

## Tema 3 - Miniprocessor

---

### Responsabili:

- Costin Carabas

Termen de predare:

- Deadline soft: **16 Ianuarie 2023**
- Deadline hard: **20 Ianuarie 2023**

Pentru fiecare zi (24 de ore) de întârziere, se vor scădea 10 puncte din nota acordată, până la atingerea deadline-ului hard.

### Întrebări

Dacă aveți nelămuriri, puteți să ne contactați pe forumul dedicat [temei de casă nr. 3](#).

### Cerință

---

Să se implementeze un interpretor de biți similar unui procesor. Acesta va avea capacitatea de a decodifica și executa instrucțiuni simple de adunare, scădere, înmulțire și împărțire.

La nivelul cel mai de bază, informația este stocată sub forma de biți. Pentru un procesor aceasta informație se repartizează în 2 categorii: instrucțiuni și date. Practic, dându-se un șir de biți, procesorul decodifică instrucțiunea, iar ulterior o execută.

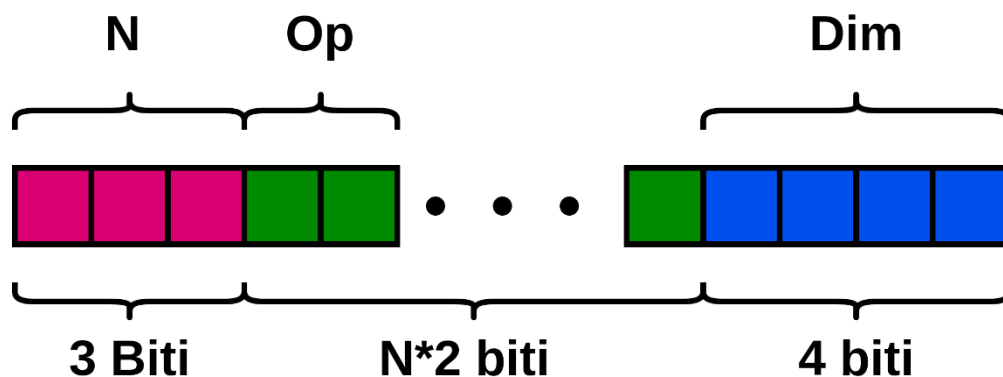
În aceasta tema, vom implementa un procesor de baza care decodifică un șir de biți și ulterior îl execută.

### Task 1 - Decodificare instrucțiune (50p)

---

Dându-se o instrucțiune în format binar, decodificați instrucțiunea.

O instrucțiune are următorul format:



Unde:

- **N** reprezintă numărul de instrucțiuni ce vor fi executate; acesta este reprezentat pe 3 biți și se obține prin convertirea valorii celor mai semnificativi 3 biți din binar în decimal și adunarea cu 1. Astfel, pentru **000** vom avea de executat o instrucțiune, pentru **010** vom avea de executat 3 instrucțiuni. Numărul maxim de instrucțiuni de executat este 8.
- **Op** reprezintă codul unei instrucțiuni și se reprezintă pe 2 biți. **Op** poate fi **+**, **-**, **\*** sau **/** conform tabelului de mai jos:

Cod	Operatie
00	+
01	-
10	*
11	/

În sirul de input, cei 3 biți care desemnează numărul de operații sunt urmați de număr de  $N*2$  biți care desemnează operațiile ce urmează a fi executate.

- **Dim** reprezintă dimensiunea unui operand și se reprezintă pe 4 biți. **Dim** se calculează similar cu **N** prin transformarea celor mai puțini semnificativi 4 biți în valoarea zecimală și adunarea cu 1. Astfel, dimensiunea operandilor poate lua valori din intervalul [1, 16].

În cadrul acestui exercițiu, veți citi de la intrarea standard un număr de tipul **unsigned int** ce conține instrucțiunea și o veți decodifica. Astfel, la ieșirea standard veți afișa **N**, operatorii și dimensiunea operandilor, toate separate printr-un spațiu.

