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);

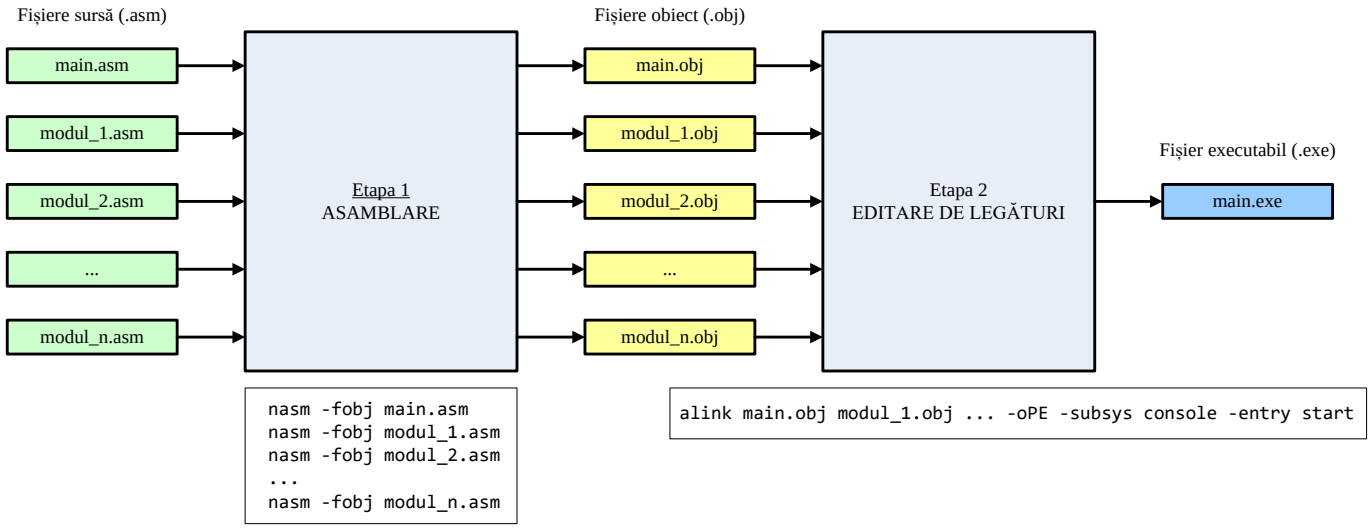
– etichetele și numele variabilelor declarate într-un modul și care trebuie *„exportate”* în alte module trebuie să declarate folosind directiva **`global`**;

– 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.

EXERCIȚII

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 siruri de caractere. Sa se afiseze sirul cu numar maxim de caractere speciale (orice in afara de litere si cifre)

Problema 1 – 3p – 26.01.2023

Se dă un șir ASCIIZ de secvențe de litere separate prin caracterul spațiu în modulul **a.asm**. Scrieţi un program multimodul (asm+asm) care determină și afișează pe ecran secvențele de litere care au număr par de vocale și în același timp număr impar de consoane, și tipărește pe ecran în baza 16 numărul acestor secvențe, fără a utiliza specificatorii %x, %X. În acest scop, programul va apela o funcție denumită **procesare** definită în modulul **b.asm**, care primește ca parametru adresa de început a secvenței curente de prelucrat si verifică dacă secvența îndeplinește condițiile pentru afișare. Nu se va folosi nici o funcție de prelucrare de șiruri a limbajului C. **Explicaţi şi detaliaţi abordarea algoritmului şi mecanismele implicate. Justificaţi şi comentaţi corespunzător textul sursă. Prezentaţi şi explicaţi mecanismele de comunicare utilizate între cele două module dezvoltate.**

Exemplu: sir db 'Ana are mere Ada are fructul pasiunii Gigel are ananas Tudor are portocale si facem salata de fructe', 0
Numărul de secvente de litere cu număr par de vocale și număr impar de consoane este 11 = 0Bh deci pe ecran se va afisa:
Ana are Ada are fructul Gigel are Tudor are portocale facem
B

Main.asm

0.25p - algoritm descriere in cuvinte
0.25p - schelet program + segment de date +comentarii
0.25p - parcurgere sir mare de litere (se puncteaza inclusiv faptul ca dupa o secventa de litere continua corect cu urmatoarea)
0.1p – are conditie de oprire si trateaza corect si ultima secventa care se termina cu 0
0.15p - apel corect functie procesare (cu parametrul corect)
0.25p - verificare daca secventa curenta respecta conditiile in functie de rezultatul functiei (e la alegerea lor ce si cum returneaza, dar trebuie sa verifice undeva nr par de vocale si nr impar de consoane)
Obs. E ok si daca fac verificarea in functie si returneaza 1 sau 0 direct. S-a lasat peste tot la libertatea lor cum impart codul intre main si functie.
0.3p - Afisare secventa de litere daca respecta conditia (de ex inlocuiesc spatial cu 0, sau copiaza in alta parte sirul, sau orice alta metoda)
0.1p - calcul numar total de secvente cu conditia data
Afisare in baza 16 (total 0.65p):
0.2p - bucla corecta pt aflarea cifrelor si stocarea lor (pe stiva sau intr-un sir)
0.2p - convertirea cifrelor in litere unde e cazul (cu xlat sau manual)
0.15p - printf cu %c (nu le-am dat voie sa foloseasca %x)(Daca fac printf cu %d fara sa verifice ca e mai mic ca 10 primesc doar 0.05p)
0.1p - salvare ECX inainte de apel printf pt a putea parcurge "cifrele" in continuare (primesc punctele si daca nu au folosit ECX, ci alta metoda de iterare)

Modul 2:

0.2p - descriere mecanisme (transmitere procedura (stiva, registrii, variabile))
0.25p - parcurgere secventa pana la spatiu sau pana la 0
0.25p - identificare consoane si vocale

Problema 2

Să se scrie un program în limbaj de asamblare care primește la intrare un șir de dublucuvinte definit în segmentul de date. Programul va forma un nou sir prin extragerea octetului superior al cuvântului inferior, urmată de extragerea octetului inferior al cuvântului superior din fiecare dublucuvânt si va forma cu aceștia un șir pe care îl va stoca în memorie. Programul ca determina toti octeții care au valori pozitive și pare, sunt diferiți ca valoare și va calcula suma lor. La final, programul va afișa în baza 16 cei k octeți obținuți și suma acestora.

Programul va fi format din două module: modulul „a.asm”, care conține programul principal și modulul „b.asm”, care conține o procedură numită „selectie”. Procedura va primi ca argumente offset-ul șirului de octeți, va determina octeții care îndeplinesc condițiile impuse și va returna suma acestora.

Explicați și detaliați abordarea algoritmului și mecanismele implicate. Justificați și comentați corespunzător textul sursă.

Exemplu:

Șirul de dublucuvinte: 630C018Fh, 640E0563h, 6102DF07h, 6202CF00h, 6506BF02h

Șirul de octeți extrași: 01h, 0Ch, 05h, 0Eh, 0DFh, 02h, 0CFh, 02h, 0BFh, 06h

Șirul de octeți pozitivi: 01h, 0Ch, 05h, 0Eh, 02h, 02h, 06h

Șirul de octeți pari si pozitivi: 0Ch, 0Eh, 02h, 06h

Suma octetilor identificati: 22h