# C + assembler

De citit [Dandamudi]:
Capitole 17

Modificat: 22-Oct-23

# Cuprins

- De ce programe mixte?
  - **Focus pe C si limbaj de asamblare**
- Compilarea programelor mixte
- Apel limbaj de asamblare din C
  - Transmiterea parametrilor
  - Valori de retur
  - Pastrarea valorilor din registre
  - Global si external
- Exemple
- Apeluri C din limbaj de asamblare

# De ce programe mixte?

- Avantaje si dezavantaje ale limbajului de asamblare
  - Avantaje:
    - Acces la operatii low-level
    - Performanta
    - Control asupra programului
  - Dezavantaje:
    - Productivitate scazuta
    - Greu de asigurat mentenanta
    - Lipsa de portabilitate
- Prin urmare, unele programe sunt mixte (system software)

# Compilarea programelor mixte

- Putem folosi programare mixta in C si limbaj de asamblare
- Vom pune accentul pe principii
- Acestea pot fi generalizate la orice tip de programare mixta
- Pentru compilare:

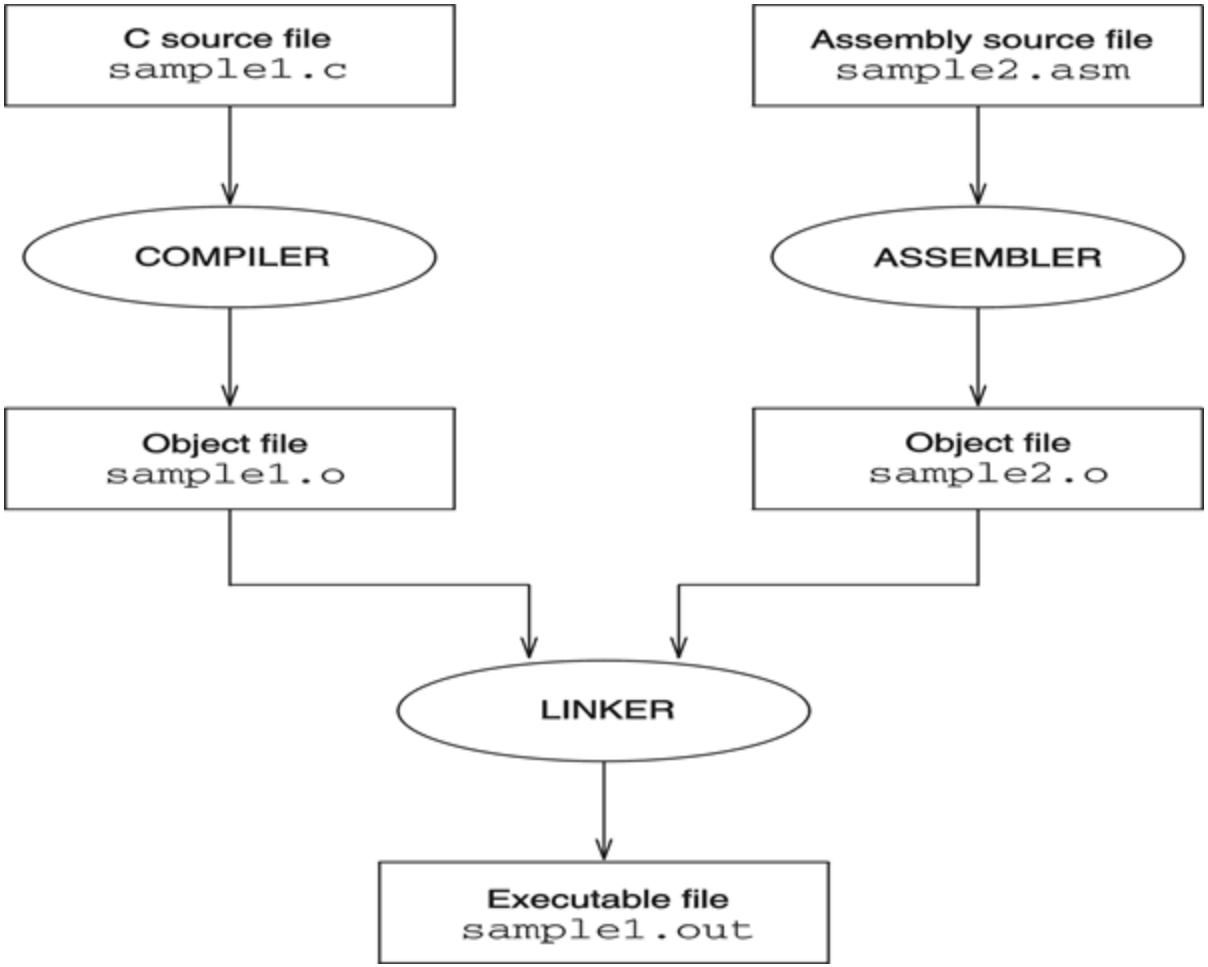
```
nasm -f elf sample2.asm
```

  - » creeaza **sample2.o**

```
gcc -o sample1.out sample1.c sample2.o
```