Exemplu:

Input: 1675493376 → Output: 4 + - / \* 16

Input: 2483421184 → Output: 5 \* \* + + + 13

Precizări:

- dimensiunea totală a unei instrucțiuni nu poate depăși  $3 + 8*2 + 4 = 23$  biți, adică ar trebui să încapă într-un **unsigned int**.

- citirea de la tastatură se va face folosind funcția **scanf("%u", &inst)**
- formatul de afișare este: **N op1 op2 .. opN Dim**
- rezolvarea acestui task se va afla în fișierul **task1.c**

## Task 2 - Executare instrucțiune (50p)

În cadrul acestui exercițiu vom continua task-ul anterior prin:

1. **Citirea operandilor.** Plecând de la programul anterior, adăugați o secțiune de cod care interpretează  $(N+1)$  operanzi de la intrarea standard. Pentru acest task, dimensiunea operandilor (**Dim**) este un număr putere a lui 2 din intervalul [1, 16]. Adică valorile posibile sunt: 1, 2, 4, 8, 16. Operandii vor fi citiți sub forma unor numere **unsigned short** (dimensiune 16) de la intrarea standard. Numărul de operanzi citiți de la tastatură

se va descompune în mai multe de numere **unsigned short**, folosind formula:  $((N+1)*Dim)/16$ , la care adaugam +1 in cazul in care rezultatul are vreun rest. Astfel, se vor citi de la tastatură  $((N+1)*Dim)/16$  numere și vor fi descompune în (N+1) operanzi.

Exemplu:

Pentru  $N = 3$  si  $Dim = 4$ , folosim instrucțiunea 1410859008, output-ul de la Task 1 va fi  $3 * * + 4$ . Conform formulei de mai sus  $((N+1)*Dim)/16$ , vom citi un singur **unsigned short** de la tastatura  $((3+1)*4)/16 = 1$ . Presupunem că vom citi 54999. Valoarea acestuia in binar este: **1101 0110 1101 0111**. Practic, primul operand va fi 13, al doilea 6, al treilea 13, iar al patrulea 7.

Daca  $N = 4$  si  $Dim = 8$ , folosim instrucțiunea 1947074560, output-uul de la Task 1 va fi  $4 * * + + 8$ . Rezulta ca vom citi 3 **unsigned short** de la tastatura  $((4+1)*8)/16 = 2,5$ . Presupunem ca vom citi 54998 (**11010110 11010110**), 64041 (**11111010 00101001**) si 42752 (**10100111 00000000**). Practic, primul operand va fi 214, al doilea tot 214, al treilea 250, al patrulea 41, iar al cincilea 167, urmat de bitii de padding.

**2. Executarea instructiunii:** din moment ce avem atât operațiile cât si operanzii, nu ne mai ramane decât să calculăm rezultatul. Calcularea rezultatului se va face in ordinea primirii operatiilor si nu conform prioritatii operatorilor (adica, \* nu are precedenta fata de +).

Exemplu:

In cazul in care avem operatiile  $+ - * +$  si operanzii 1 2 3 4 5 se va valcula  $1 + 2 - 3 * 4 + 5 = 5$  ( $1 + 2 = 3 - 3 = 0 * 4 = 0 + 5 = 5$ )

Pentru acest task, se vor citi de la tastatura instructiunea si operanzii si se va afiza rezultatul:

Exemplu utilizare:

```
./task1
1410859008
3 * * + 4
Numere de introdus = 1
Introduceți număr: 54999
Operanzii: 13 6 13 7
Rezultat: 1021
```

sau

```
./task1
1947074560
4 * * + + 8
```

Numere de introdus = 3

Introduceți număr: 54998

64041

42752

Operanzii: 214 214 250 41 167 0 0 0 0 0

Rezultat: 11449208

### Precizari:

- rezolvarea acestui exercițiu se va afla in fisierul task2.c
- va recomandăm mai intai sa rezolvati taskul 1 **și apoi sa faceti copy-paste codului in fisierul task2.c** și să porniti rezolvarea de acolo
- numerele citite de la tastatura sunt considerate pozitive, insa rezultatul poate fi negativ asa ca folositi o variabila de tip **int** pentru a salva rezultatul executarii instrucțiunii.
- afișați doar numărul rezultat, altfel checker-ul nu va lua in considerare testul

