Arhitectura Calculatorului Didactic :

- **Arhitectura bazata pe Registre Generale**
  - Calculatorul dispune de 8 Registre Generale

- **Arhitectura pe 16 biti :**
  - Registrele generale au dimensiunea de 16 biti
  - Unitatea aritmetico-logica prelucreaza operanzi pe 16 biti
  - Magistrala procesorului este pe 16 biti
  - Spatiul de adresare este de 2^{16} Cuvinte, adica 64 Kcuv

- **UAL pentru intregi reprezentati in cod complementar**
  - Procesorul dispune de o singura unitate aritmetico-logica ce opereaza cu intregi cu semn pe 16 biti

- **Moduri de adresare complexe**
  - Setul de instructiuni si modurile de adresare deriva din arhitectura x86
  - Modurile de adresare sunt specifice procesoarelor CISC
    - Permite lucrul cu operanzi din memorie, fara incarcare prealabila in registrele generale
    - Modurile de adresare sunt numeroase si foarte flexibile

OBSERVATII

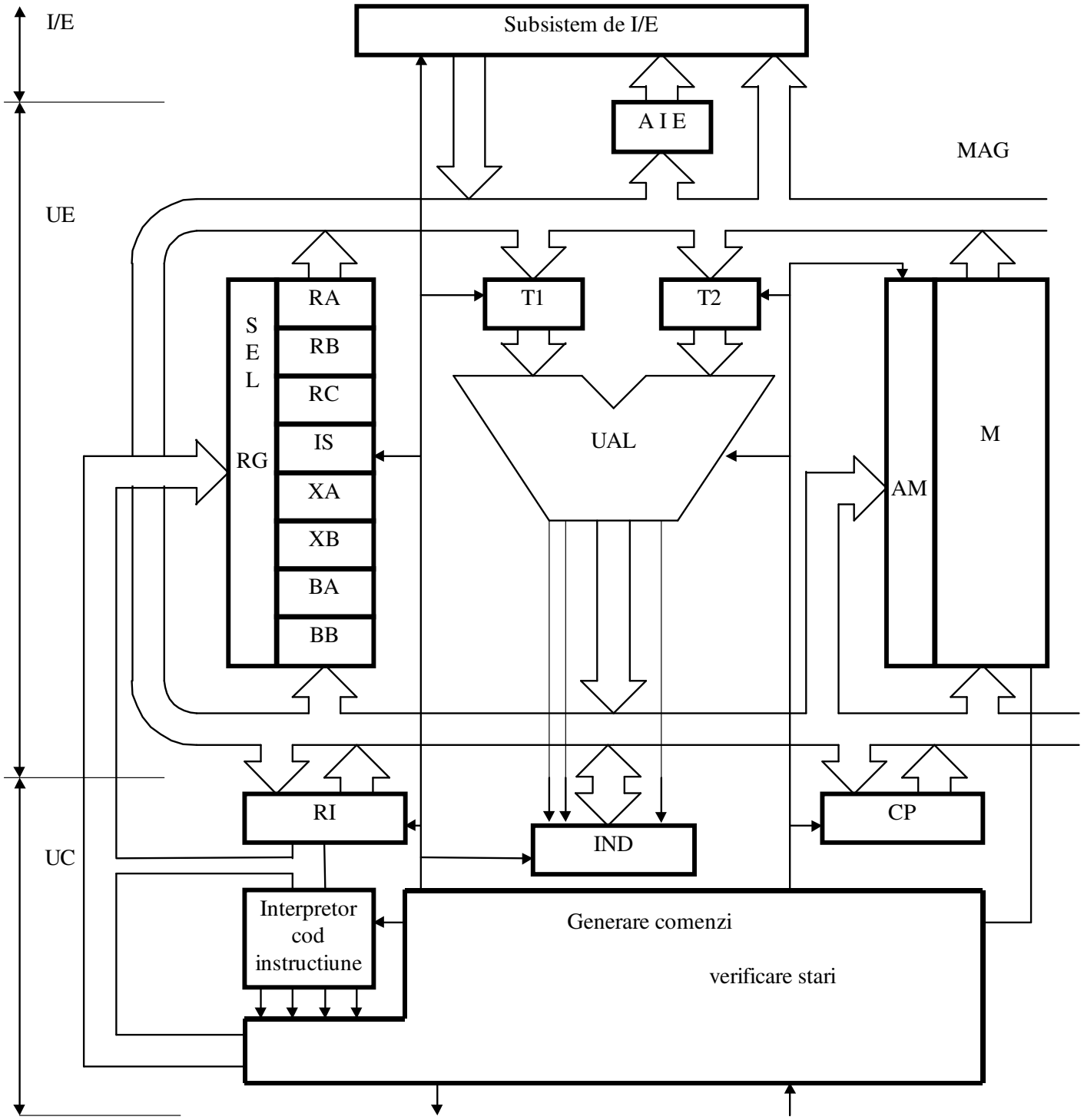
1. Procesorul didactic este consistent din punctul de vedere al dimensiunii : 16 biti. Procesoarele reale, in marea lor majoritate nu respecta aceasta regula. Spre exemplu, Pentium 4 cu arhitectura pe 32 de biti (IA-32) include registre pe 128 de biti, si dispune de o magistrala de adrese pe 36 de biti (spatiul total de adresare este de 64 TB). In consecinta parerile sunt impartite in legatura cu care ar fi o definitie corecta pentru dimensiunea procesorului. Cea mai frecventa definitie spune ca dimensiunea unui procesor este data de dimensiunea registrelor si a unitatilor aritmetico-logice.

