Laboratorul 6 - Arhitectura Calculatorului Didactic

Caracteristici ale calculatorului didactic

- Arhitectura bazată pe Registre Generale
  - Calculatorul dispune de 8 Registre Generale
- Arhitectura pe 16 biţi:
  - Registrele generale au dimensiunea de 16 biţi
  - Unitatea aritmetico-logică (UAL) prelucrează operanzi pe 16 biţi
  - Magistrala procesorului (MAG) este pe 16 biţi
  - Spaţiul de adresare este de 2^{16} Cuvinte, adică 64 Kcuvinte
- UAL pentru întregi reprezentaţi în cod complementar
  - Procesorul dispune de o singură unitate aritmetică-logică ce operează cu întregi cu semn pe 16 biţi
- Moduri de adresare complexe
  - Setul de instrucţiuni şi modurile de adresare derivă din arhitectura x86
  - Modurile de adresare sunt specifice procesoarelor CISC
    - Lucrul cu operanzi din memorie, fără încărcare prealabilă în registrele generale
    - Modurile de adresare sunt numeroase şi foarte flexibile

Arhitectura setului de instrucţiuni (ISA - Instruction Set Architecture) este folosită pentru a abstractiza funcţionarea internă a unui procesor. ISA defineşte “personalitatea” unui procesor: cum funcţionează procesorul d.p.d.v. al programatorului, ce fel de instrucţiuni execută, care este semantica acestora. ISA este cea mai importantă parte a design-ului unui procesor; alte aspecte cum sunt componentele de calcul și stocare, interacţiunea cu memoriile, pipeline-ul, fluxul de date în procesor putând fi schimbate de la o versiune la alta a procesorului. La ora actuală există două filozofii de design pentru un procesor: *Complex Instruction Set Computer (CISC)* şi *Reduced Instruction Set Computer (RISC)*. În afară de acestea există şi ISA-uri pentru procesoare specializate, cum sunt GPU-urile pentru plăci grafice si DSP-urile pentru procesare de semnal.

Principalele categorii de instrucțiuni sunt cele aritmetico-logice, de control secvențial, și respectiv de acces la memorie. Formatul instrucțiunilor RISC are o lungime fixă, cu lungimea unei instrucțiuni în general egală cu lungimea cuvantului de memorie; în cazul CISC, lungimea unei instrucțiuni variază în funcție de formatul instrucțiunii. RISC are un număr mic de moduri de adresare, spre deosebire de CISC, care are un număr mare de moduri de adresare (dar care nu sunt totdeauna utilizate).

Setul de instrucțiuni RISC este orientat pe registre (peste 32 de registre). Pentru că accesul la memorie e mult mai lent decât lucrul cu registrele, RISC încurajează lucrul cu aceștia. Face acest lucru prin creșterea numărului de registreși prin limitarea explicită a acceselor la memorie. În general instrucțiunile au 2 operanzi (registre) și un registru destinație. Ca principiu, arhitectura RISC are mai multe registre generale, în timp ce CISC are mai mulți speciali. Practic toate procesoarele moderne

împrumută atât caracteristici CISC, cât și RISC.

În cadrul arhitecturilor RISC există o limitare explicită, și anume: singurul mod de acces la memorie este prin *load* și *store*. Aceasta se deosebește fundamental de CISC care are instrucțiuni cu operanzi locații de memorie. Totuși, deși RISC impune această disciplină de lucru cu memoria, doar 20-25% din codul unui program e reprezentat de loads & stores.

