

Laboratorul 9 - Calculatorul Didactic: Instrucțiuni cu doi operanzi

Scopul laboratorului este de a implementa execuția instrucțiunilor cu 2 operanzi (ADD, ADC, SUB, SBB, AND, OR, XOR, CMP și TEST) în unitatea de comandă.

Implementarea unei instrucțiuni în unitatea de comandă

În [laboratorul 8](#) au fost enumerate cele patru etape de prelucrare a instrucțiunilor:

- *Fetch* - aducerea instrucțiunii din memorie în registrul instrucțiune (RI)
- *Decode* - decodificarea instrucțiunii
- *Execute* - executarea instrucțiunii
- *Write-back* - scrierea rezultatului (dacă este cazul)

Ce trebuie să facă unitatea de comandă atunci când trebuie executată o instrucțiune aritmetică logică cu doi operanzi? De exemplu ADC RA, RB.

1. Semnale către CP și AM: valoarea din CP e pusă pe magistrală și scrisă în AM;
2. Semnale către AM și memorie pentru a lua valoarea (codul instrucțiunii) de la adresa specificată de AM;
3. Semnal către RI pentru a pune codul instrucțiunii în el
4. Decodificare instrucțiune;
5. Semnal către blocul de registre generale pentru a determina activarea conținutului lui RA pe magistrală;
6. Semnal către T1 pentru a încărca valoarea aflată în acest moment pe magistrală;
7. Semnal către blocul de registre generale pentru a determina activarea conținutului lui RB pe magistrală;
8. Semnal către T2 pentru a încărca valoarea aflată în acest moment pe magistrală;
9. Semnal către UAL ce indică operația ADC;
10. Semnal către IND pentru ca UAL-ul să poată scrie flagurile;
11. Semnal către blocul de registre generale pentru a încărca în RA valoarea de pe magistrală;
12. Incrementarea CP pentru adresa instrucțiunii următoare - semnal către CP pentru a scrie în el.

Implementare

Unitatea de comandă este implementată ca un automat de stări. Modulul acesteia are următoarele semnale:

- intrări: *clk*, *rst*, *ri* (codul instrucțiunii), *ind* (indicatorii de condiție)
- ieșiri:
 - semnale *oe* (output-enable) și *we* (write-enable) pentru registre, bancul de registre, memorie și unitatea aritmetică logică (doar *oe*)
 - *alu_opcode* - codul operației ce trebuie efectuată de unitatea aritmetică logică
 - *alu_carry* - carry-ul folosit de UAL în cadrul operației ce trebuie să o execute
 - *regs_addr* - indexul unui registru din bancul de registre
 - *ind_sel* - controlează sursa de scriere în registrul IND (0 = bus, 1 = alu flags)

Automatul trebuie să ofere stări pentru:

- aducerea instrucțiunii din memorie în registrul *RI*
- decodificarea codului instrucțiunii pentru identificarea operației ce trebuie efectuate și a operanzilor acesteia (dacă este cazul)
- interpretarea fiecărei instrucțiuni. Aceasta se traduce printr-o serie de stări care setează semnalele de output ale modului pentru a comanda execuția instrucțiunii.
- incrementarea registrului *CP*

La fel ca în laboratorul trecut, în implementarea unității de comandă vom considera că UAL-ul va pune rezultatul în T1, și de acolo va fi transferat în registre sau în memorie. Această convenție simplifică stările care comandă execuția operațiilor aritmetice-logice.

Codificarea instrucțiunilor

Formatul instrucțiunilor este același precum cel din [laboratorul 8](#), așa cum se observă din [Fig. 1](#).

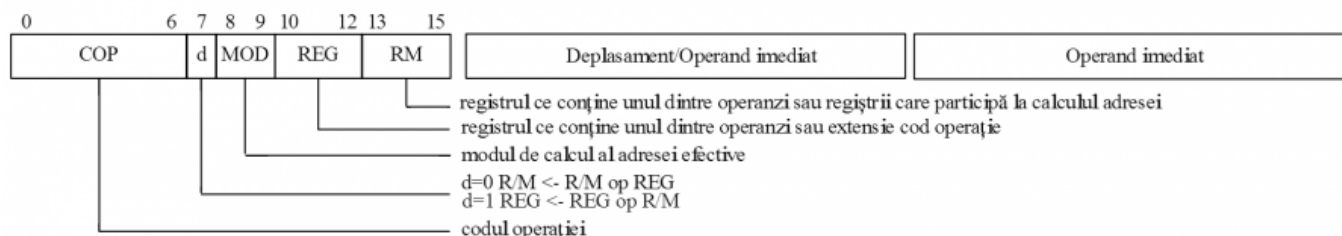


Fig. 1: Formatul instrucțiunilor calculatorului didactic

Click AICI pentru explicarea câmpurilor din Fig. 1.