  - » creeaza **sample1.out** fisier executabil

# Obținerea unui executabil mixt



# Apel limbaj de asamblare

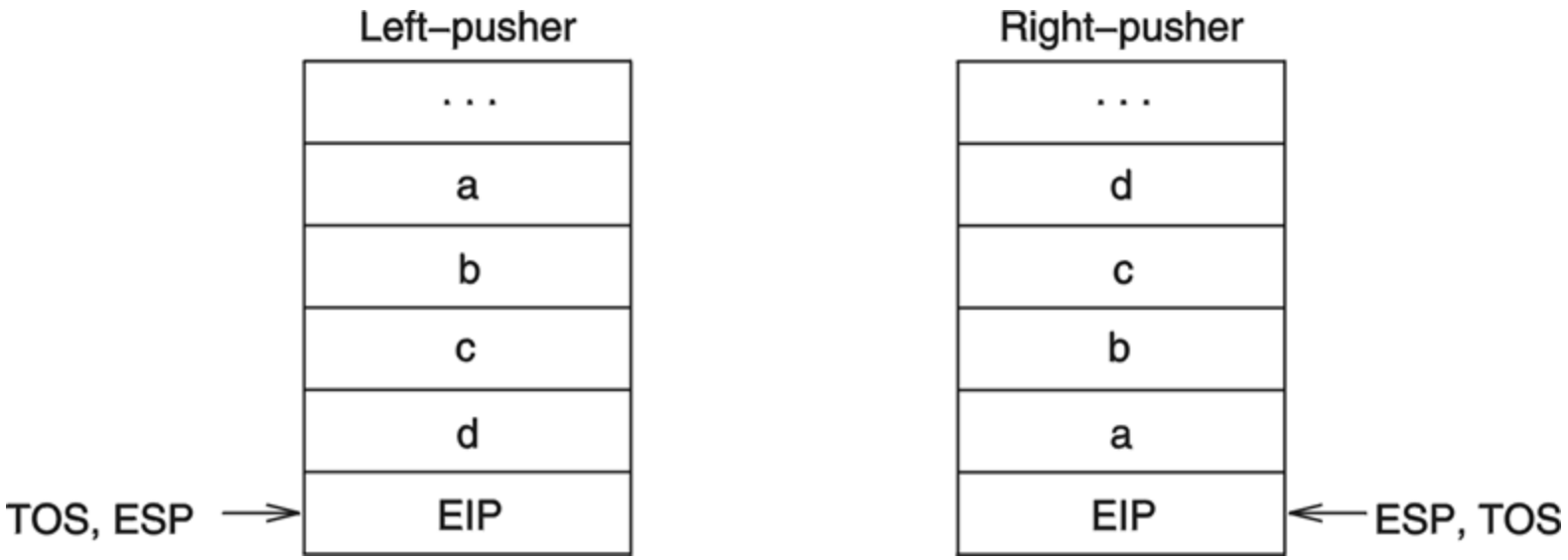
## Transmiterea parametrilor

- Stiva este folosita pentru transmiterea parametrilor
- Doua modalitati de a pune parametrii pe stiva
  - Left-to-right
    - » Majoritatea limbajelor inclusiv Basic, Fortran, Pascal folosesc aceasta metoda
    - » Aceste limbaje se numesc limbaje *left-pusher*
  - Right-to-left
    - » **C foloseste aceasta metoda**
    - » Aceste limbaje se numesc limbaje *right-pusher*

# Apel limbaj de asamblare din C

Exemplu:

```
sum(a,b,c,d)
```



# Apel limbaj de asamblare din C

## Valoare de retur

- Registrele folosite pentru valori de retur

| Tip valoare return | Registru folosit |
|---|---|
| 8-, 16-, 32-bit value | EAX |
| 64-bit value | EDX:EAX |

# Apel limbaj de asamblare din C

## Pastrarea valorilor registrelor

- Valorile urmatoarelor registre trebuie pastrate

  EBP, EBX, ESI, si EDI

- Alte registre
  - Daca este necesar, valorile pot fi pastrate de catre functia apelanta