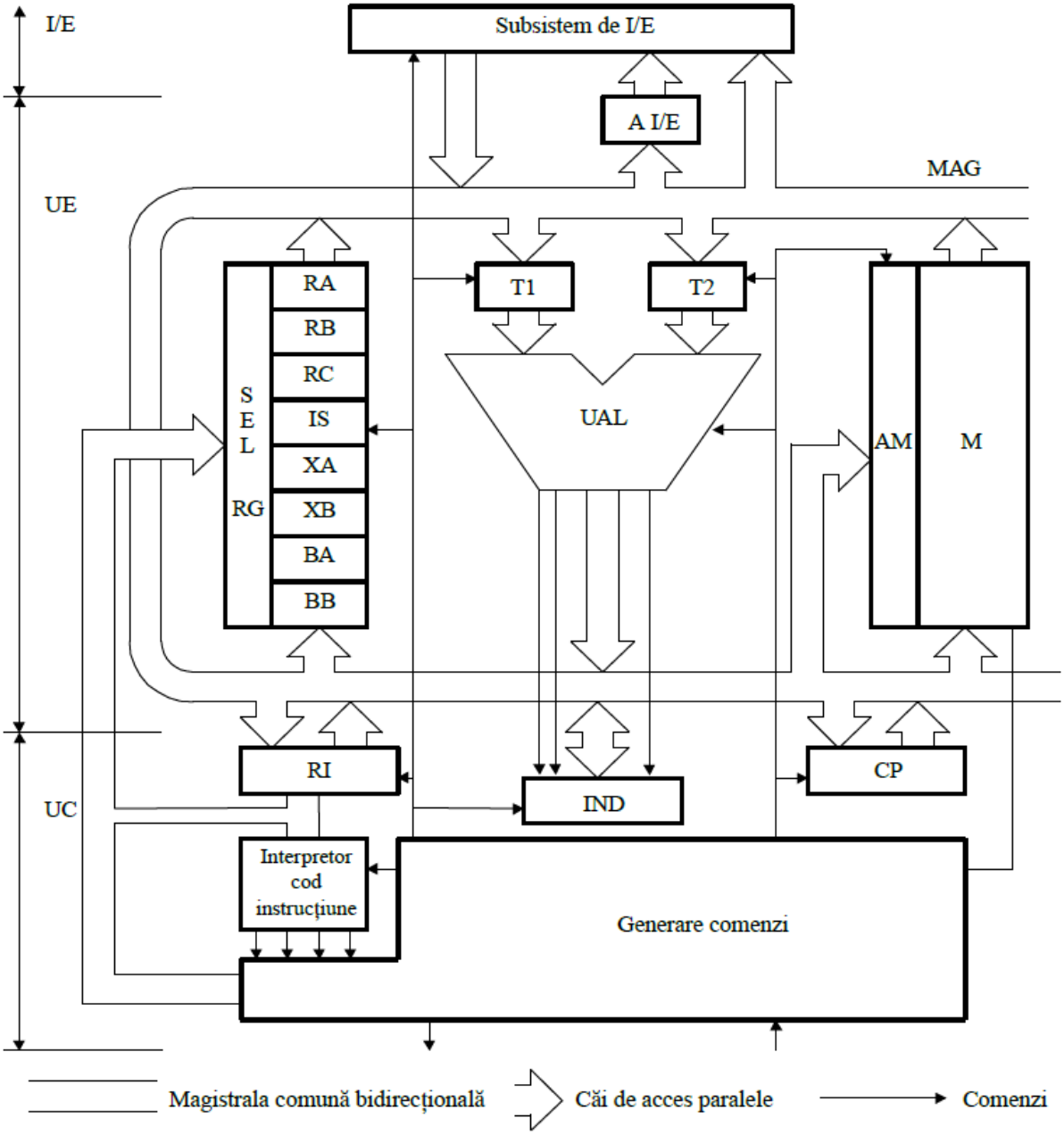
Observaţii

1. **Procesorul didactic este consistent din punctul de vedere al dimensiunii: 16 biţi**. Procesoarele reale, în marea lor majoritate, nu respectă aceasta regulă. Spre exemplu, Pentium 4 cu arhitectura pe 32 de biţi (IA-32) include registre pe 128 de biţi şi dispune de o magistrală de adrese pe 36 de biţi (spaţiul total de adresare este de 64 TB). În consecinţă părerile sunt impărţite în legătură cu care ar fi o definiţie corectă pentru dimensiunea procesorului. Cea mai frecventă definiţie spune că dimensiunea unui procesor este dată de dimensiunea registrelor şi a unităţilor aritmetico-logice.

