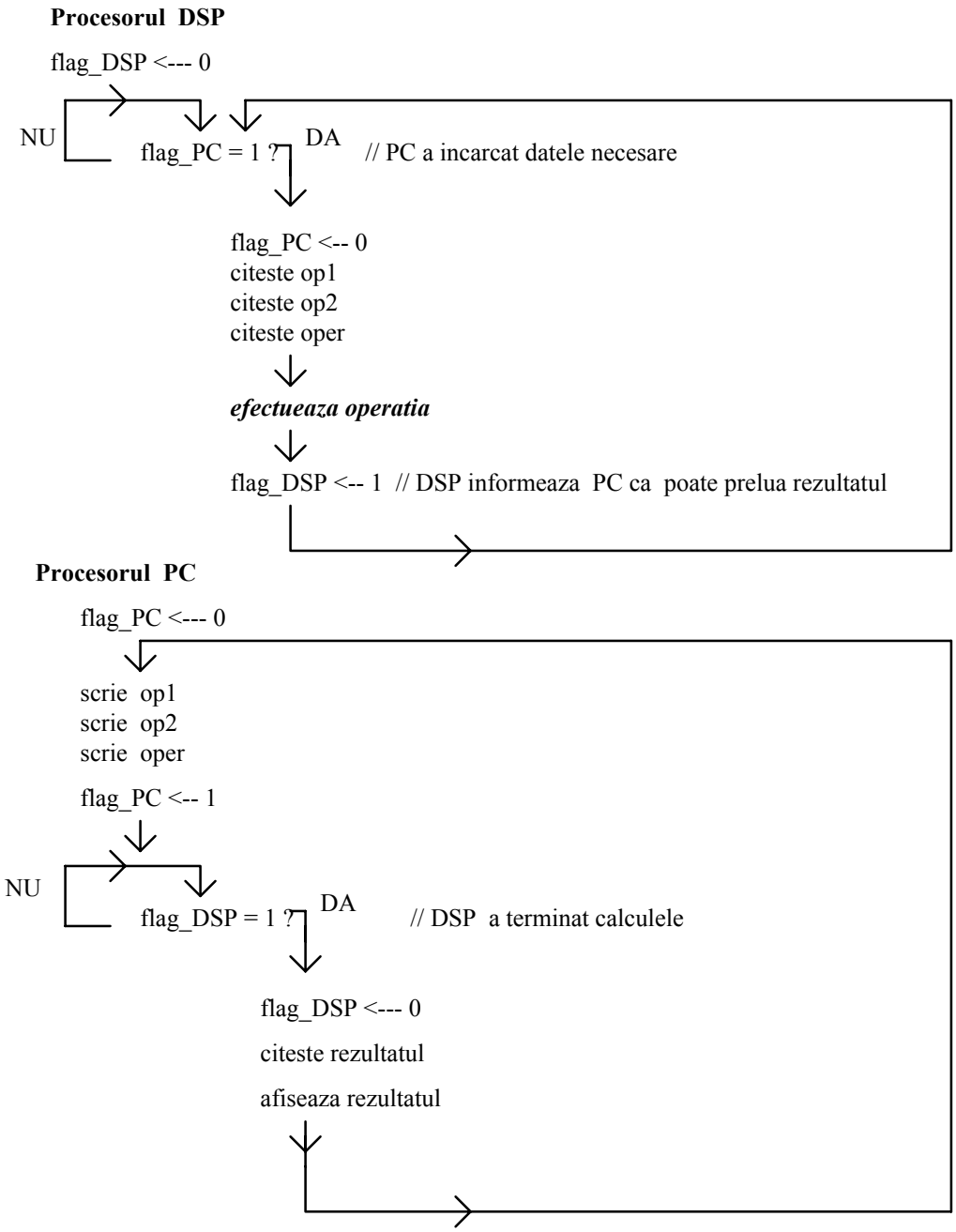
Ultimele două cîmpuri ale structurii de date vor fi utilizate pentru sincronizarea activităților celor două procesoare.

Cererea de bus (BR/) se efectuează prin scrierea unui 0 logic în portul de adresă 0x301.Anularea cererii de bus se efectuează prin scrierea în acelaşi port a unui 1 logic.

Se consideră că cerearea de bus este acceptată imediat (diferențele de viteză de execuție între DSP şi PC sînt mari , semnalul BG/ - acceptarea cererii de bus - se va activa înainte ca I80x86 să efectueze ciclul de bus cerut) . (Conectarea este realizată după ideea din figura 5.)

3. Sincronizarea activităţilor procesoarelor din sistemul multiprocesor I80x86 - ADSP210x

Organigramele de funcționare pentru cele două procesoare din sistemul multiprocesor sînt ilustrate în figura 7.