2. Spatiul de adresare pentru un procesor pe N biti este de 2^N locatii de memorie. In functie de organizarea acesteia insa, aceste locatii pot fi octeti sau cuvinte de mai multi octeti. Memoria calculatorului didactic este organizata ca un spatiu contiguu de 64 Kcuvinte de 16 biti fiecare. Asadar spatiul total de adresare pentru calculatorul didactic este de 128 Kbytes. Memoria din calculatoarele voastre insa este adresabila la nivel de octet. Daca calculatorul didactic ar fi fost echipat cu o astfel de memorie, spatiul de adresare ar fi fost de 64 KB, deoarece la fiecare locatie se poate stoca fix un byte.

3. Prin faptul ca procesorul permite lucrul cu operanzi direct din memorie se intelege ca ei nu trebuie adusi in prealabil de catre programator intr-un registru general. Cu toate acestea nu se poate lucra cu ambii operanzi direct din memorie. Acest tip de procesare specific arhitecturilor CISC poarta numele de procesare Registru-Memorie. Spre deosebire de aceasta, arhitecturile RISC tipice necesita incarcarea prealabila a operanzilor in registrele generale. De aceea, aceste procesoare se mai numesc Registru-Registru sau Load/Store. Arhitecturile Memorie-Memorie sunt foarte rare, datorita complexitatii hardware-ului si performantelor scazute. Iata un exemplu ce aduna doua numere aflate in memorie la adresele 10 si 20:

R-R	R-M	M-M

MOV RA, [10]
MOV RB,[20]
ADD RA, RB
MOV [10], RA
	MOV RB, [20]
	ADD [10],RB
		ADD [10],[20]



Schema bloc a calculatorului didactic

Resursele calculatorului didactic

Magistrala MAG

Interconectarea tuturor resurselor se realizează prin intermediul unei magistrale, MAG, care constituie suportul fizic de comunicație între aceste resurse. Dimensiunea magistralei este de 16 linii. Deoarece magistrala este în totalitate pasivă (este un set de conductori), un singur cuvânt de informație poate exista pe magistrală la un moment dat.

Registrele Generale RG

Deoarece timpul de acces la memoria M este relativ mare (de ordinul zecilor de nanosecunde) procesorul dispune de 8 registre generale de câte 16 biți fiecare ce lucreaza la frecventa de ceas a procesorului. In tabelul de mai jos sunt sumarizate functiile acestora :

RA, RB, RC	▪ La dispozitia programatorului pentru stocarea operanzilor ▪ RA este folosit in lucrul cu porturile
IS	▪ Indicatorul de stiva
XA, XB	▪ Se pot folosi pentru stocarea operanzilor ▪ Sunt folositi pentru adresarea memoriei ca si registre index
BA, BB	▪ Se pot folosi pentru stocarea operanzilor ▪ Sunt folositi pentru adresarea memoriei ca si registre de baza

Unitatea aritmetică logică UAL

Unitatea aritmetică logică (UAL) realizează operațiile aritmetice şi logice ale calculatorului didactic. Ea este utilizată pentru prelucrarea datelor şi pentru calculul adresei efective. Unitatea aritmetică logică prelucrează operanzi pe 16 biți reprezentați în cod complementar. Caracteristicile rezultatului (zero, par, transport si depasire) sunt depuse într-un registru de indicatori IND, in urma executiei oricarei instructiuni aritmetice-logice.