2. **Spaţiul de adresare pentru un procesor pe N biţi este de 2^N locaţii de memorie**. În funcţie de organizarea acesteia însă, aceste locaţii pot fi octeţi sau cuvinte de mai multi octeţi. Memoria calculatorului didactic este organizată ca un **spaţiu contiguu de 64 Kcuvinte de 16 biţi fiecare**. Aşadar spaţiul total de adresare pentru calculatorul didactic este de 128 Kbytes. Memoria din calculatoarele voastre însă este adresabilă la nivel de octet. Dacă calculatorul didactic ar fi fost echipat cu o astfel de memorie, spaţiul de adresare ar fi fost de 64 KB, deoarece la fiecare locaţie se poate stoca fix un byte.

3. Prin faptul ca procesorul permite lucrul cu operanzi direct din memorie se înţelege că ei nu trebuie aduşi în prealabil de către programator într-un registru general. Cu toate acestea **nu se poate lucra cu ambii operanzi direct din memorie**. Acest tip de procesare specific arhitecturilor CISC poartă numele de procesare Registru-Memorie. Spre deosebire de aceasta, arhitecturile RISC tipice necesită încărcarea prealabilă a operanzilor în registrele generale. De aceea, aceste procesoare se mai numesc Registru-Registru sau Load/Store. Arhitecturile Memorie-Memorie sunt foarte rare, datorită complexităţii hardware-ului şi performanţelor scăzute.

Resursele calculatorului didactic



Arhitectura 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 lucrează la frecvența de ceas a procesorului. În tabelul de mai jos sunt sumarizate funcțiile acestora:

Registrul	Funcția
RA, RB, RC	La dispoziția programatorului pentru stocarea operanzilor. RA este folosit în lucrul cu porturile.
IS	Indicatorul de stivă.
XA, XB	Se pot folosi pentru stocarea operanzilor. Sunt folosiți pentru adresarea memoriei ca și registre index.
BA, BB	Se pot folosi pentru stocarea operanzilor. Sunt folosiți pentru adresarea memoriei ca și registre de bază.

Unitatea aritmetică logică UAL

Unitatea aritmetică logică (UAL) realizează operațiile aritmetice și logice ale calculatorului didactic. Ea este utilizată pentru prelucrarea datelor și pentru calculul adresei efective. Unitatea aritmetică logică prelucrează operanzi pe 16 biți reprezentați în cod complementar. Caracteristicile rezultatului (zero, par, transport și depășire) sunt depuse într-un registru de indicatori IND, în urma execuției oricărei instrucțiuni 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 64Kcuvinte 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. Când se dorește realizarea unei operații de citire din memorie, adresa solicitată va fi depusă în acest registru, iar unitatea de comandă va lansa comanda «Memory Read». După un anumit timp, memoria va furniza pe magistrală cuvantul de la adresa solicitată. Analog, când se dorește să se scrie la o anumită adresă din memorie un cuvant, aceasta este depusă în registrul AM, datele de scris sunt activate pe magistrală, iar unitatea de comandă 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. După încărcarea fiecarei instrucțiuni, el se va incrementa pentru a marca avansul la următoarea instrucțiune. În cazul în care instrucțiunea executată este una de salt,

adresa de salt va fi încărcata în CP în urma execuției instrucțiunii.

Registrul de instrucțiuni RI

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

Prin activarea ieșirii lui RI, cu ri_oe, doar deplasamentul este pus pe magistrală pentru a putea fi folosit în calcule (biții 8:15 sunt inversați și extinși la 16 biți prin multiplicarea bitului de semn pentru calculul adresei în instrucțiunile de salt condiționat).