- **COP** - codul operației, 7 biți
 - bitul [0] - separă instrucțiunile care folosesc o adresă efectivă de cele care nu folosesc:
 - 0 - instrucțiuni cu calcul de adresă efectivă
 - 1 - instrucțiuni fără calcul de adresă efectivă (salturi condiționate, ret etc.)
 - bitul [1] - separă instrucțiunile care au 1 operand de cele cu 2 operanzi:
 - 0 - un operand
 - 1 - doi operanzi
 - bitul [2] - separă instrucțiunile cu operand imediat de cele fără operand imediat:
 - 0 - fără operand imediat
 - 1 - cu operand imediat
 - bitul [3] - separă instrucțiunile de transfer de date/control de celelalte:
 - 0 - transfer de date/control (mov, push, call etc.) sau care nu salvează rezultatul (cmp, test)

- 1 - instrucțiuni aritmetico-logice cu salvarea rezultatului, salturi condiționate
 - biții [4][5][6] - cod operație
- *d* - pentru instrucțiunile cu doi operanzi, folosit pentru a ști care e primul și care e al doilea dintre cele două câmpuri REG și RM din codul instrucțiunii:
 - 0 - RM = RM op REG
 - 1 - REG = REG op RM
- *MOD* - modul de calcul al adresei efective (4 moduri - 2 biți)
- *REG* - indexul registrului care conține unul dintre operanzi
- *RM* - indexul registrului care conține unul dintre operanzi

Instrucțiunile care vor fi implementate în acest laborator se regăsesc în [Tab. 1](#). Acestea sunt instrucțiuni aritmetice și logice cu doi operanzi ale căror valori sunt în registrele generale și fie pun un rezultat înapoi în registrul destinație, fie nu stochează rezultatul ci doar setează indicatorii de condiție (CMP și TEST).

Instrucțiune	Funcție	Cod RI[0:6]
ADD	$op_{dst} = op_{dst} + op_{src}$	0101000
ADC	$op_{dst} = op_{dst} + op_{src} + carry$	0101001
SUB	$op_{dst} = op_{dst} - op_{src}$	0101010
SBB	$op_{dst} = op_{dst} - op_{src} - carry$	0101011
AND	$op_{dst} = op_{dst} \& op_{src}$	0101100
OR	$op_{dst} = op_{dst} op_{src}$	0101101
XOR	$op_{dst} = op_{dst} \wedge op_{src}$	0101110
CMP	$op_{dst} - op_{src}$, fără stocare rezultat, doar setare indicatori	0100010
TEST	$op_{dst} \& op_{src}$, fără stocare rezultat, doar setare indicatori	0100100

Tab. 1: Instrucțiuni aritmetice-logice cu doi operanzi.

Pentru **decodificarea** instrucțiunilor din acest laborator trebuie să identificăm atât grupul instrucțiunilor aritm-logice cu doi operanzi, fără operand imediat și care stochează rezultatul ($RI_{0..3} = 0101$) cât și cele care nu stochează rezultatul ($RI_{0..3} = 0100$).

Adresarea directă la registru

Operanzii se găsesc în registrele specificate de câmpurile REG și RM. Pentru a selecta ordinea operanzilor din aceste două câmpuri, trebuie să ținem cont de valoarea din bitul *d*. Vom folosi adresarea directă la registru (câmpul mod trebuie să conțină valoarea 2'b11), așa cum se observă în [Fig. 2](#).

