TUTORIAT ASC 1

MARIA PREDA, TUDOR MIU

1. Despre Tutoriat

In cadrul acestui tutoriat, la fel ca si in cadrul cursului, vom face o introducere in limbajul Assembly x86, utilizand sintaxa AT&T. De asemenea, ne propunem sa acoperim in fiecare saptamana si probleme similare cu cele de la examen, adica probleme cu: calcule in diferite baze, circuite etc.

2. Compilare si cod masina

Codul pe care îl scriem în limbaje de nivel înalt (precum C, C++, Java, Python) nu poate fi executat direct de către procesorul unui calculator, deoarece acesta înțelege doar codul mașină, care constă din instrucțiuni binare. Astfel, codul de nivel înalt trebuie să fie prelucrat și transformat într-o formă care să poată fi interpretată de mașina de calcul.

În cazul limbajelor compilate (cum ar fi C sau C++), procesul începe cu compilarea, în cadrul căreia codul sursă este transformat într-un cod intermediar, precum cod assembly (un limbaj de nivel jos, aproape de codul mașină). Urmează etapa de asamblare, în care codul assembly este convertit într-un fișier obiect (object file), ce conține instrucțiuni mașină, dar care nu este încă un executabil complet.

După asamblare, urmează linkarea, unde diferitele fișiere obiect sunt combinate, iar legăturile cu bibliotecile externe (cum ar fi funcțiile din bibliotecile standard) sunt realizate. În această etapă, linkerul creează un fișier binar executabil, care conține toate instrucțiunile necesare rulării programului.

În acest punct, fișierul binar generat este complet și poate fi executat direct de procesorul mașinii de calcul. Procesorul poate acum interpreta și executa instrucțiunile binare conținute în fișierul executabil.

Este important de menționat că limbajele interpretate (cum ar fi Python) nu urmează același proces. În cazul acestora, codul sursă este transformat într-o formă intermediară (bytecode), care este apoi interpretată și executată la runtime de un interpretator.

3. Componentele principale

(1) Registrii: Sunt variabila usor si rapid de accesat. Folosing o arhitectura pe 32 de biti, registrii nostri vor avea tot 32 de biti. Dintre acestia, mai importanti sunt:
   (a) EAX: pentru operatii aritmetice
   (b) EBX: pointer la date, adesea folosit pentru a retine adrese de memorie
   (c) ECX: utilizat pentru instructiunile repetitive
   (d) EDX: utilizat pentru pentru operatii aritmetice si citiri/scrieri
   (e) ESI: pointer la sursa in operatii stream
   (f) EDI: pointer la destinatie in operatii stream

(g) ESP: il vom folosi mai tarziu, atunci cand vom lucra cu striva de date, pentru varful acesteia

(h) EBP: tot pentru stiva, pentru baza acesteia

Toti acesti registri pot fi folositi oricum, aceste sunt doar conventii.
Registrii EAX, EBX, ECX, EDX au cate doua componente: AL(primi 8 biti de la least significant) si AH(urmatorii 8 biti), BH si BL etc.
Pentru utilizarea registrilor vom pune intotdeauna "%"

(2) Instructiuni: Trebuie sa putem muta date intre registri, intre regiistri si memorie, sa facem calcule, citiri/scrieri de date.

(3) Flaguri: Dupa efectuarea unor operatii, vrem sa stim care a fost rezultatul acesteia si eventualele erori aparute. Flagurile sunt coduri care semnifica aparitia unui anumit caz dupa efectuarea operatiilor, de exemplu overflow.

(4) Adrese de memorie: Pe procesor putem stoca un numar mic de valori. Pentru a executa operatii complexe este nevoie sa ne referim la valori din memoria calculatoruli. Lucram cu adrese din memoria RAM (vom folosi termenul "memorie" pentru a ne referi la aceasta). Pentru a identifica locul la care se afla o valoare in memorie vom folosi una dintre urmatoarele variante:
(a) valoarea unui registru pe 32 de biti: (%eax);

(b) valoarea unui registru + un offset numeric: 4(%eax), adica %eax + 4;

(c) valoarea unui registru + valoarea unui registru: (%eax, %edx), adica %eax + %edx;

(d) suma adoi registrii + un offset numeric: 4(%eax, %edx), adica %eax + %edx + 4;

(e) valoarea unui registru inmultita cu 2,4 sau 8 + valoarea unui registru + o constanta (constanta poate sa lipseasca): 16(%eax, %edx, 4), adica %eax + 4 * %edx + 16;

(5) Stiva programului: Este o parte din memorie, alocata pentru fiecare program care este executat. Aceasta este o structura de tip LIFO. Ultimul element introdus in stiva, va fi primul element eliminat cu ajutorul functiei pop().