Indicatorii de condiții IND

Registrul de indicatori constituie o grupare a unor flag-uri provenite din rezultatele instrucțiunilor de tip aritmetico-logic. Registrul IND permite unei instrucțiuni să folosească informații rezultate în urma execuției unei instrucțiuni anterioare.

Spre exemplu, dacă se dorește efectuarea unei sume cu operanzi pe 32 de biți, din moment ce dimensiunea procesorului este 16 biți, este nevoie să se prelucreze pe rând octeții inferiori, apoi octeții superiori. Cei doi operanzi vor ocupa două adrese consecutive de memorie, fie ele 0xA16, 0xA17 pentru primul, respectiv 0xA18, 0xA19 pentru cel de-al doilea. Suma va fi depusă la adresele 0xA20, 0xA21.

Iată programul în assembler pentru situația prezentată anterior:

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

După cum se poate observa, cea de-a doua operație aritmetică este ADC, adică adunare cu transport (carry). Semnificația este următoarea: dacă de la operația anterioară de adunare a apărut transport, acest transport va trebui propagat în octeții superiori. Astfel, operația ADD, în cazul în care a apărut transport, setează bitul T din registrul IND. Operația ADC realizează suma între operanzi, dar include în calcul și acest bit de transport.

Un alt exemplu este saltul condiționat. O instrucțiune de tip «if (a==2)» se va implementa astfel:

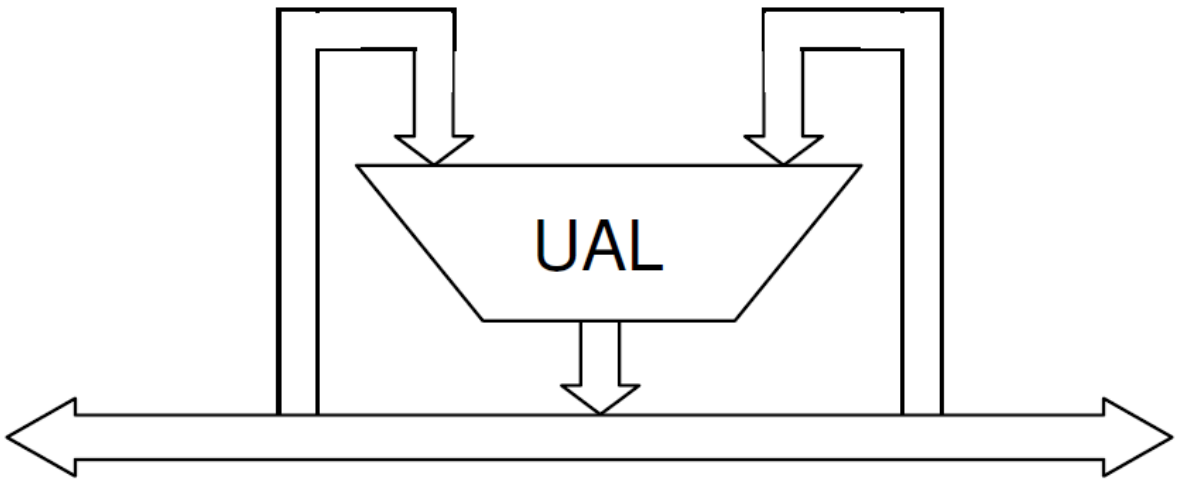
```
CMP RA,2
JNE etichetă
```

Prima instrucțiune are ca efect scăderea conținutului registrului RA cu 2. Rezultatul acestei operații nu se stochează, însă, ca orice operație aritmetico-logică, **va afecta registrul de indicatori**. Astfel, dacă rezultatul a fost zero (continutul lui RA a fost 2), se va seta flagul Z (zero) din registrul IND. Următoarea instrucțiune, JNE (jump if not equal) **va testa acest flag** și în functie de valoarea sa va realiza sau nu saltul.

