

ARHITECTURA SISTEMELOR DE CALCUL

– Seminar 6 –

1. Proceduri scrise în asamblare (subprograme)
2. Programe din mai multe module

1. PROCEDURI SCRISE ÎN ASAMBLARE (SUBPROGRAME)

Limbajul de asamblare nu recunoaște noțiunea de *subprogram*. Putem însă să creăm o secvență de instrucțiuni, care să poată fi apelată din alte zone ale programului și care, la finalul execuției, să returneze controlul programului apelant. O astfel de secvență se numește *procedură*.

1.a. Apelul unei proceduri scrise în asamblare

Apelul unei proceduri se poate realiza, teoretic, și printr-o instrucțiune de salt necondiționat **JMP**. Problema care apare în acest caz este că procesorul nu știe de unde va trebui să reia execuția programului după terminarea procedurii.

De aceea, pe timpul apelului unei proceduri, este necesar să salvăm undeva offsetul instrucțiunii de la care se va relua execuția programului. Locul unde se salvează offsetul de revenire este stiva de execuție.

Apelul unei proceduri scrise în asamblare se realizează cu ajutorul instrucțiunii **CALL** astfel:

call nume_procedura

Instrucțiunea **CALL** execută un „salt cu revenire”, prin urmare, execuția sa va produce următoarele efecte:

- se pune în vârful stivei offset-ul instrucțiunii imediat următoare (valoarea curentă din registrul EIP);
- se execută un salt de tip NEAR la eticheta **nume_procedura** (la începutul secvenței de instrucțiuni).

După execuția secvenței de instrucțiuni proprii, orice procedură trebuie să transfere controlul programului apelant. De aceea, în orice procedură, ultima instrucțiune este, în mod obligatoriu, instrucțiunea **RET**.

Execuția instrucțiunii **RET** va produce următoarele efecte:

- se extrage offset-ul de revenire din vârful stivei;
- se execută un salt de tip NEAR la offset-ul extras anterior din stivă.

Exemplu

Considerăm următorul program scris în limbajul C:

```
...
// definitia functiei 1
void functia_1() {
    ...           // liniile de cod care compun corpul functiei
}

// definitia functiei 2
int functia_2() {
    ...           // liniile de cod care compun corpul functiei
    return x;
}

// programul principal
int main(int argc, char* argv[]) {
    ...
```

```

functia_1();          // apelez functia 1
int r = functia_2();  // apelez functia 2
...
return 0;
}

```

În limbaj de asamblare, programul arată astfel:

```

...

segment code use32 class=code

; programul principal
start:

    ...          ; transmit argumentele
    call functia_1 ; apelez functia_1
    ...          ; eliberez stiva

    ...          ; transmit argumentele
    call functia_2 ; apelez functia_2
    ...          ; eliberez stiva
    ...          ; stochez valoarea returnata

    push dword 0
    call [exit]

; definitia functiei1
functia_1:
    ...          ; secventa de instructiuni
    ret          ; revin in programul principal

; definitia functiei 2
functia_2:
    ...          ; secventa de instructiuni
    ret          ; revin in programul principal

```

1.b. Modalități de transmitere a argumentelor și de returnare a rezultatului

Atunci când apelăm în limbaj de asamblare o funcție din librăria C (librăria de funcții **msvcrt.dll**) este obligatoriu să respectăm convenția de apel **_cdecl**. În cazul procedurilor scrise în asamblare, nu mai suntem constrânși să respectăm o anumită convenție de apel.

Cel care definește procedura este obligat să stabilească:

- modul și ordinea de transmitere a argumentelor funcției;
- cum va fi returnat rezultatul execuției;
- care sunt regiștrii care pot fi modificați pe timpul execuției;
- cine are obligația să „șteargă” argumentele (în cazul în care acestea sunt puse pe stivă).

Transmiterea argumentelor unei proceduri scrise în asamblare se poate face în regiștri sau pe stivă.