## Bonus 1 - Operanzi cu dimensiuni ciudate (20p)

In cadrul acestui task vom extinde implementarea de la Task 2 pentru a executa instrucțiuni ale căror operanzi pot avea **orice** dimensiune din intervalul [1, 16]. În continuare, operanzii vor fi citiți de la intrarea standard sub forma **unsigned short**, însă un operand ar putea să se întindă pe 2 citiri succesive.

Exemplu:

**N = 1, Dim = 11.**  $(1+1)*11/16 = 1,375 \Rightarrow$  se citesc 2 numere de tipul **unsigned short**. Sa presupunem ca acestea sunt 37444 (**10010010010 00100**) si 33792 (**100001 0000000000**). Va trebuie sa facem o operație între 1170 (**10010010010**) si 289 (**00100 100001**)

### Precizări:

- Acest task se va rezolva in fisierul task3.c
- Pentru acest task veti copia continutul fisierul task2.c in fisierul task3.c si veti extinde functionalitatea acestuia.
- Pentru acest task, veti afisa doar rezultatul final.

## Bonus 2 - Precedenta operatorilor (20p)

Implementati precedenta operatorilor, astfel incat \* si / au precedenta egala între ei, insa precedenta mai mare ca + si -.

Exemplu:

Pentru + \* - / si 1 2 3 4 5 vom avea  $1 + 2*3 - 4/5 = 1 + 6 - 0 = 7$

### Precizari:

- Acest task se va rezolva in fisierul task4.c.
- Veti afisa doar rezultatul final.

## Trimitere temă

Tema va fi trimisă folosind `vmchecker`, cursul **Programarea Calculatoarelor (CB & CD)**.

Formatul arhivei va fi următorul:

1. {task1.c task2.c task3.c task4.c Makefile}
2. Un fișier `README` în care vă descrieți rezolvarea taskurilor.

1. Arhiva trebuie să fie de tipul **zip**.

2. Makefile-ul și testele vor fi cele din aceasta arhiva: `miniprosesor.zip`

Nu includeti fisierele checkerului în arhiva voastră. **Nu folosiți Makefile.checker** pe post de Makefile în arhiva voastra: asta va duce la recursivitate infinita pe vmchecker. Puteți să folosiți direct makefile-ul prezent in arhiva (**Makefile, nu Makefile.checker**).

În cazul în care testele vă trec local, însă pică pe vmchecker cel mai probabil aveți o sursă de *"undefined behavior in cod"*. Pentru a va asigura ca scapati de aceste probleme, compilati cu flagul de compilare `-Wall`` și rezolvati toate warning-urile.

## Listă depunctări

Lista nu este exhaustivă.

- o temă care nu compilează și nu a rulat pe **vmchecker** nu va fi luată în considerare
- o temă care nu rezolvă cerința și trece testele prin alte mijloace nu va fi luată în considerare
- [-1.0]: numele variabilelor nu sunt sugestive
- [-1.0]: linii mai lungi de 80 de caractere
- [-5.0]: abordare ineficientă
- [-10.0]: variabile globale
- [-25.0]: Daca se folosesc vectori pentru stocarea bitilor
- [-100.0]: TOT punctajul, în cazul în care se încearcă "obținerea" punctajului pe temă folosind alte metode decât cele normale
  - în cadrul cursului de programare nu avem ca obiectiv rezolvarea în cel mai eficient mod posibil a programelor; totuși, ne dorim ca abordarea să nu fie una ineficientă, de genul să nu folosiți instrucțiuni repetitive acolo unde clar nu era cazul, etc.