# Apel limbaj de asamblare din C

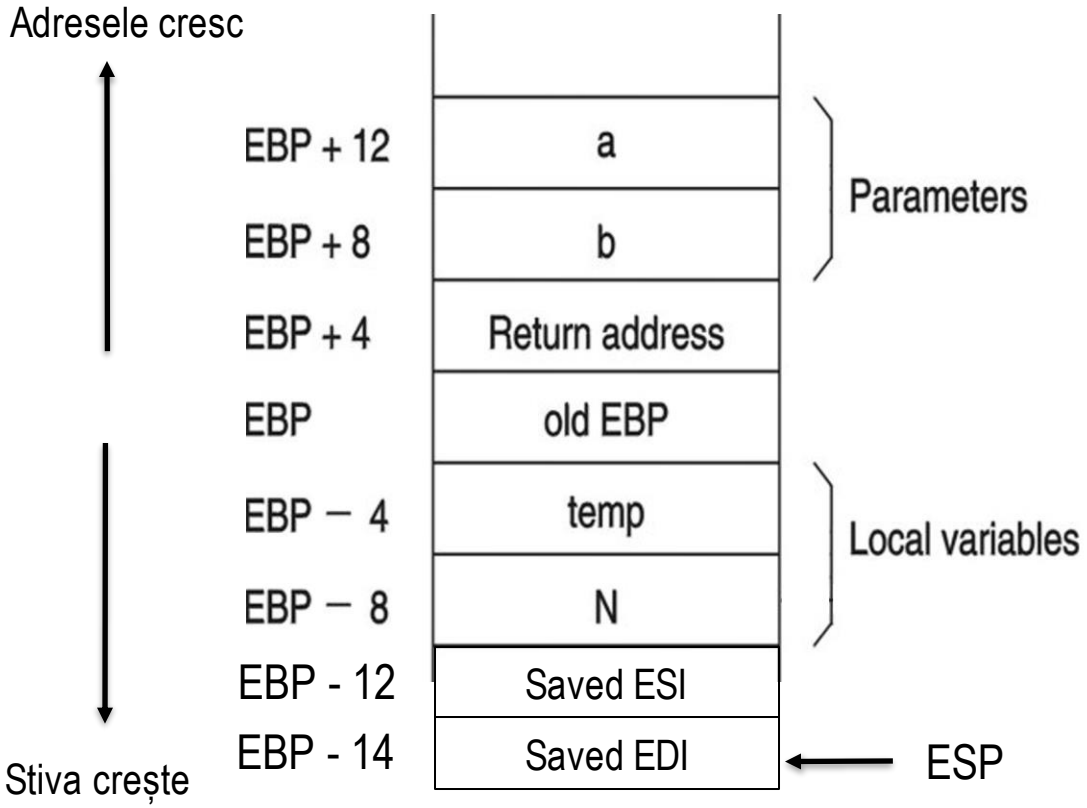
## Global si External

- Programarea mixta implica cel putin 2 module
  - » Un modul in C si un modul in limbaj de asamblare
- Trebuie sa declaram functiile si procedurile care nu sunt definite in acelasi modul ca fiind external
  - » Section 5.10
- Procedurile folosite de alte module sunt globale

# Git:demo/curs-12

- Exemplu 1 în
  - **hll_ex1c.c**
  - **hll_test.asm**
- Exemplu 2
  - **hll_minmaxc.c**
  - **hll_minmaxa.asm**
- Exemplu 3
  - **hll_arraysumc.c**
  - **hll_arraysuma.asm**

# x86 cdecl calling convention




Convenția de apel folosită de C

- Default for gcc, Linux, MSVC
- Parametri pe stivă începând cu ultimul
- Return, old EBP, locale
- Salvare registre dacă e cazul
- Valoare retur în EAX
- Stiva eliberată de parametri în apelant
- ABI = application **binary** interface
  - Biblioteci
  - Plugin-uri

# Apelarea unei funcții cu parametri

push paramN

push paramN-1

....

push param2

push param1

call func

# Restaurarea stivei după apelul unei funcții

```
push paramN
push paramN-1
....
push param2
push param1
call func
add esp, N*4
```

# Salvarea registrelor

- Unele registre pot modificate în cadrul unei funcții
- Dacă avem nevoie de valori înainte sau după trebuie salvate în prealabil
  - Pe stivă (push eax), cel mai comun
  - Într-o zonă de memorie (date)
- După apelul funcției registrele în cauză sunt restaurate (pop de pe stivă, de exemplu)

# Referirea parametrilor unei funcții

- [EBP]: saved ebp
- [EBP+4]: return address
- [EBP+8]: first parameter
- [EBP+12]: second parameter
- [EBP+16]: third parameter
- ...

# Convenția de apel CDECL

Important!

- Apelantul
  - plasează parametrii pe stivă (în ordine inversă)
  - conservă EAX, ECX, EDX dacă este cazul
  - curăță stiva **după retur**

```
»add     ESP,4
```

Numărul total de octeți trimiși pe stivă

- Apelatul
  - alocă și eliberează variabilele locale
  - pastrează: EBP, EBX, ESI, si EDI (callee preserved)
  - valoarea de retur în EAX, sau EDX:EAX

# Inline Assembly

- Cod în limbaj de asamblare integrat în codul C
  - » Nu este necesar un modul separat scris in limbaj de asamblare
  - » Depinde de compilator, platformă
  - » gcc/x86: simplu, sau cu intrinsics
- Constructiile in limbaj de asamblare sunt identificate cu ajutorul cuvantului cheie **`asm`**
- Putem folosi **( )** pentru a grupa mai multe constructii in limbaj de asamblare

```
asm(
        assembly statement

    assembly statement...
);
```

# Inline Assembly (cont'd)

- Inline – curs-01/inline.c
- Exemple Inline (cu intrinsics)
  - Exemplu 1
    - » **`hll_ex1_inline.c`**
  - Exemplu 2
    - » Suma elementelor unui vector
    - » **`hll_arraysum_inline.c`**
  - Exemplu 3
    - » A doua versiune a ultimului exemplu
    - » **`hll_arraysum_inline2.c`**

Last slide