(6) Intreruperi: Un program nu are acces direct la toate resursele. De exemplu, nu are acces direct la memoria reala, folosind o memorie virtuala, cu adrese "false" de memorie, pe care sistemul de operare le asociaza printr-un tabel cu cele reale. Asadar, in multe cazuri, avem nevoie sa facem un apel catre sistemul de operare care sa faca diferite operatii pentru noi. Momentele in care facem cereri catre sistemul de operare sunt numite intreruperi.
(7) Constantele numerice: Acestea sunt valori utilizate in program si trebuie precedate de simbolul "$". Exemplu: $19.

4. Tipuri de date

La fel ca si in limbajele de nivel inalt studiate pana acum, exista mai multe tipuri de date:

Denumire	Spațiu ocupat	Utilitate
`byte`	1 octet (8 biți)	Stochează valori numerice între 0 și 255 (fără semn)
`single`	4 octeți (32 biți)	Reprezintă un număr în virgulă mobilă pe 32 de biți
`word`	2 octeți (16 biți)	Stochează valori numerice între 0 și 65535 (fără semn)
`long`	4 octeți (32 biți)	Stochează valori numerice între -2^{31} și $2^{31}-1$ (cu semn)
`quad`	8 octeți (64 biți)	Stochează valori numerice între -2^{63} și $2^{63}-1$ (cu semn)
`ascii`	1 octet per caracter	Stochează caractere ASCII
`asciz`	1 octet per caracter, terminat cu `0x00`	Stochează șiruri de caractere ASCII terminate cu `0x00`
`space`	Variabil, în funcție de specificație	Rezervează un anumit număr de octeți în memorie pentru date sau alocări viitoare

5. Structura unui program

Un program Assembly x86 va urma urmatorul schelet:

```
.data
    // declarari de date

.text
    // in aceasta zona declaram proceduri.
    // Fiecare procedura va avea o eticheta proprie
    // prin intermediul careia o vom "apela"

.global main

main:
    // instructiuni
```

6. Instructiuni

6.1. **Instructiunea MOV.** Sintaxa: mov sursa, destinatie.

Aste folosita pentru a muta date intre registri, din registri in memorie, din memorie in registri sau pentru a pune o constanta intr-un registru.

Observaţie 6.1. *Nu se pot pune constante direct in memorie.*

Observaţie 6.2. *Instructiunea poate fi sufixata pentru a indica tipul de date folosit. In acest caz, poate fi folosita si pentru a pune direct constante in memorie.*

6.2. **Instructiunile ADD si SUB.** *Sintaxa: add operator1, operator2*

sub operator1, operator2

Acestea functioneaza in mod similar, prima pentru a realiza o adunare, iar a doua pentru scadere. Acestea se interpreteaza ca fiind "operator2 operatie operator1".

Exemple:

```
mov $7, %eax
mov $5, %ebx
add %eax, %ebx
```

Valoarea din registrul %ebx va fi, in acest caz, 12.

```
mov $7, %eax
mov $5, %ebx
sub %ebx, %eax
```

Dupa efectuarea acestei bucati de cod, valoarea din %ebx ramane 5*, iar in registrul %eax vom aveam* 2.

6.3. **Intreruperi de sistem.** *Asa cum spuneam mai sus, avem de multe ori nevoie sa facem un apel catre sistemul de operare, lucru pe care il putem face folosind linia* ***int $0x80****.*

Un exemplu pe care il vom folosi in fiecare program pe care il scriem este SYSTEM EXIT. Acest lucru se face folosind functia exit(0). Cu toate acestea, nu putem apela aceasta functie exact sub forma aceasta. In schimb, trebuie sa punem parametrii functiei in registri. In registrul %eax trebuie sa punem codul functiei, care este 1*, iar in registrul %ebx punem argumentul functiei, in cazul nostru:* 0*. Astfel, apelul va arata asa:*

```
mov $1, %eax
mov $0, %ebx
int $0x80
```

7. Schimbarea bazei de numeratie

Cu toate ca singura baza pe care calculatorul o "intelege", exista situatii in care este util sa putem transforma rapid si in alte baze. De exemplu, baza 16 (hexazecimal) este utilizata pentru adresele mac, acestea fiind o insiruire de 12 caractere in baza 16. De asemenea, vom folosi baza 16 *pentru adresele de memorie.*

Baza 10 (zecimal) este baza pe care noi o folosim in mod curent. Istoric vorbind, aceasta este o conventie bazata pe numarul de degete de la maini. Nu a fost prima sau singura incercare de a stabili o baza. In acest sens, istoria numararii este prezentata mai bine in cartea "1+1 nu face (intotdeauna) 2" de John D. Barrow.

