2.4 Formatul instructiunilor

Vom construi codul masina pentru acest procesor. Pentru aceasta, este important de precizat ca vom lucra in stilul unui procesor pe 8 biti (putem lucra cu numere pana la cel mult 255). Formatul instructiunilor noastre va fi putin diferit, pentru ca trebuie sa putem inlantui expresii de complexitatea celor amintite anterior, iar pentru, de exemplu, `-7 18 + 13 / -5 *` nu putem avea instructiuni standard, dar le putem face cu lungime fixa.

Pentru codificarea operanzilor avem urmatoarea structura:

b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	identificator		bit semn	codificare operand							

- Codificarea este pe 12 biti (1.5 Bytes);
- semnificatia bitilor este urmatoarea:
  - primul bit, b_0 este intotdeauna egal cu 1;
  - urmatorii 2 biti identifica ce tip are operandul: daca avem 00, atunci operandul este numar, iar daca avem 01, atunci operandul este variabila;
  - daca am avut identificatorul de numar, 00, atunci b_3 este bit-ul de semn: 0 inseamna ca numarul este considerat pozitiv, respectiv 1 inseamna ca numarul este considerat negativ. Daca am avut identificatorul 01, atunci bit-ul b_3 de semn va fi considerat 0;
  - operandul poate fi, dupa caz, asa cum am vazut: sau numar pozitiv (de la 0 la 255), sau numar negativ (tot de la 0 la 255, dar cu semn schimbat), sau o variabila - variabilele sunt formate **doar dintr-o singura litera**. De exemplu, daca avem in codificarea operandului `01111000`, 1111000 este, de fapt, 120 in baza 10, care corespunde codului ASCII pentru `x`. Astfel, o codificare completa pe 12 biti `1 01 0 01111000` inseamna ca operandul curent este variabila x. Daca am fi avut aceeasi reprezentare in operand, dar cu alt cod de identificator, de exemplu 00, `1 00 0 01111000`, ar fi fost numarul intreg 120, iar daca aveam bitul de semn 1, `1 00 1 01111000`, ar fi fost numarul intreg -120.

Ramane doar sa codificam operatiile. Ele vor respecta o structura similara - un bit initial 1, un identificator, si un cod de operatie aplicata pana la lungimea de 12 biti:

b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	identificator		codificarea operatiei								

Avem urmatoarele codificari:

Operatie	Codificare
let	000000000
add	000000001
sub	000000010
mul	000000011
div	000000100

Identificatorul este, in acest caz, **10** (cei doi biti care precizeaza identificatorul). Astfel, reprezentarea unei operatii *add* ar fi `1 10 000000001`.

Pentru claritate, vom prezenta identificatorii in urmatorul tabel:

Identificator	Semnificatie
00	numar intreg
01	variabila
10	operatie

2.5 Un exemplu de translatare

Sa consideram ca vrem sa reprezentam instructiunea `x 1 let x -14 div`.

Vom folosi formatele descrise mai sus, si vom reprezenta pe rand fiecare camp.

1. `x`: cum a fost aratat intr-un exemplu de mai sus, `x` se codifica `1 01 0 01111000`. Vom grupa cate 4 cifre, astfel ca reprezentarea lui `x` este `1010 0111 1000`.
2. il codificam pe `1`. Este o codificare de operand numar intreg pozitiv, deci avem identificatorul `00` si bit-ul de semn `0`. Reprezentarea lui va fi `1 00 0 00000001`, iar pe gruparea cate 4 cifre `1000 0000 0001`.
3. `let` este o operatie, se va codifica precum o operatie, avand deci identificatorul `10`, respectiv codul de operatie `000000000`. Reprezentarea lui va fi `1 10 000000000`, iar pe gruparea cate 4 cifre `1100 0000 0000`.
4. `x` se va reprezenta din nou ca `1010 0111 1000`.
5. `-14` se reprezinta ca operand numar intreg negativ, deci cu identificatorul `00`, bit-ul de semn `1`, iar valoarea `14` in baza $2 \rightarrow$ `1110`, dar pe 8 biti, deci cu 4 biti de 0 la stanga (nesemnificativi): `00001110`. `-14` va fi reprezentat, deci `1 00 1 00001110`, iar pe gruparea cate 4 cifre `1001 0000 1110`.
6. In final, `div` este o operatie, deci identificatorul `10`, si codificarea oepratiei conform tabelului `000000100`, astfel ca va fi `1 10 000000100`, iar pe gruparea cate 4 cifre `1100 0000 0100`.

In acest caz, putem concatena toate reprezentarile binare, si sa avem:

```
1010 0111 1000 1000 0000 0001 1100 0000 0000 1010 0111 1000 1001 0000 1110 1100 0000
0100
```

respectiv in hexa:

```
A7 88 01 C0 0A 78 90 EC 04
```

Avem, astfel, ca operatia din *assembly*-ul nostru, `x 1 let x -14 div`, se translateaza in cod masina in reprezentare hexa in `A7 88 01 C0 0A 78 90 EC 04`.

Cerinta:

Fie dat ca input un sir hexa, se cere sa se afiseze la *standard output* instructiunea *assembly* de executat.

De exemplu, pentru inputul `A78801C00A7890EC04`, se va afisa la standard output `x 1 let x -14 div`.

Sursa : https://cs.unibuc.ro/~crusu/asc/Arhitectura%20Sistemelor%20de%20Calcul%20(ASC)%20-%20Tema%201%20Laborator%202021.pdf