Registrele temporare T1,T2

Registrele temporare T1 și T2 sunt utilizate pentru a păstra operanzii unei operații executate în unitatea aritmetică logică, precum și rezultate intermediare la calcularea adresei efective. Ele nu sunt accesibile în mod explicit programatorului.

Pentru a explica necesitatea existenței acestor registre, să consideram că ele nu ar fi existat. În acest caz, am fi obținut următoarea schemă:



UAL fără registre temporare

În acest caz, ambii operanzi precum și rezultatul operației ar trebui să se găsească simultan pe magistrală, ceea ce este imposibil.

Subsistemul de Intrări/Ieșiri

Subsistemul de intrări și ieșiri permite procesorului comunicația cu mediul extern prin intermediul dispozitivelor periferice. Subsistemul este format din interfețe (spre exemplu interfațele serială, paralelă, IDE, USB, etc.) capabile să comunice cu dispozitivele periferice în conformitate cu un standard. Aceste interfețe includ un set de registre (de date/stări/comenzi) pentru comunicația cu procesorul. De exemplu, când procesorul dorește să transmită un cuvânt de date prin interfața USB,

el va depune în registrul de date asociat interfeței USB cuvântul respectiv, apoi va scrie în registrul de comenzi comanda de transmisie. Fiecare astfel de registru este identificat printr-o adresă unică în sistem și poartă numele de port de intrare/ieșire. Așadar, totalitatea registrelor asociate interfețelor din sistem (porturilor) este echivalentă cu o memorie în care fiecare adresă este asociată unei interfețe.

Registrul de adrese de intrare/ieșire AIE

Întrucât subsistemul de intrări/ieșiri apare procesorului ca o memorie în care fiecare locație reprezintă un port asociat unei interfețe, ca și în cazul memoriei este nevoie de un registru de adrese. Acesta va fi folosit pentru a stoca adresa portului cu care se dorește să se comunice. Spre exemplu, atunci când procesorul vrea să transmită ceva pe interfața paralelă, el va depune în registrul AIE valoarea 0x378 (ce identifică portul asociat interfeței paralele), va activa pe magistrală cuvantul de date ce se dorește a fi transmis, iar unitatea de comandă va lansa semnalul «I/O Write» ce va determina încărcarea cuvântului în registrul interfeței paralele.

Unitatea de comandă

Toate resursele prezentate până în acest punct formează unitatea de execuție. Unitatea de comandă dirijează aceste resurse pentru a executa o instrucțiune sau o alta. Spre exemplu, în cazul instrucțiunii ADD RA, RB unitatea de comandă va genera urmatoarea secvență de semnale:

1. Semnal către blocul de registre generale pentru a determina activarea conținutului lui RA pe magistrală.
2. Semnal către T1 pentru a încărca valoarea aflata în acest moment pe magistrală.
3. Semnal către blocul de registre generale pentru a determina activarea conținutului lui RB pe magistrală.
4. Semnal către T2 pentru a încărca valoarea aflată în acest moment pe magistrală.
5. Semnal către UAL ce indică operația de adunare.
6. Semnal către IND pentru a seta flag-urile Z, S, D, T, P.
7. Semnal către blocul de registre generale pentru a încărca în RA valoarea de pe magistrală.

Cu alte cuvinte, unitatea de comandă este cea care controlează funcționarea procesorului. Ea dirijează întreg procesul de citire-interpretare-execuție a instrucțiunilor.

Descrierea AHPL de la curs este în realitate descrierea funcționala a acestei unități de comandă. În momentul în care această descriere este gata, ce mai rămâne de facut în proiectarea unui procesor real este de a stabili ce semnale de comandă trebuiesc generate la fiecare pas.

Moduri de adresare

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 găsi:

- Ambii în registrele generale RG;
- Unul în registrele generale RG și altul în memorie;
- Unul în registrele generale RG și altul în cadrul instrucțiunii respective (numit operand imediat);
- Unul în memorie și altul imediat.

