**RULARE**

Pentru rularea unui program multimodul:

Pas 1: Adaugare in program ALINK.exe si NASM.exe (in nasm din asmtools)

Pas 2: Rezolvare probleme (scrii cod)

Pas 3: Pozitionare in director in Command Prompt: cd \_\_PATH\_\_

Pas 4: **Comenzi**:

nasm -f obj main.asm

nasm -f obj second.asm

alink main.obj second.obj -oPE -subsys console -entry start

main.exe

* main e numele fisierului principal si second numele fisierului secundar

**REGULI**

* Sa nu fie numita a doua functi ca si numele fisierului (da eroare la link-editare)
* Cand apelam functia sa se scrie fara paranteze drepte []: call functie nu call [functie]
* Cand apelam functii normale se scriu intre paranteze drepte []

**CUM SE SCRIE DE LA 0**:

* In modulul principal trebuie declarant in **SEGMENTUL DE DATE** eticheta de unde porneste al doilea modul
* In modulul secundar trebuie declarata in **SEGMENTUL DE DATE** eticheta de unde porneste modulul ca fiind globala

**TEORIE MULTIMODUL**

Apelul unei procedure se face cu ajutorul instructiunii CALL, care poate fi apel direct sau indirect

**CALL operand**

* Transfera controlul la adresa desemnata de operad, dar si salveaza in stiva adreasa urmatoarei instructiuni dupa CALL
* Echivalent cu:
  + Push Urmatoare
  + Jmp operand
  + Urmatoare:

**RET [n]**

* Terminarea executiei secventei apelate este marcata de intalnirea unei instructiuni RET
* Acesta preia din stiva adresa de revenire depusa acolo de CALL, preluand controlul la instructiunea de la aceasta adresa
* Parametrul optional n indica cati octeti o sa fie eliberiati de pe stiva
* Echivalent cu:
  + Pop [B] ; In B se va afla astfel adresa de revenire, unde B e o eticheta din memorie (doubleword)
  + Add ESP,[n] ; OPTIONAL
  + Jmp [B]

**COD APEL**: (Cel care apeleaza functia)

* Salvare resurse volatile
* Transmitere paramentrii
* Efectuare apel cu salvare adresa de revenire (adica call sau varianta echivalenta)

**COD DE INTRARE**: (Adica inceputul codului din modulul apelat)

* Creare stackframe nou: (Pas necesar pentru programarea ASM+C)
  + Push EBP
  + Mov EBP,ESP
* Alocare spatiu de variabile locale SUB ESP, nr\_octeti (Pas necesar pentru programarea ASM+C)
* Salvare resurse nevolatile (registrii care nu tin implicit de apel)

**COD DE IESIRE**: (Adica sfarsitul codului din modulul apelat)

* Restaurare registrii nevolatili
* Eliberare spatiu de variabile locale: ADD ESP, nr\_octeti (Pas necesar pentru programarea ASM+C)
* Eliberare cadru stiva: (Pas necesar pentru ASM+C)
  + Mov esp, ebp
  + Pop ebp
* Revenire din subprogram (RET) si scoaterea de pe stiva a parametriilor (asta in programul principal)

**OBSERVATII**:

* Pentru functia: **functie(op1,op2,op3)**
  + Cod apelant:
    - Pushad; Aici salvam toti registrii (daca se folosesc in functie aka resurse volatile) (nu in cazul asta)
    - Push dword op3
    - Push dword op2
    - Push dword op1
    - Call functie
    - Add esp,4\*3
    - Popad; Aici restauram registrii (daca se folosesc in functie aka restaurare resurse volatile) (nu in cazul asta)
  + Cod de intrare:
    - ; Salvare registrii folositi daca urmeaza sa fie folositi (si nu sunt explicit in parametrii functiei)
    - ; [ESP+0] este adresa de revenire
    - ; [ESP+4] este op1
    - ; [ESP+8] este op2
    - ; [ESP+12] este op3
    - …
    - ;Restaurare registrii folositi daca au fost folositi (si nu sunt explicit in parametrii functiei)
    - Ret
  + Aici nu avem nici un registru resursa VOLATILA, deci toti registrii sunt resurse NEVOLATILE, deci daca vrem sa ii folosim in subprogram trebuie salvati in SUBPROGRAM
  + Resursele VOLATILE pot fi si op1, op2, op3 daca ele sunt operanzi cu adresare la memorie, deci daca avem nevoie de valoarea lor de dinainte de apel, datoria salvarii este in programul PRINCIPAL
* Valoriile tuturor registriilor sunt transmise de la un modul la altul, deci valoarea de baza a registriilor cand au intrat in subprogram va fi valoarea lor inainte de apelul CALL, in programul principal
* Parametrii functiei pot fi transmisi:
  + Prin registrii (METODA NEINDICATA)
  + Prin stiva (METODA INDICATA) -> Cea pe care o folosim
* Daca parametrii sunt transmisi prin registrii, atunci nu mai ii punem pe stiva, si nu mai eliberam stiva la final
  + Daca parametrii unei functii sunt eax, si op1 vom avea:
    - Push op1
    - Call functie
    - Add esp,4
  + Registrii sunt transmisi automat in subprogram
  + Aici EAX este o resursa VOLATILA (deci daca o sa mai avem nevoie de valoarea lui initiala, trebuie salvat si restaurant in programul PRINCIPAL), iar restul registriilor sunt resurse NEVOLATILE, deci daca vrem sa ii folosim in subprogram, datoria salvarii si restaurarii lor, este in subprogram (EAX poate fi modificat fara a fi salvat deoarece e resursa VOLATILA, si nu e datoria subprogramului sa o salveze)