Memoria M

Memoria este utilizată pentru a păstra informații reprezentând date sau instrucțiuni. Memoria M este o matrice de elemente de memorare organizată într-un spațiu de adresare unic de 65536 cuvinte a câte 16 biți fiecare. Astfel capacitatea memoriei este de 64Kcuv x 2 octeti = 128 KB.

Registrul AM.

Registrul de adresare a memoriei, AM, păstrează adresa celulei de memorie la care se face acces la un moment dat. Cand se doreste realizarea unei operatii de citire din memorie, adresa solicitata va fi depusa in acest registru, iar unitatea de comanda va lansa comanda « Memory Read ». Dupa un anumit timp, memoria va furniza pe magistrala cuvantul de la adresa solicitata. Analog, cand se doreste sa se scrie la o anumita adresa din memorie un cuvant, aceasta este depusa in registrul AM, datele de scris sunt activate pe magistrala iar unitatea de comanda va lansa comanda « Memory Write ».

Registrul Contor Program CP

Registrul contor program CP este utilizat pentru păstrarea adresei instrucțiunii ce urmează să se execute după terminarea execuției instrucțiunii curente. Registrul CP va fi inițializat cu o valoare dată la pornirea sau resetarea sistemului. Dupa incarcarea fiecarei instructiuni, el se va incrementa pentru a marca avansul la urmatoarea instructiune. In cazul in care instructiunea executata este una de salt, adresa de salt va fi incarcata in CP in urma executiei instructiunii.

Registrul de instructiuni RI

Registrul de instrucțiuni RI păstrează instrucțiunea în curs de execuție. Conținutul său este folosit de unitatea de comandă in vederea generararii semnalelor de comanda pentru toate resursele.

Indicatorii de condiții IND

Registrul de indicatori constituie o grupare a unor flag-uri provenite din rezultatele instructiunilor de tip aritmetico-logic. Registrul IND permite unei instructiuni sa foloseasca informatii rezultate in urma executiei unei instructiuni anterioare. Spre exemplu, daca se doreste efectuarea unei sume cu operanzi pe 32 de biti, din moment ce dimensiunea procesorului este 16 biti, este nevoie sa se prelucreze pe rand octetii inferiori apoi octetii superiori. Cei doi operanzi vor ocupa doua adrese consecutive de memorie, fie ele 0xA16, 0xA17 pentru primul, respectiv 0xA18, 0xA19 pentru cel de-al doilea. Suma va fi depusa la adresele 0xA20,0xA21. Iata programul :

```
MOV RA, [0xA16]
MOV RB, [0xA18]
ADD RA, RB
MOV [0xA20],RA
MOV RA, [0xA17]
MOV RB, [0xA19]
ADC RA, RB
MOV [0xA21],RA
```

Dupa cum se poate observa, cea de-a doua operatie aritmetica este ADC, adica adunare cu transport (carry). Semnificatia este urmatoarea : daca de la operatia anterioara de adunare a aparut transport, acest transport va trebui propagat in octetii superiori. Astfel, operatia ADD, in cazul in care a aparut transport, seteaza bitul T din registrul IND. Operatia ADC realizeaza suma intre operanzi dar include in calcul si acest bit de transport.