Modurile de adresare permise de calculatorul didactic sunt tipice arhitecturilor CISC, fiind derivate din arhitectura standard x86. În total, procesorul permite 11 moduri de adresare, ceea ce îi conferă o foarte bună flexibilitate în programare. Pentru o mai ușoară înțelegere, le vom structura astfel:

Specificarea operandului	**Moduri de adresare**
Operandul nu se găseşte în memorie	Adresare directă la registru Adresare imediată
Operandul e specificat doar prin deplasament	Adresare directă Adresare indirectă
Operandul e specificat doar prin registre	Adresare indirectă prin registru Adresare indirectă prin suma de registre Adresare indirectă prin suma de registre cu autoincrementare Adresare indirectă prin suma de registre cu autodecrementare
Operandul e specificat prin registre și deplasament	Adresare Bazată Adresare Indexată Adresare Bazată Indexată

În continuare vor fi prezentate și discutate aceste moduri de adresare.

Operandul nu se găsește în memorie

1. Adresare directă la registru

Operandul se găseşte în RG.

Exemplu:

```
MOV RA, RB
```

Instrucțiunea are ca efect încărcarea în registrul RA a valorii din registrul RB.

2. Adresare imediată

Operandul este specificat în instrucțiune.

Exemplu:

```
MOV RA, 7
```

Instrucțiunea va avea ca efect încarcarea valorii “7” în registrul RA. “7” poartă numele de operand imediat.

Operandul e specificat doar prin deplasament

3. Adresare directă

Adresa efectivă este specificată în instrucțiune.

Exemplu:

```
MOV RA, [12]
```

Instrucțiunea va încărca valorea aflată în memorie la adresa “12” în registrul RA. “12” poarta numele de deplasament.

4. Adresare indirectă

Adresa efectivă se citește din memorie, din locația a cărei adresă este specificată în instrucțiune.

Exemplu:

```
MOV RA, [[12]]
```

Instrucțiunea are ca efect încărcarea valorii aflată la adresa ce se găsește în memorie la adresa “12”. Acest mod de adresare seamană foarte bine cu pointerii din C. La adresa “12” se găsește pointer-ul către operand.

Operandul e specificat doar prin registre

5. Adresare indirectă prin registru

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

Exemplu:

```
MOV RA, [BA]
```

Instrucțiunea va încărca în RA valoarea aflată în memorie la adresa conținută în 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]
```

Instrucțiunile încarcă în RA valoarea aflată în memorie la adresa obținută prin adunarea conținutului registrelor BA și XA. BA, BB poartă numele de registre de bază, iar XA, XB poartă numele de registre index. Întotdeauna suma se va face între un registru bază și 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 instrucțiuni sunt echivalente cu:

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

8. Adresare indirectă prin sumă de registre cu autodecrementare

Adresa efectivă se obține prin suma unui registru de bază cu continutul registrului XA. Înaintea generării adresei, registrul XA este decrementat.

Exemplu:

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

sau

```
MOV RA, [BA+XA-]
```

Aceste instrucțiuni sunt echivalente cu:

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

Observații: Motivul pentru care XB nu poate fi utilizat în acest mod de adresare nu este unul logic, ci unul tehnic: biții din codul de instrucțiune nu au fost suficienți pentru a se putea codifica și acest mod.

Operandul e specificat prin registre și deplasament

9. Adresare bazată

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

Exemplu:

```
MOV RA, [BA]+7
```

sau

```
MOV RA, [BA+7]
```

Instrucțiunea încarcă în registrul RA valoarea aflată în memorie la adresa rezultată în urma adunării conținutului registrului de bază 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]
```

Instrucțiunea încarcă în registrul RA valoarea aflată în memorie la adresa rezultată în urma adunării conținutului registrului de index cu deplasamentul (7).