Numarul bazei corespunde numarului de "cifre" din acea baza. Odata ce depasim aceste cifre, se mai adauga, exact ca si in matematica cu care am fost obisnuiti, inca o cifra. In mod generic, pentru a calcula echivalentul in baza 10 a unui numar dintr-o baza oarecare N*, trebuie sa inmultim fiecare dintre cifre cu* $N^{nr\ cifre\ de\ la\ dreapta\ ei}$*, iar apoi sa facem suma.*

7.1. **Transformare intre bazele 10 si 2.** *In binar, avem doar doua cifre:* 0 *si* 1*. Bitii sunt in sistem binar. Pentru a transforma din baza 2 in baza 10, luam fiecare bit, il inmultim cu* $2^{nr\ biti\ de\ la\ dreapta\ bituluicurent}$*. Astfel, avand doar* 0 *si* 1*, putem sa inmultim doar bitii care sunt* 1*.*

In general, avem:

$numar_{(2)} = bit_n\ bit_{n-1}\ \cdots\ bit_3\ bit_2\ bit_1 \rightarrow numar_{(10)} = bit_n * 2^{n-1} + bit_{n-1} * 2^{n-2} + \cdots + bit_2 * 2^1 + bit_1 * 2^0$

Deci:

$$numar_{(2)} = bit_n\ bit_{n-1}\ \cdots\ bit_3\ bit_2\ bit_1 \rightarrow numar_{(10)} = \sum_{i=1}^{n} bit_i \cdot 2^{i-1}$$

In sens invers, daca dorim sa efectuam o transformare din baza 10 *in baza* 2*. Exista mai multe metode de a face asta. Prima, cea folosita si in liceu/gimanziu, este cea prin care impartim in mod repetat la* 2*, obtinem resturile de la fiecare impartire si le scriem de la coada spre inceput intr-un sir de biti.*

Exemplu: Transformarea lui $84_{(10)}$ *in baza* 2*:*

$$84 : 2 = 42\ rest\ 0$$
$$42 : 2 = 21\ rest\ 0$$
$$21 : 2 = 10\ rest\ 1$$

$$10 : 2 = 5 \; rest \; 0$$
$$5 : 2 = 2 \; rest \; 1$$
$$2 : 2 = 1 \; rest \; 0$$
$$1 : 2 = 0 \; rest \; 1$$
$$\Rightarrow 84_{(10)} = 1010100_{(2)}$$

Observaţie 7.1. *Daca facem impartirile pana cand avem catul* 0*, atunci luam doar resturile, altfel, trebuie sa consideram ultimul cat, adica* 1*.*

O alta metoda, ar fi sa gasim cea mai mare putere a lui 2 *care sa fie mai mica decat numarul, apoi, pentru numarul ramas pe care il mai avem de "acoperit", sa gasim noua cea mai mare putere a lui* 2 *care sa fie mai mica decat numarul ramas. Vom pune* 1 *pe pozitiile cu* 1 *mai mari decat exponentul, numarate de la dreapta la stanga, iar in rest* 0*. Lucrand mai mult cu aceste transformari, metoda aceasta devin mult mai rapida decat cea precedenta.*

Exemplu: Tot transformarea lui $84_{(10)}$ *in baza* 2*:*

Observam ca cea mai mare putere a lui 2 *mai mica decat* 84 *este* $2^6 = 64$*, deci primul bit nenul de la dreapta la stanga va fi al 7-lea. Mai trebuie sa obtinem o suma de puteri ale lui doi egala cu* 20*. Noua cea mai mare putere posibila este* $2^4 = 16$*, deci vom avea bitul* 5 *de la dreapta la stanga egal cu* 1*, iar bitul* 6 *egal cu* 0 *etc.*

Indiferent de metoda aleasa, rezultatul va fi acelasi.

Observaţie 7.2. *Pentru transformarea din baza* 10 *in oricare alta baza sau invers, se procedeaza similar, doar ca vor fi folosite puteri ale acelei baze.*

7.2. **Transformarea dintr-o baza N intr-o baza putere a lui N.** *Sa luam, de exemplu, schimbarea unui numar scris in baza* 2 *in baza* 16*. Cum spuneam mai devreme, in baza* 16 *vom aveam* 16 *cifre. Deaorece, in mod traditional, nu dispunem de atatea, vom folosi, in ordine, literele de la* A *la* F*. Astfel,* F *va fi practic a 16-a cifra, adica un echivalent al lui* 15 *din baza* 10*.*

Pentru a face o transformare rapida din baza 2 *in baza* 16*, putem grupa, de la dreapta la stanga, cate* 4 *biti, apoi sa le gasim echivalentul unui numar din baza* 10*, evident, un numar intre* 0 *si* 15*, dupa care vom pune "cifra" corespunzatoare din baza* 16*. Pentru transformarea inversa, pentru fiecare cifra din baza* 16 *va fi scrisa drept reprezentarea sa binara pe* 4 *biti.*