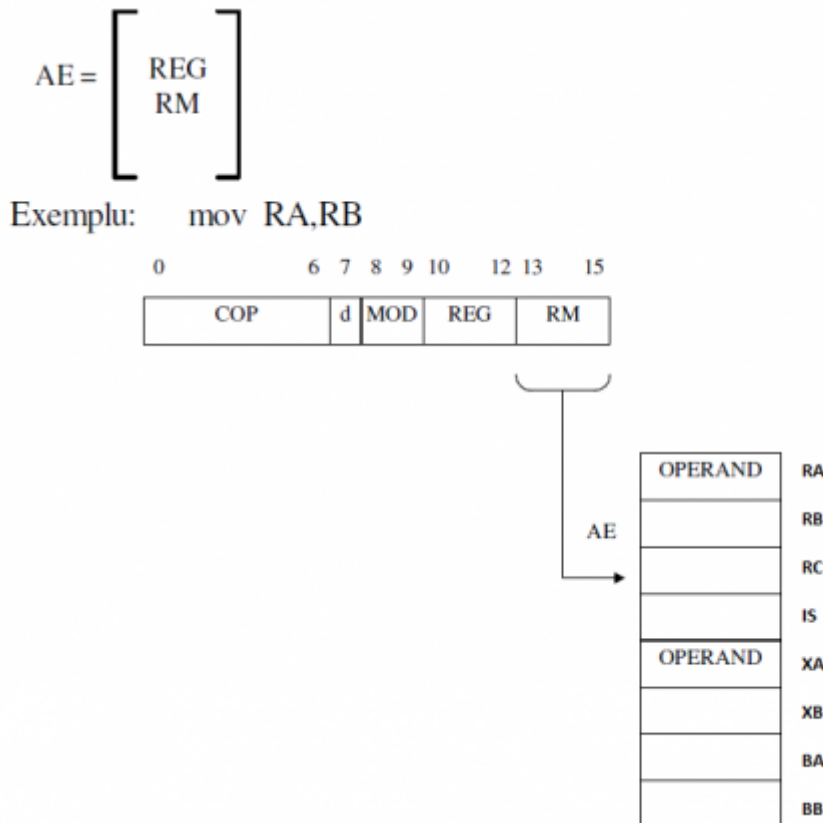


Fig. 2:

Adresarea directă la registru pentru instrucțiuni cu doi operanzi.

Instrucțiunea `AND RA, RB` efectuează ȘI logic între cei doi operanzi și pune rezultatul în registrul destinație (RA). Un exemplu de calcul al codului operației se regăsește mai jos:

- [0] = 0 - cu calcul de adresă efectivă
- [1] = 1 - doi operanzi
- [2] = 0 - fără operand imediat
- [3] = 1 - operație aritmetică cu salvarea rezultatului
- [4,5,6] = [1,0,0] - codul dat operației (stabilit de arhitectură)
- [7] - selectează operandul destinație. Pentru exemplul acesta vom considera că acest bit ia valoarea 1 (destinația va fi registrul RA):
 - 1: REG = REG AND RM
- [8,9] = [1,1] - modul de adresare (folosim adresarea directă la registru)
- [10,11,12] = [0,0,0] - indexul registrului RA în bancul de registre
- [13,14,15] = [0,0,1] - indexul registrului RB în bancul de registre

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	0	0	1	1	0	1	0	

Exerciții

Modificați modulul `uc.v` din scheletul de cod al laboratorului, astfel încât să implementați instrucțiunile

din tabelul [Tab. 1](#). Citiți secțiunea de [implementare](#).

1. (4p) Decodificați instrucțiunea ADD reg_{dst}, reg_{src} .
 1. Identificați grupul de instrucțiuni în care se încadrează operația (ADD) și modul de adresare.
 2. Identificați operandul destinație.
 3. Transferați conținutul registrului reg_{src} în registrul T2
 4. Indicați UAL-ului să execute operația și puneți rezultatul în T1
 - Registrul IND trebuie conectat la UAL ca să poată fi setate flagurile în urma execuției operației
2. (6p) Decodificați restul instrucțiunilor.

Resurse

- [Schelet de cod](#)
- [Soluție laborator](#) (disponibilă începând cu 30.11.2019)
- [PDF laborator](#)
- [Cheat-sheet calculator didactic](#)
- [Arhitectura calculatorului didactic](#)

From:

<https://elf.cs.pub.ro/ac/wiki/> - **AC Wiki**

Permanent link:

<https://elf.cs.pub.ro/ac/wiki/lab/lab09>

Last update: **2019/11/23 18:10**

