Laborator 07: Date Structurate. Structuri, vectori. Operatii pe siruri

În acest laborator vom introduce noțiunea de structură din limbajul assembly și vom lucra cu operații specializate pe șiruri.

Structuri

Structurile sunt folosite pentru a grupa date care au tipuri diferite, dar care pot fi folosite împreună pentru a crea un tip compus.

În continuare vom trece prin pașii necesari pentru a folosi o structură: declararea, instanțierea și accesarea câmpurilor unei structuri.

Declararea unei structuri

În NASM, o structură se declară folosind construcția `struc <nume structura>`, urmată de o listă de câmpuri și încheiată cu `endstruc`.

Fiecare câmp al structurii este definit prin următoarele: o etichetă (folosită pentru a putea accesa membrii), specificatorul de tip și numărul de elemente.

Exemplu:

```
struc mystruct
    a:    resw 1    ; a va referi un singur element de dimensiune un cuvânt
    b:    resd 1    ; b va referi un singur element de dimensiune un dublu cuvânt
    c:    resb 1    ; c va referi un singur element de dimensiune un octet
    d:    resd 1    ; d va referi un singur element de dimensiune un dublu cuvânt
    e:    resb 6    ; e va referi 6 elemente de dimensiune un octet
endstruc
```

Aici sunt folosite pseudo-instrucțiunile NASM din familia `res` pentru a defini tipul de date și numărul de elemente pentru fiecare dintre câmpurile structurii. Pentru mai multe detalii despre sintaxa `res` urmați acest link: https://www.nasm.us/doc/nasmdoc3.html#section-3.2.2 [https://www.nasm.us/doc/nasmdoc3.html#section-3.2.2]

Fiecare etichetă ce definește un câmp reprezintă offset-ul câmpului în cadrul structurii. De exemplu, `b` va avea valoarea 2, deoarece sunt 2 octeți de la începutul structurii până la câmpul `b` (primii 2 octeți sunt ocupați de cuvântul `a`).

Dacă doriți să folosiți același nume de câmp în două structuri diferite, trebuie să prefixați numele etichetei cu `.` (dot) astfel:

```
struc mystruct1
   .a:    resw 1
   .b:    resd 1
endstruc

struc mystruct2
   .a:    resd 16
   .b:    resw 1
endstruc
```

Folosiți contrucția `mystruct2.b` pentru aflarea valorii offset-ului lui 'b' din cadrul structurii mystruct2.

Instanțierea unei structuri

O primă variantă pentru a avea o structură în memorie este de a declara-o static în secțiunea `.data`. Sintaxa folosește macro-urile NASM `istruc` și `iend` și keyword-ul `at`.

În exemplul următor este prezentată instanțierea statică a structurii declarate mai sus, unde `struct_var` este adresa din memorie de unde încep datele.

```
struct_var:
    istruc mystruct
        at a, dw        -1
        at b, dd        0x12345678
        at c, db        ' '
        at d, dd        23
```

```
        at e, db        'Gary', 0
    iend
```

În cazul în care definiți câmpurile structurii folosind . (dot), instanțierea structurii se face în felul următor:

```
struct_var:
    istruc mystruct
        at mystruct.a, dw        -1
        at mystruct.b, dd        0x12345678
        at mystruct.c, db        ' '
        at mystruct.d, dd        23
        at mystruct.e, db        'Gary', 0
    iend
```

Pentru a nu inițializa valorile membrilor greșit, va trebui să aveți grijă ca pentru fiecare câmp, tipul de date din instanțiere să corespundă tipului din declarare.

Accesarea valorilor dintr-o structură

Pentru a accesa și/sau modifica un anumit membru al structurii instanțiate trebuie să îi cunoaștem adresa. Această adresă se poate obține calculând suma dintre adresa de început a structurii și offset-ul din cadrul structurii al membrului dorit.

Următoarea secvență de cod prezintă punerea unei valori în câmpul b al structurii și, ulterior, afișarea valorii acestui câmp.

```
mov eax, 12345
mov dword [struct + b], eax ; adresa câmpului b este adresa de bază a structurii instanțiate static + offset-ul câmpului (dat de eticheta 'b')

mov ebx, dword [struct + b] ; punerea valorii din câmpul b în registrul ebx pentru afișare
PRINTF32 `%d\n\x0`, ebx
```

Vectori

Putem considera un vector ca o înșiruire de elemente de același tip, plasate contiguu în memorie. Ați observat ceva similar în laboratoarele trecute când declaram static șiruri de caractere în secțiunea .data.