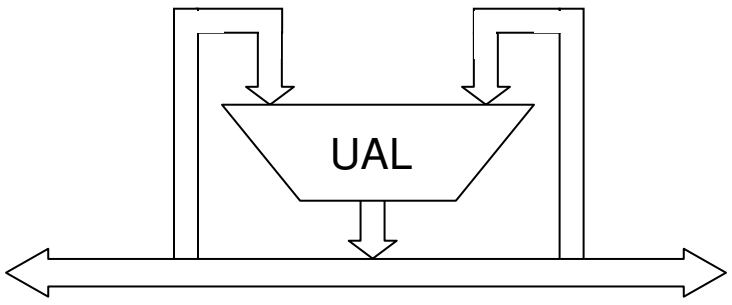
Un alt exemplu este saltul conditionat. O instructiune de tip if (a=2) se va implementa astfel :

```
CMP RA,2
JNE eticheta
```

Prima instructiune are ca efect scaderea continutului registrului RA cu 2. Rezultatul acestei operatii nu se stocheaza, insa ca orice operatie aritmetico-logica va afecta registrul de indicatori. Astfel, daca rezultatul a fost zero (continutul lui RA a fost 2), se va seta flagul Z (zero) din registrul IND. Urmatoarea instructiune, jump if not equal va testa acest flag si in functie de valoarea sa va realiza sau nu saltul.

Registrele temporare T1,T2

Registrele temporare T1 şi T2 sunt utilizate pentru a păstra operanzii unei operații executate în unitatea aritmetică logică precum si rezultate intermediare la calcularea adresei efective. Ele nu sunt accesibile în mod explicit programatorului. Pentru a explica necesitatea existentei acestor registre, sa consideram ca ele nu ar fi existat. In acest caz, am fi obtinut urmatoarea schema :



In acest caz, ambii operanzi cat si rezultatul operatiei ar trebui sa se gaseasca simultan pe magistrala, ceea ce este imposibil.

Subsistemul de Intrari/Iesiri

Subsistemul de intrari iesiri permite procesorului comunicatia cu mediul extern prin intermediul dispozitivelor periferice. Subsistemul este format din interfete (spre exemplu interfata seriala, paralela, IDE, USB, etc.) capabile sa comunice cu dispozitivele periferice in conformitate cu un standard. Aceste interfete includ un set de registre (de date/stari/comenzi) pentru comunicatia cu procesorul. De exemplu, cand procesorul doreste sa transmita un cuvant de date prin interfata USB, el va depune in registrul de date asociat interfetei USB cuvantul respectiv, apoi va scrie in registrul de comenzi comanda de transmisie. Fiecare astfel de registru este identificat printr-o adresa unica in sistem si poarta numele de ***port de intrare/iesire***. Asadar, totalitatea registrelor asociate interfetelor din sistem (porturilor) este echivalenta cu o memorie in care fiecare adresa este asociata unei interfete.

Registrul de adrese de intrare/iesire AIE

Intrucat subsistemul de intrari/iesiri apare procesorului ca o memorie in care fiecare locatie reprezinta un port asociat unei interfete, ca si in cazul memoriei este nevoie de un registru de adrese. Acesta va fi folosit pentru a stoca adresa portului cu care se doreste sa se comunice. Spre exemplu, atunci cand procesorul vrea sa transmita ceva pe interfata paralela, el va depune in registrul AIE valoarea 0x378 (ce identifica portul asociat interfetei paralele), va activa pe magistrala cuvantul de date ce se doreste transmis iar unitatea de comanda va lansa semnalul « I/O Write » ce va determina incarcarea cuvantului in registrul interfetei paralele.

Unitatea de comanda

Toate resursele prezentate pana in acest punct formeaza unitatea de executie. Unitatea de comanda dirijeaza aceste resurse pentru a executa o instructiune sau o alta. Spre exemplu, in cazul instructiunii ADD RA, RB unitatea de comanda va genera urmatoarea secventa de semnale :

1. Semnal catre blocul de registre generale pentru a determina activarea continutului lui RA pe magistrala.
2. Semnal catre T1 pentru a incarca valoarea aflata in acest moment pe magistrala.
3. Semnal catre blocul de registre generale pentru a determina activarea continutului lui RB pe magistrala.
4. Semnal catre T2 pentru a incarca valoarea aflata in acest moment pe magistrala.
5. Semnal catre UAL ce indica operatia de adunare
6. Semnal catre IND pentru a seta flag-urile Z, S, D, T, P
7. Semnal catre blocul de registre generale pentru a incarca in RA valoarea de pe magistrala

Cu alte cuvinte, unitatea de comanda este cea care controleaza functionarea procesorului. Ea dirijeaza intreg procesul de citire-interpretare-executie a instructiunilor. Descrierea AHPL de la curs este in realitate descrierea functionala a acestei unitati de comanda. In momentul in care aceasta descriere este gata, ce mai ramane de facut in proiectarea unui procesor real este de a stabili ce semnale de comanda trebuiesc generate la fiecare pas.