11. Adresare bazată indexată

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

Exemplu:

```
MOV RA, [BA][XA]+7
```

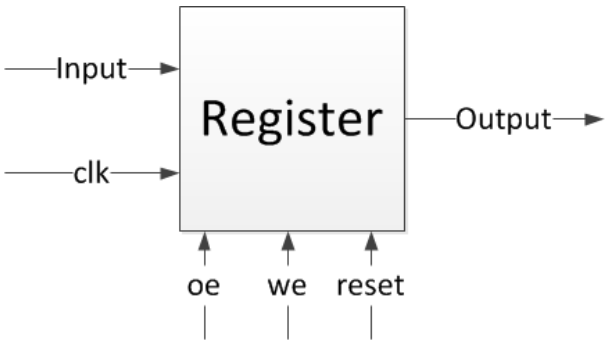
sau

```
MOV RA, [BA+XA+7]
```

Instrucțiunea încarcă în registrul RA valoarea aflată în memorie la adresa rezultată în urma adunării conținutului registrului de bază (BA) cu registrul index (XA) și cu deplasamentul (7).

Exerciții

Partea practică a acestui laborator constă în realizarea unei implementări incipente a calculatorului didactic. Scheletul de laborator cuprinde structura generală a calculatorului împreună cu toate modulele folosite, oferite sub formă binară.



Interfață registru

1. **(1p)** Explicați conținutul fișierului *calc_didactic/calc_didactic.v* corelându-l cu schema bloc a calculatorului didactic din secțiunea Resursele calculatorului didactic. Rulați proiectul pe placa de laborator și urmăriți execuția primei instrucțiuni.
   - *Hint:* Citiți secțiunea testare (N/A) pentru informații despre testarea calculatorului didactic pe placa de laborator.
2. **(3p)** Implementați modulul *register* pornind de la declarația din fișierul *calc_didactic/register.v*.
   - *Hint:* Revedeți laboratorul 5.
   - *Hint:* Pentru testarea pe placă (N/A) revedeți tutorialul de programare. Asignarea pinilor I/O este deja realizată în fișierul .ucf din schelet.
3. **(3p)** Implementați modulul *registers*, care reprezintă bancul celor 8 registre generale, pornind de la declarația din fișierul *calc_didactic/registers.v*.
   - Semnalul `addr` identifică registrul selectat (dintre cele 8) pentru citire/scriere. Corespondența dintre valoarea lui `addr` și registrul selectat este dată de prima și, respectiv ultima coloană a celui de-al doilea tabel din cheat-sheet.
   - Semnalele `disp_addr` și `disp_out` sunt folosite pentru afișare/debugging. `disp_out` va trebui să aibă valoarea memorată de registrul selectat de `disp_addr` în momentul curent. În mod normal aceste semnale nu sunt prezente într-un calculator. Aceste semnale **nu trebuie** să fie afectate de `oe` și trebuie să fie **asincrone**: valoarea disponibilă pe `disp_out` va în orice moment egală cu valoarea memorată de registrul selectat cu `disp_addr`. Efectul lui `disp_addr` trebuie să fie imediat și nu se va aștepta tranziția semnalului de ceas.
   - *Hint:* Pentru declararea array-urilor multidimensionale revedeți laboratorul 3.
4. **(3p)** Implementați în unitatea de comandă etapa de *fetch* a execuției unei instrucțiuni.

Testare (N/A)