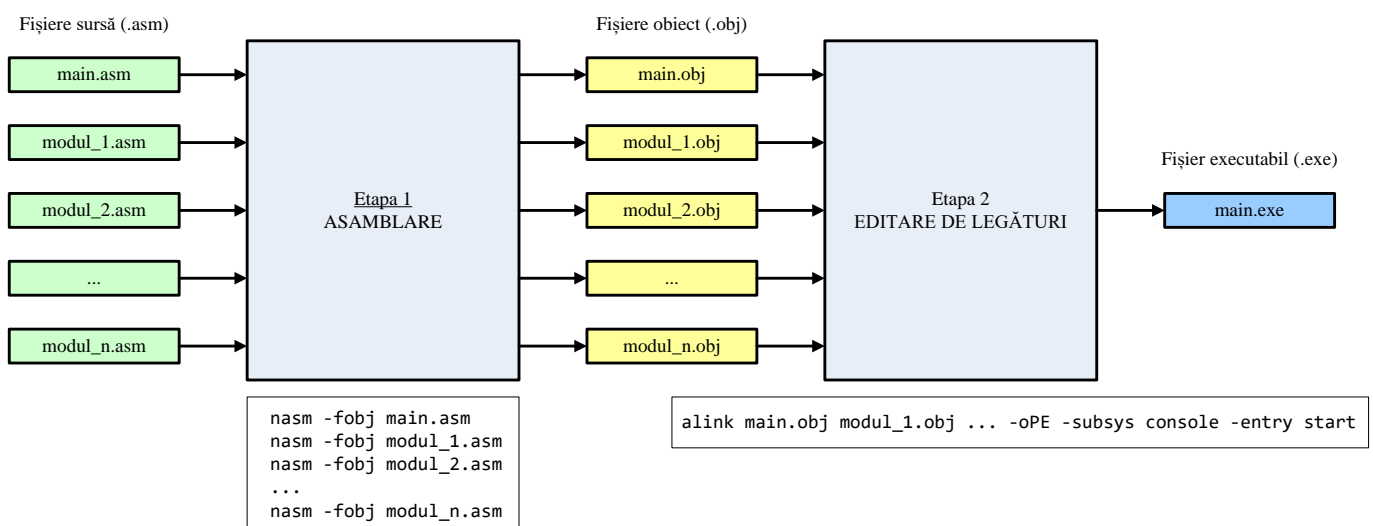
Procedura poate să returneze rezultatul într-un registru sau pe stivă.

2. PROGRAME DIN MAI MULTE MODULE

Un program scris în limbaj de asamblare poate fi împărțit în mai multe fișiere sursă (*module*), fiecare fiind asamblat separat în *fișiere obiect* (.obj).

Pentru scrierea unui program compus din mai multe module trebuie să respectăm următoarele reguli:

- toate segmentele vor fi declarate cu specificatorul **public**, pentru că în programul final segmentul de cod va fi construit prin concatenarea segmentelor de cod din fiecare modul (la fel și segmentul de date);
- etichetele și numele variabilelor declarate într-un modul și care trebuie „exportate” în alte module trebuie să declare folosind directiva **global**;
- etichetele și variabilele care sunt declarate într-un modul și sunt folosite în alt modul trebuie să fie „importate” prin directiva **extern**;
- o variabilă trebuie declarată în întregime într-un modul (nu poate fi jumătate într-un modul și jumătate într-altul);
- transferul controlului execuției dintr-un modul în altul se face cu instrucțiunile: **jmp**, **call** sau **ret**;
- punctul de intrare în program (**global start**) trebuie declarat doar în modulul care conține „programul principal”.



Construcția unui program din mai multe module se realizează în două etape:

1. etapa de asamblare:

- se realizează cu ajutorul unui program numit *asamblor*;
- fiecare fișier sursă (modul) va fi asamblat folosind asamblorul *NASM (The Netwide Assembler)*:

```
nasm -fobj modul_1.asm
nasm -fobj modul_2.asm
...
nasm -fobj modul_n.asm
```

- în urma procesului de asamblare se va genera câte un fișier obiect (.obj): `modul_1.obj`, `modul_2.obj`, ..., `modul_n.obj`

2. etapa de editare de legături (linkeditare):

- se realizează cu ajutorul unui program numit *editor de legături (linkeditor)*;
- toate fișierele obiect (.obj) rezultate în urma asamblării vor fi concatenate într-un singur fișier executabil (.exe):

```
alink modul_1.obj modul_2.obj ... modul_n.obj -oPE -subsys console -entry start
```

La finalul procesului de construcție, se obține un singur fișier executabil pe 32 de biți (.exe) care poate fi rulat de către sistemul de operare Windows.

EXERCITII

1. Scrieți un program care calculează valoarea expresiei $x = a + b$.
2. Scrieți un program care calculează factorialul unui număr natural n citit de la tastatură.
3. Scrieți un program care concatenează două șiruri de caractere citite de la tastatură.
4. Se dau 2 șiruri de caractere. Sa se afișeze șirul cu număr maxim de caractere speciale (orice în afara de litere și cifre)