Modurile de adresare la calculatorul didactic

Modurile de adresare reprezintă modalitatea prin care se poate specifica adresa efectivă a operanzilor. Instrucțiunile calculatorului didactic pot prelucra maxim doi operanzi ce se pot gasi:

-ambii în registrele generale RG ;
-unul în registrele generale RG şi altul în memorie ;
-unul în registrele generale RG şi altul în cadrul instrucțiunii respective (operand imediat) ;
-unul în memorie şi altul imediat.

Modurile de adresare permise de calculatorul didactic sunt tipice arhitecturilor CISC, fiind derivate din arhitectura standard x86. In total, procesorul permite 11 moduri de adresare ceea ce ii confera o foarte buna flexibilitate in programare. Pentru o mai usoara intelegere, le vom structura astfel:

Operandul nu se gaseste in memorie	Adresare directa la registru Adresare imediata
Operandul e specificat doar prin deplasament	Adresare directa Adresare indirecta
Operandul e specificat doar prin registre	Adresare indirecta prin registru Adresare indirecta prin suma de registre Adresare indirecta prin suma de registre cu autoincrementare Adresare indirecta prin suma de registre cu autodecrementare
Operandul e specificat prin registre si deplasament	Adresare Bazata Adresare Indexata Adresare Bazata Indexata

In continuare vor fi prezentate si discutate aceste moduri de adresare.

Operandul nu se gaseste in memorie

1. **Adresare directă la registru**

Operandul se găseşte in RG.

Exemplu: MOV RA, RB
Instructiunea are ca efect incarcarea in registrul RA, a valorii din registrul RB

2. **Adresare imediată**

Operandul este specificat in instructiune

Exemplu: MOV RA,7
Instructiunea va avea ca efect incarcarea valorii 7 in registrul RA. ***7*** poarta numele de operand imediat.

Operandul e specificat doar prin deplasament

3. **Adresare directă**

Adresa efectivă este specificata in instructiune

Exemplu: MOV RA, [12]

Instructiunea va incarca valorea aflata in memorie la adresa 12 in registrul RA. ***12*** poarta numele de deplasament

4. **Adresare indirectă**

Adresa efectivă se citeşte din memorie, din locația a cărei adresă este specificata in instrucțiune.

Exemplu: MOV RA, [[12]]

Instructiunea are ca efect incarcarea valorii aflata la adresa ce se gaseste in memorie la adresa 12. Acest mod de adresare seamana foarte bine cu pointerii din C. La adresa 12 se gaseste pointer-ul catre operand.

Operandul e specificat doar prin registre

5. **Adresare indirectă prin registru**

Adresa efectivă se găseşte în unul din registrele XA, XB,BA, BB

Exemplu: MOV RA,[BA]

Instructiunea va incarca in RA valoarea aflata in memorie la adresa continuta in BA.

6. **Adresare indirectă prin sumă de registre**

Adresa efectivă se obține ca sumă a conținutului unui registru de bază cu conținutul unui registru index.

Exemplu: MOV RA,[BA] [XA] sau MOV RA,[BA + XA]

Instructiunile incarca in RA valoarea aflata in memorie la adresa obtinuta prin adunarea continutului registrelor BA si XA. BA, BB poarta numele de registre de baza, iar XA, XB poarta numele de registre index. Intotdeauna suma se va face intre un registru baza si unul index.

7. **Adresare indirectă prin sumă de registre cu autoincrementare**

Față de modul precedent de adresare apare deosebirea că registrele index se incrementează după generarea adresei efective. Incrementarea registrului index (XA sau XB) are deci loc după participarea la calculul adresei efective.

Exemplu: MOV RA,[BA][XA+] sau MOV RA, [BA+XA+]

Aceste instructiuni sunt echivalente cu :

```
MOV RA, [BA][XA]
INC XA
```

8. **Adresare indirectă prin sumă de registre cu autodecrementare**

Adresa efectiva se obtine prin suma unui registru de baza cu continutul registrului XA. Inaintea generarii adresei, registrul XA este decrementat.