Observaţie 7.3. *Pentru transformare intre bazele* 2 *si* 8*, se grupeaza cate* 3 *cifre etc.*

De ce functioneaza aceasta metoda? (Optional)

Pe 4 *biti din baza* 2*, putem scrie maxim numarul* 15 *din baza* 10*. Sa luam, de exemplu, numarul* 10101110 *in baza* 2*. Acesta s-ar scrie, pentru a fi transformat in baza* 10 *ca:*

$$2^7 + 0 \cdot 2^6 + 2^5 + 0 \cdot 2^4 + 2^3 + 2^2 + 2^1 + 0 = 128 + 32 + 8 + 4 + 2 = 174_{(10)}$$

Daca am grupa cate 4 *cifre, am avea:*

$$(2^7 + 0 \cdot 2^6 + 2^5 + 0 \cdot 2^4) + (2^3 + 2^2 + 2^1 + 0) = 2^4 \cdot (2^3 + 2^1) + (2^3 + 2^2 + 2^1) = 16^1 * 10 + 14$$

Asa cum am mentionat anterior, pentru a transforma un numar dintr-o baza in baza 10*, inmultim fiecare cifra a lui cu baza sa la puterea cate cifre avem la dreapta.*

Asadar, observam ca, dand factor comun pe perechi de cate 4 *cifre, vom obtine puteri ale lui 16 inmultite cu "cifre" din baza* 16*, ceea ce demonstreaza corectitudinea metodei de grupare a cifrelor. Aceasta metoda este mult mai rapida decat trecerea prin baza intermediara* 10*.*

8. Exercitii

8.1. **Transformari intre baze.**

(1) *Efectuati urmatoarele transformari:*

(a) *Avem numarul* $0XBEEF$ *in baza* 16*, transformati-l in bazele* 2*,* 4*,* 8 *si* 10*.*

Soluţie:
Pentru baza 2***:***
Fiecare cifra se transforma intr-o secventa echivalenta de 4 *cifre din baza* 2*:*

$$B = 1011$$
$$E = 1110$$
$$F = 1111$$

Asadar, numarul va fi in baza 2*:* 1011111011101111*.*
Pentru baza 4***:***
Grupa cifrele numarului scris in baza 2 *cate doua, de la dreapta la stanga, dupa care vedem care este cifra echivalenta:*

$$11 = 3$$
$$10 = 2$$

Numarul in baza 4 *va fi:* 23323233*.*
Pentru baza 8***:***
Grupa cifrele numarului scris in baza 2 *cate trei, de la dreapta la stanga, dupa care vedem care este cifra echivalenta:*

$$111 = 7$$
$$101 = 5$$
$$011 = 3$$
$$1 = 1$$

Numarul in baza 8 *va fi:* 137357*.*
Pentru baza 10***:***
Putem transforma din oricare din bazele in care avem pana acum numarul in baza 10*. Deoarece am mai oferit in sectiunea de teorie un exemplu de transformare din baza* 2 *in baza* 10*, de data aceasta vom transforma din baza* 16*:*

$$0XBEEF_{(16)} -> 11 \cdot 16^3 + 14 \cdot 16^2 + 14 \cdot 16^1 + 15 = 45056 + 3584 + 224 + 15 = 48879_{(10)}$$

(b) *Avem numarul* 1427 *in baza* 8*, transformati-l in bazele* 2*,* 4*,* 10 *si* 16*.*

In baza 2*:* 1100010111*. De ce nu avem un numar de biti divizibil cu* 3*? Cea mai din stanga cifra,* 1*, se reprezinta cu* 3 *cifre din baza* 2 *ca fiind* 001*, dar, fiind primele cifre, nu este nevoie de primele doua zerouri.*
In baza 4*:* 120113*.*
In baza 10*: Aici suntem nevoiti sa facem calcule, nu mai putem folosi grupari de cifre.*

$$2^9 + 2^8 + 2^4 + 2^2 + 2 + 1 = 512 + 256 + 16 + 4 + 2 + 1 = 791$$

In baza 16*:* 317

(c) *Avem numarul* 100123 *in baza* 4*, transformanti-l in bazele* 2*,* 8*,* 10 *si* 16*.*
In baza 2*:* 10000011011*.*
In baza 8*:* 2033*.*
In baza 10*:* 1051*.*
In baza 16*:* $0X41B0$*.*

(d) *Avem numarul* 876 *in baza* 10*, transformati-l in bazele* 2*,* 4*,* 8 *si* 16*.*