Declararea unui vector

În general, datele statice declarate pot fi inițializate sau neinițializate. Diferențierea se face atât prin faptul că la datele inițializate oferim o valoare inițială, dar și prin sintaxa NASM folosită. De exemplu, pentru a declara un vector de 100 de cuvinte inițializate cu valoarea 42, vom folosi construcția:

```
section .data
    myVect:    times 100    dw 42
```

Pe de altă parte, dacă dorim declararea unui vector de 20 de elemente dublu cuvinte neinițializate, folosim instrucțiuni din familia `res` astfel:

```
section .bss
    myVect:    resd 20
```

Vectori de structuri

Adesea vom avea nevoie de vectori care să conțină elemente de dimensiuni mai mari decât cea a unui cuvânt dublu. Pentru a obține acest lucru vom combina cele două concepte prezentate anterior și vom folosi vectori de structuri. Bineînțeles, instrucțiunile de operare pe șiruri nu vor funcționa, deci vom fi nevoiți să ne întoarcem la metoda clasică de accesare a elementelor: cea prin adresarea explicită a memoriei.

Pentru exemplul din această secțiune, creăm o structură ce reprezintă un punct într-un spațiu 2D.

```
struc point
    .x:    resd 1
    .y:    resd 1
endstruc
```

Declararea unui vector de structuri

Deoarece NASM nu suportă niciun mecanism pentru a declara explicit un vector de structuri, va trebui să declarăm efectiv o zonă de date în care să încapă vectorul nostru.

Considerând că ne dorim un vector zeroizat de 100 de elemente de tipul structurii point (care este de dimensiune 8 octeți), trebuie să alocăm 100 * 8 (= 800) octeți.

Obținem:

```
section .data
    pointArray:    times 800    db 0
```

În plus, NASM oferă o alternativă la calculul "de mână" al dimensiunii unei structuri, generând automat macro-ul `<nume structura>_size`. Astfel, exemplul anterior poate deveni:

```
section .data
    pointArray:    times point_size * 100    db 0
```

Dacă ne dorim să declarăm un vector de structuri neinițializat putem folosi:

```
section .bss
    pointArray:    resb point_size * 100
```

Parcurgerea unui vector de structuri

Cum am mai spus, pentru accesarea câmpului unui element dintr-un vector trebuie să folosim adresarea normală (în particular adresarea "based-indexed with scale"). Formula pentru aflarea adresei elementului este `baza_vector + i * dimensiune_struct`.

Presupunând că avem în registrul `ebx` adresa de început a vectorului și în `eax` indicele elementului pe care dorim să îl accesăm, exemplul următor prezintă afișarea valorii câmpului y a acestui element.

```
mov ebx, pointArray                         ; mutăm în ebx adresa de început a șirului
mov eax, 12                                 ; să zicem că vrem al 13-lea element
mov edx, [ebx + point_size * eax + point.y] ; se calculează adresa câmpului dorit între []
                                            ; și apoi se transferă valoarea de la acea adresă
                                            ; în registrul edx

PRINTF32 `%u\n\x0`, edx
```

Parcurgem vectorul, având la fiecare iterație indicele curent în registrul eax. Putem să afișăm valorile din ambele câmpuri ale fiecărui element din vector cu următorul program:

```
struc   point
    .x: resd 1
    .y: resd 1
endstruc

section .data
    pointArray: times point_size * 100 db 0

section .text
    global CMAIN

CMAIN:
    push ebp
    mov ebp, esp

    xor edx, edx
    xor eax, eax
label:
    mov edx, [pointArray + point_size * eax + point.x] ; accesăm membrul x
    PRINTF32 `%u\n\x0`, edx

    mov edx, [pointArray + point_size * eax + point.y] ; accesăm membrul y
    PRINTF32 `%u\n\x0`, edx

    inc eax ; incrementarea indicelui de iterare
    cmp eax, 100
    jl label

    leave
    ret
```

Exerciții

În cadrul laboratoarelor vom folosi repository-ul de git al materiei IOCLA - https://github.com/systems-cs-pub-ro/iocla [https://github.com/systems-cs-pub-ro/iocla]. Repository-ul este clonat pe desktop-ul mașinii virtuale. Pentru a îl actualiza, folosiți comanda `git pull origin master` din interiorul directorului în care se află repository-ul (`~/Desktop/iocla`). Recomandarea este să îl actualizați cât mai frecvent, înainte să începeți lucrul, pentru a vă asigura că aveți versiunea cea mai recentă. Dacă doriți să descărcați repository-ul în altă locație, folosiți comanda `git clone https://github.com/systems-cs-pub-ro/iocla ${target}`. Pentru mai multe informații despre folosirea utilitarului `git`, urmați ghidul de la Git Immersion [https://gitimmersion.com/].