Exemplu: MOV RA,[BA][XA-] sau MOV RA, [BA+XA-]

Aceste instructiuni sunt echivalente cu :

```
DEC XA
MOV RA, [BA][XA]
```

Observatii: Motivul pentru care XB nu poate fi utilizat in acest mod de adresare nu este unul logic ci unul tehnic: bitii din codul de instructiune nu au fost suficienti pentru a se putea codifica si acest mod.

Operandul e specificat prin registre si deplasament

9. Adresare bazată

Adresa efectivă se obține prin adunarea unui registru de baza cu un deplasament.

Exemplu: MOV RA,[BA]+7 sau MOV RA,[BA+7]

Instructiunea incarca in registrul RA valoarea aflata in meorie la adresa rezultata in urma adunarii continutului registrului de baza cu deplasamentul (7).

10. Adresare indexată

Adresa efectivă se obține prin adunarea unui registru index cu un deplasament.

Exemplu: MOV RA,[XA]+7 sau MOV RA,[XA+7]

Instructiunea incarca in registrul RA valoarea aflata in meorie la adresa rezultata in urma adunarii continutului registrului de index cu deplasamentul (7).

11. Adresare bazată indexată

Adresa efectivă se obține prin adunarea unui registru de baza cu un registru index si cu un deplasament.

Exemplu: MOV RA,[BA][XA]+7 sau MOV RA, [BA+XA+7]

Instructiunea incarca in registrul RA valoarea aflata in meorie la adresa rezultata in urma adunarii continutului registrului de baza (BA) cu registrul index (XA) si cu deplasamentul (7).

Recapitulare Moduri de adresare

Tip adresare	Mod Adresare	Exemplu			Detalii	Observatii
		Instructiune	Adresa efectiva	Efecte laterale		
Operandul nu se gaseste in memorie	Directa la registru Imediata	MOV RA, RB MOV RA, 7	- -	- -	Operandul se afla in orice RG Operandul e o constanta data in instr	Poate fi folosit orice RG Constanta este pe 16 biti
Operandul e specificat doar prin deplasament	Directa Indirecta	MOV RA, [7] MOV RA,[[7]]	7 [7]	- -	Adresa operandului e data in instructiune Adresa adresei operandului e data in instr	Seamana foarte bine cu pointerii

Operandul e specificat doar prin registre	Indirecta prin registru
	Indirecta prin suma de registre
	Indirecta prin suma de registre cu autoincrementare
	Indirecta prin suma de registre cu autodecrementare
		MOV RA, [BA]
		MOV RA, [BA+XA]
		MOV RA, [BA+XA+]
		MOV RA, [BA+XA-]
			BA
			BA+XA
			BA+XA
			BA+XA-1
				-
				-
				XA++
				XA--
					Orice registru baza sau index
					Un reg baza + un reg index
					Un reg baza + un reg index
					Un reg baza + XA
						Adresa e continuta intr-un registru
						Adresa se calculeaza ca suma de reg
						Incrementarea se face dupa adresare
						Decrementarea se face inainte de adresare

Operandul e specificat prin registre si deplasament	Bazata
	Indexata
	Bazata Indexata
		MOV RA, [BA+7]
		MOV RA, [XA+7]
		MOV RA, [BA+XA+7]
			BA+7
			XA+7
			BA+XA+7
				-
				-
				-
					Un reg baza + Depls
					Un reg index + Depls
					Un reg baza + un reg index + Depls	

Glosar :

Operand Imediat – O constanta specificata in instructiune , ce va fi folosita ca operand
Deplasament – O valoare specificata in instructiune ce intra in calculul adresei operandului
Registru de baza – Registrele BA, BB
Registru Index – Registrele XA, XB

Exemple de utilizare a modurilor de adresare

Problema 1
Sa se calculeze suma elementelor unui vector cu 10 de elemente, fiecare avand 16 biti. Adresa de baza a vectorului (adresa la care acesta se gaseste in memorie) este 20 ;

Solutie :

Varianta 1 – Adresare indirecta prin registru

```
        MOV RA, 0       ;Registru acumulator pentru suma
        MOV BA, 20      ;Adresa de baza a vectorului
Bucla:
        ADD RA, [BA]    ;Se aduna elementul curent
        INC BA          ;Se incrementeaza BA pentru a avansa la elementul urmator
        CMP BA, 30      ;Se testeaza daca s-a atins adresa de final a vectorului
        JNE Bucla       ;Nu s-a atins adresa de final, repeta de la Bucla
```

Programul functioneaza astfel : Se depune adresa de baza in registrul BA care va fi folosit pentru adresare. Dupa fiecare acces BA este incrementat si se testeaza daca s-a atins finalul vectorului.