4. Programe aplicative pentru sistemul multiprocesor

// ASC3.C - program pentru I80x86 - PC

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>


char cmd[6]={0,0,0,-1,0,0};
/*
cmd[0] <--- operand 1
cmd[1] <--- operand 1
cmd[2] <--- operatia astfel :
                                        0 - operand1 + operand 2 = rezultat
                                        1 - operand1 - operand 2 = rezultat
                                        2 - operand1 * operand 2 = rezultat
cmd[3] <--- rezultat
cmd[4] <--- flag_PC
cmd[5] <--- flag_DSP
*/

char unsigned far *p1;
char unsigned far *p2;

void wbuff(char c,char p)
// scrie caracterul c in buffer-ul comun
// la adresa data de indexul p
{
outport(0x301,0x0100); // BR/ activat

p1=(char unsigned far*)MK_FP(0xD000,0x6000+p); // octetul LSB
p2=(char unsigned far*)MK_FP(0xD000,0x8000+p); // octetul MSB

 *p1=c;
 *p2=0;  // octetul MSB este 0

outport(0x301,0x0101); // BR/ dezactivat
}

char rbuff(char p)
// intoarce caracterul citit din buffer-ul comun
// la adresa data de indexul p
{
char c1,c2;
outport(0x301,0x0100); // BR/ activat

p1=(char unsigned far*)MK_FP(0xD000,0x6000+p); // octetul LSB
p2=(char unsigned far*)MK_FP(0xD000,0x8000+p); // octetul MSB

 c1=*p1;
 c2=*p2;
```

```
outport(0x301,0x0101); // BR/ dezactivat

return(256*c2 + c1);
}

void main(void)
{
char key;

clrscr();

while(key!='x')

{
// scriere operanzi

printf("Operandul 1 =  " );
scanf("%d",&cmd[0]);
wbuff(cmd[0],0);

printf("Operandul 2 =  " );
scanf("%d",&cmd[1]);
wbuff(cmd[1],1);

// scriere operatie

printf("Operatia (0 - adunare, 1 - scadere, 2 - inmultire) =  " );
scanf("%d",&cmd[2]);
wbuff(cmd[2],2);

// flag_PC=1

cmd[4]=1;
wbuff(1,4);     // scriere  in bufferul comun

//===== Sectiunea A ======
// intirziere pentru ca DSP sa poata efectua prelucrarea
//fara a fi oprit de o cerere de bus

//delay(1);
//==== Sectiunea A ======


//===== Sectiunea B ======

// test flag_DSP=1 si flag_PC=0
// DSP a preluat datele si a efectuat operatia
///*
while((cmd[5]==0)||(cmd[4]==1))
{
cmd[5]=rbuff(5);
cmd[4]=rbuff(4);
}
//*/
```

//===== Sectiunea B ======

```
// pozitioneaza flag_DSP=0

cmd[5]=0;
wbuff(0,5);     // scriere  in bufferul comun

// citeste rezultatul

cmd[3]=rbuff(3);

printf("\nRezultatul este =  %d\n\n",cmd[3]);
printf("x - Exit\tEnter - Continue\n\n");
key=getch();


}

}
```

// RK.C - program pentru procesorul DSP

```
/*
cmd[0] <-- operand 1
cmd[1] <-- operand 2
cmd[2] <-- operatia
cmd[3] <-- rezultatul
cmd[4] <-- flag_PC
cmd[5] <-- flag_DSP
*/

/* functia de adunare */

int add(int a,int b)
{
return(a+b);
}

/* functia de scadere */

int sub(int a,int b)
{
return(a-b);
}

/* functia de inmultire */

int mpy(int a,int b)
{
return(a*b);
}

int op1; /* operand 1 */
int op2; /* operand 2 */
```

```
int oper; /* operatia */
int rez; /* rezultatul */
int flag_PC; /* flag pentru PC */
int flag_DSP; /*flag pentru DSP */


void main(void)
{
asm(".var/dm/abs=0          cmd[6];");
asm(".init cmd[0] : 0,0,0,-1,0,0;");

while (1)
{
asm("     ax0=dm(cmd+4);\
          dm(flag_PC_)=ax0;"); /* flag_PC <-- cmd[4] */


if (flag_PC==1)
{

flag_PC=0;
asm("     ax0=dm(flag_PC_);\
          dm(cmd+4)=ax0;"); /* cmd[4] <-- flag_PC */

asm("     ax0=dm(cmd);\
          dm(op1_)=ax0;"); /* op1 <-- cmd[0] */

asm("     ax0=dm(cmd+1);\
          dm(op2_)=ax0;"); /* op2 <-- cmd[1] */

asm("     ax0=dm(cmd+2);\
          dm(oper_)=ax0;"); /* oper <-- cmd[2] */


switch (oper)
{
case 0: {rez=add(op1,op2);break;}
case 1: {rez=sub(op1,op2);break;}
case 2: {rez=mpy(op1,op2);break;}
default : rez=0;
}

asm("     ax0=dm(rez_);\
          dm(cmd+3)=ax0;"); /* cmd[3] <-- rez */
flag_DSP=1;
asm("     ax0=dm(flag_DSP_);\
          dm(cmd+5)=ax0;"); /* cmd[5] <-- flag_DSP */

}
}
}
```

5. Desfăsurarea lucrării

a) Să se studieze programele ASC3.C şi RK.C (după organigrama din figura 7).
b) Se va încărca programul RK.EXE în memoria procesorului DSP cu comenzile (de la prompter-ul DOS) :

LOAD RK.EXE

apoi se vor specifica parametrii :

Segment address 0,1,2,3,4 : **3**

I/O address 0,1,2,3 : **0**

După încărcare procesorul DSP execută automat programul.

Se lansează , din mediul integrat Borland C , programul ASC3.C .

Se va urmări execuția programelor.

c) Să se execute programul ASC3.C cu secțiunea A sau cu secțiunea B (cu întîrziere sau cu sincronizare prin falg-uri). Explicați funcționarea în cele două cazuri.

Temă

Să se modifice organigrama din figura 7 (pentru PC) astfel încît sincronizarea să se efectueze utilizînd întreruperi de la DSP (generate cînd DSP a terminat prelucrările).