0. Recapitulare: Fibonacci sum

Pornind de la fișierul `fibo_sum.asm`, implementați un program care calculează suma primelor N numere din șirul fibonacci utilizând instrucțiunea `loop`. Suma primelor 9 este 88.

Puteți să investigați secțiunea Instrucțiuni de transfer de date [https://ocw.cs.pub.ro/courses/iocla/laboratoare/laborator-05#instructiuni_de_transfer_de_date] din laboratorul 5.

1. Tutorial: Afișarea conținutului unei structuri

În programul `print_structure.asm` sunt afișate câmpurile unei structuri.

Urmăriți codul, observați construcțiile și modurile de adresare a memoriei. Rulați codul.

O explicație utilă pentru instrucțiunea lea o găsiți: aici [https://www.aldeid.com/wiki/X86-assembly/Instructions/lea]

Treceți la următorul pas doar după ce ați înțeles foarte bine ce face codul. Vă va fi greu să faceți următorul exercițiu dacă aveți dificultăți în înțelegerea exercițiului curent.

2. Modificarea unei structuri

Scrieți cod în cadrul funcției `main` astfel încât să modificați câmpurile structurii `sample_student` pentru ca:

- anul nașterii să fie 1993
- vârsta să fie 22
- grupa să fie 323CA

Nu modificați ce se afișează, modificați codul structurii. Nu vă atingeți de codul de afișare, acel cod trebuie să rămână același. Trebuie să adăugați la începutul funcției `main`, în locul marcat cu TODO codul pentru modificarea structurii.

Trebuie să modificați conținutul structurii din cod, adică trebuie să scrieți în zona de memorie aferentă câmpului din structură. Nu modificați structura din secțiunea `.data`, este vorba să folosiți cod pentru a modifca structura.

Pentru modificarea grupei, va trebui să schimbați al treilea octet/caracter al câmpului `group` (adică octetul/caracterul cu indexul 2).

3. Getter

În fișierul `getter_setter_printf.asm` implementați funcțiile `get_int`, `get_char`, respectiv `get_string`, ce vor returna valorile câmpurilor `int_x`, `char_y`, respectiv `string_s` din structura `my_struc`. Valorile vor fi returnate prin registrul `eax`.

Funcțiile primesc ca argument un pointer la începutul structurii. Parametrul se află la adresa `ebp + 8` și pentru a fi folosit ca pointer, trebuie citită valoarea sa într-un registru (ex. registrul `ebx`).

Output-ul programului după o rezolvare corectă este:

```
1000
a
My string is better than yours
```

Urmăriți comentariile marcate cu **TODO**.

4. Setter

Mai departe, implementați funcțiile `set_int`, `set_char`, respectiv `set_string`, ce vor suprascrie valorile câmpurilor `int_x`, `char_y`, respectiv `string_s` din `structura my_struc` cu noile valori date.

Funcțiile primesc doi parametri - un pointer la începutul structurii, ca la exercițiul anterior, și o valoare care trebuie să fie folosită ca sursă pentru atribuire. Cei doi parametri sunt în ordine la adresele `ebp + 8` (primul parametru) și `ebp + 12` (al doilea parametru).

Output-ul programului după o rezolvare corectă este:

```
2000
b
Are you sure?
```

Urmăriți comentariile marcate cu `TODO`.

5. Printf

În funcția `main`, afișați câmpurile structurii utilizând apeluri ale funcției `printf`. Verificați că programul afișază valorile corespunzătoare cu, respectiv fără, folosirea funcțiilor `set_*`. Puteți folosi formaturile de la liniile 10-12 pentru a printa câmpurile.

6. Bonus: Căutarea unui subșir într-un șir

Găsiți toate aparițiile subșirului `substring` în șirul `source_text` din fișierul `find_substring.asm`.

Afișați rezultatele sub forma:

```
Substring found at index: <N>
```

Nu puteți folosi funcția de bibliotecă `strstr` (sau similar) pentru acest subpunct.

Pentru afișare puteți folosi atât macro-ul `PRINTF32`, cât și funcția `printf`, ca la exercițiile anterioare. Pașii pentru afișare folosind `printf` sunt următorii:

- puneți pe stivă valoarea pe care vreți să o afișați (poziția unde a fost găsit subșirul)
- puneți pe stivă adresa șirului `print_format`
- apelați funcția `printf`
- curățați parametrii adăugați anterior de pe stivă prin adăugarea valorii 8 la registrul `esp` (fiecare dintre parametri are 4 octeți).

Soluții

Soluțiile pentru exerciții sunt disponibile: aici [https://elf.cs.pub.ro/asm/res/laboratoare/lab-07-sol.zip]