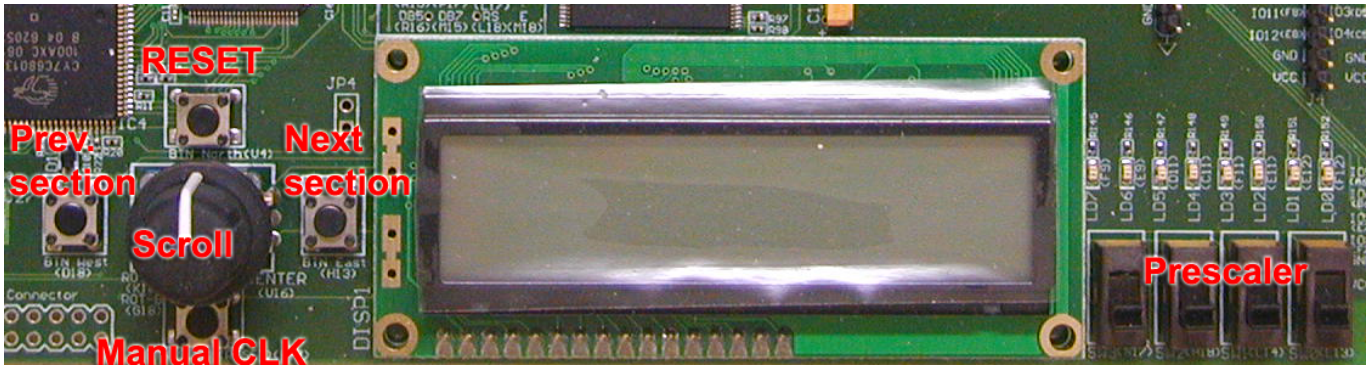
Testarea implementării se face pe placa de laborator, scheletul de cod conținând deja o variantă binară a tuturor modulelor necesare pentru functionare. Această variantă binară este înlocuită

automat în momentul în care completați conținutul modulelor cerute cu implementarea voastră.

Memoria calculatorului didactic este creată cu ajutorul unui *IP core* pentru RAM oferit de Xilinx și este inițializată din conținutul fișierului *calc_didactic/ram.coe*, care codifică programul descris de fișierul *calc_didactic/ram.asm*. La prima compilare acest *core* pentru RAM trebuie generat, împreună cu un alt core folosit pentru afișarea VGA. Răspundeți cu *Yes* în momentul în care sunteți întrebați dacă doriți regenerarea *core*-ului. Veți fi întrebați de două ori, pentru cele două *core*-uri folosite.

Scheletul permite afișarea stării resurselor interne ale calculatorului didactic. În mod implicit această afișare este făcută pe portul VGA. Pentru a vedea aceste informații conectați un monitor la portul VGA al plăcii de laborator. O a doua modalitate de a vizualiza aceaste informații este prin intermediul ecranului LCD text prezent pe placă. În acest caz trebuie însă să modificați conținutul fișierului *top_calc_didactic.v* pentru a ruta afișarea către ecranul LCD. Pentru aceasta schimbați valoarea parametrului `lcd_ui`, de la linia 47, în 1, apoi recompilați proiectul.

Scheletul permite următoarele modalități de interacțiunea cu calculatorul didactic:



Butoanele folosite pentru controlarea calculatorului didactic

- Switch-urile *SW3-SW0* controlează factorul de divizare al unui prescaler conectat la ceasul de 50MHz al plăcii. Ceas-ul divizat rezultat din prescaler este folosit ca semnal de ceas pentru calculator. Poziționarea tuturor switch-urilor pe 0 reprezintă un caz special în care ceasul calculatorului este controlat manual.
- Butonul *BTN_SOUTH* reprezintă semnalul de ceas manual.
- Butonul *BTN_NORTH* reprezintă semnalul de *reset*. El este activ pe 1 și este sincron (are nevoie de o tranziție de ceas pentru a fi recepționat).
- Butoanele *BTN_WEST* și *BTN_EAST* permit ciclarea prin diferite secțiuni de resurse care sunt afisate pe LCD-ul text: registre generale (RA, RB, RC etc.), memorie și resurse interne (CP, RI, T1 etc.).
- Butonul rotativ BTN_ROT permite ciclarea prin resursele din cadrul secțiunii curente afișate pe LCD-ul text.

Resurse

- Schelet de cod
- PDF laborator
- Cheat-sheet calculator didactic
- Arhitectura calculatorului didactic
- PDF laborator

- Soluție laborator