Varianta 2 – Adresare indirecta prin suma de registre

```
        MOV RA, 0       ;Registru acumulator pentru suma
        MOV BA, 20      ;Adresa de baza a vectorului
        MOV XA, 0       ;Indexul in vector
Bucla:
        ADD RA, [BA+XA] ;Se aduna elementul curent
        INC XA          ;Se incrementeaza indexul
        CMP XA, 10      ;Se testeaza daca indexul a ajuns la val maxima
        JNE Bucla       ;Nu s-a atins finalul vectorului, repeta de la Bucla
```

Aceasta a doua varianta este ceva mai clara, intrucat XA indica exact indexul elementului din vector. Astfel ca XA porneste de la 0 si se opreste la 10, exact ca un contor de la o bucla C.

Varianta 3 - indirecta prin suma de registre cu autoincrementare

```
        MOV RA, 0       ;Registru acumulator pentru suma
        MOV BA, 20      ;Adresa de baza a vectorului
        MOV XA, 0       ;Indexul in vector
Bucla:
        ADD RA, [BA+XA+];Se aduna elementul curent cu autoincrementarea index-ului
        CMP XA, [10]    ;Se testeaza daca indexul a ajuns la val maxima
        JNE Bucla       ;Nu s-a atins finalul vectorului, repeta de la Bucla
```

Aceasta varianta este cea mai eficienta. Se elimina o instructiune din bucla, ceea ce va spori mult performanta algoritmului. E drept, adunarea lui XA consuma in contiunare ceva timp, intrucat ea totusi se executa in cadrul instructiunii ADD. Cu toate acestea, ceea ce se castiga fata de cazul precedent sunt ciclurile de citire-interpretare necesare unei instructiuni suplimentare.

Problema 2

Sa se calculeze sumele valorilor de pe linii pentru o matrice 20x10 avand adresa de baza 20. Sumele vor fi depuse intr-un vector avand adresa de baza 300.

Matricea este stocata liniarizat pe linii. Acest lucru inseamna ca la adresa 20 se afla primul element de pe prima linie, la adresa 21 al doilea element de pe prima linie etc. La adresa 30 se afla primul element de pe a doua linie, la adresa 31 al doilea element de pe a doua linie, etc. Cu alte cuvinte matricea este transformata intr-un vector prin concatenarea liniilor.

Solutie :

```
        MOV RA, 0               ;Registrul acumulator pentru suma

        MOV BA, 20              ;Adresa de baza a liniei curente
        MOV XA, 0               ;Indexul pentru coloana curenta

        MOV BB,300              ;Adresa de baza a vectorului cu sume
        MOV XB,0                ;Indexul pentru elementul curent din vectorul cu sume

Bucla:
        ADD RA, [BA+XA+]        ;Se aduna elementul curent si se autoincrementeaza indexul
        CMP XA, 10              ;Se testeaza daca s-a ajuns la final de linie
        JNE Bucla               ;Daca nu s-a ajuns la final de linie repeta de la Bucla

        MOV [BB][XB+], RA       ;Copiaza in vectorul cu sume suma calculata si incrementeaza index
        MOV RA,0                ;Reseteaza acumulatorului

        ADD BA,10               ;Trecere la linia urmatoare
        MOV XA,0                ;Reseteaza indexul de coloana
        CMP BA, 20              ;Se testeaza daca s-a trecut de ultima linie
        JNE Bucla               ;Daca nu s-a ajuns la ultima linie, repeta de la Bucla
```

Solutia foloseste adresarea indirecta prin suma de registre cu autoincrementare atat pentru matrice cat si pentru vectorul de sume. Pentru matrice, BA indica adresa de baza a liniei curente, iar XA numarul coloanei curente. Astfel, adresa elementului curent este intotdeauna BA+XA. La fiecare pas se aduna elementul de la pozitia curenta la RA si se avanseaza pe linie. Cand se atinge finalul liniei curente, se depune suma in vectorul de suma, se avanseaza BA asa incat sa indice la adresa de baza a liniei urmatoare si se reseteaza XA. Daca BA a trecut de ultima linie, nu se continua calculul.

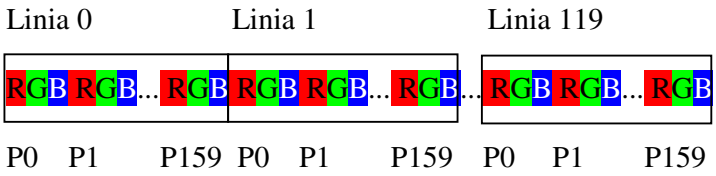
Pentru vectorul de sume, se folosesc registrele BB si XB. BB va indica mereu adresa de inceput a vectorului, iar XB va indica elementul curent. Dupa fiecare actualizare in vector, XB este incrementat pentru a indica elementul urmator.

Problema 3

Memoria video este o zona de memorie in care este stocata imaginea afisata la momentul curent pe ecran. Fiecarui pixel ii corespund trei locatii consecutive de memorie, una pentru fiecare canal de culoare : Rosu, Verde, respectiv Albastru (RGB). Memoria video este organizata ca o matrice liniarizata pe linii, ca la problema 2. Rezolutia ecranului este de 160x120, iar pentru fiecare canal de culoare sunt necesari 8 biti. Cu toate acestea, pentru eficienta fiecare canal este stocat pe 16 biti. Sa se transforme imaginea de pe ecran in alb-negru (grayscale). Se cunoaste adresa de baza a matricei : 10.

Solutie

Memoria video este organizata astfel



Prima locatie corespunde pixelului din stanga-sus a ecranului, iar ultima locatie corespunde pixelului stanga-jos. In total matricea liniarizata are 160x120 = 7200 de locatii x 3 canale de culoare.

Transformarea in alb-negru presupune asocierea fiecarei culori cu o nuanta de gri. O nuanta de gri are aceeasi valoare pe fiecare canal de culoare, adica R=G=B. Ceea ce avem noi de facut este sa calculam aceasta valoare pentru fiecare pixel. Se stie ca o valoare mai mare pe un canal de culoare, inseamna o culoare mai deschisa. Deci o varianta pentru a face transformarea in alb-negru ar fi sa calculam luminozitatea medie pentru pixelul curent, folosind o medie aritmetica. Apoi ceea ce ramane de facut este sa depunem aceasta valoare medie pe fiecare canal. Ceea ce trebuie sa facem noi deci este sa parcurgem vectorul, sa facem media valorilor pe fiecare canal de culoare si sa inlocuim valorile initiale de pe fiecare canal cu valoarea obtinuta. Iata un cod C pentru acest lucru :

```
for (i = 0 ; i < 7200*3 ; i += 3)
{
  M = a[i]+a[i+1]+a[i+2]/3 ;
  a[i] = M ;
  a[i+1] = M ;
  a[i+2] = M ;
}
```

In limbaj de asamblare pentru calculatorul didactic o varianta de cod ar fi urmatoarea :

```
        MOV BA, 10              ;Se initializeaza BA cu adresa de baza a memoriei video
        MOV XA, 0               ;Se initializeaza indexul catre pixelul curent la 0

Bucla:
        MOV RA, [BA][XA]        ;Se aduce valoarea pentru Rosu a pixelului curent in RA
        ADD RA, [BA][XA][1]     ;Se aduna RA cu valoarea pentru Verde
        ADD RA, [BA][XA][2]     ;Se aduna RA cu valoarea pentru Albastru
        DIV RA, 3               ;Se face media prin impartire la 3
        MOV [BA][XA+], RA       ;Se depune in canalul Rosu valoarea obtinuta
        MOV [BA][XA+], RA       ;Se depune in canalul Verde valoarea obtinuta
        MOV [BA][XA+],RA;       ;Se depune in canalul Albastru valoarea obtinuta
        CMP XA, 21600           ;Se testeaza daca s-a ajuns la finalul memoriei video
        JNE Bucla
```

Observatie : Am presupus implementata instructiunea DIV.