Laborator 0x05

1. Proceduri în x86, sintaxa AT&T
2. Instrucțiunile CALL și RET
3. Convenții de scriere a procedurilor
4. Convenții privind restaurarea codului de apel

Mecanismul de a apela o procedură

- se încărcau parametrii procedurii pe stivă
- se apela procedura prin instrucțiunea call

```
proc ( arg1, arg2, arg3)
```

- argumentele se încarcă în ordine inversă pe stivă, și anume

```
pushl arg3
pushl arg2
pushl arg1
call proc
```

- pentru fiecare push(l) făcut, trebuie să facem și un pop(l)

```
popl %ebx
popl %ebx
popl %ebx
```

- până acum, am lucrat cu PRINTF, SCANF și FFLUSH

Instrucțiunile CALL și RET

call este instrucțiunea utilizată pentru a apela o procedură

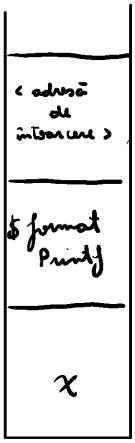
ret este instrucțiunea utilizată pentru a reveni dintr-o procedură

În x86 există registrul **%eip** (instruction pointer), care ține minte adresa următoarei instrucțiuni ce va trebui executată

În cazul procedurilor, utilizarea instrucțiunii call pune pe stivă adresa de întoarcere

```
pushl   x
pushl   $ format Printf
call    printf
popl    %ebx
popl    %ebx
```

=> push cu < adresa de întoarcere >



← pointează %esp D(%esp)

← 4 (%esp)

← 8 (%esp)

- ret se leagă înapoi de locul în care a fost apelată procedura, ținând minte adresa următoarei instrucțiuni de după call

Convenții de scriere a procedurilor

- este impusă utilizarea registrului %ebp (base pointer). El este fixat la începutul procedurii, în cadrul de apel

Operațiile push(l) și pop(l) modifică registrul %esp (stack pointer)

- push(l) <=> sub $4, %esp - se scade 4 din valoarea curentă reținută în stack pointer
- pop(l) <=> add $4, %esp - se adaugă 4 din valoarea curentă reținută în stack pointer (%esp)

Stiva { crește către valori mai mici
scade către valori mai mari

Reg. %esp va fi utilizat strict în operațiile push și pop, în timp ce accesarea din cadrul stivei va fi efectuată prin reg %ebp

%esp ← push ^ pop

%ebp ← accesarea din cadrul stivei (accesarea argumentelor a variabilelor locale

Convenții privind restaurarea codului de apel

```
movl $15, %ebx
movl index, %ecx
call printf
```

← în punctul ăsta, %ecx nu va mai avea valoarea din index

← în punctul ăsta, știm sigur că %ebx e tot 15 (restaurarea lui %ebx)

printf modifică și el, intern, regiștii %eax, %ebx, %ecx, %edx, %esi, %edi

Orice procedură care este dezvoltată în x86 este obligată să restaureze doar valorile din regiștii %ebx, %esi, %edi

Asta înseamnă că, dacă înainte de un call la o anumită procedură avem anumite valori stocate în %eax, %ecx, %edx NU avem garanția că vom găsi aceleași valori la revenirea din codul de apel

Obs mecanismul de restaurare nu este implicit, ci trebuie asigurat de noi

Restaurarea se va face prin intermediul stivei, astfel:

- la intrarea în procedură, se va face push cu registrii de restaurat (%ebx, %esi, %edi) pe care îi utilizăm; în acest mod, am pus vechea valoare a lui %ebx pe stivă

- la ieșirea din procedură, voi face pop și voi indica registrul pe care vreau să-l restaurez

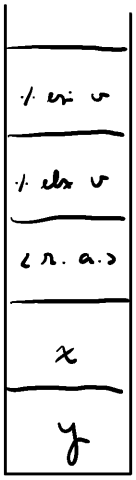
pushl %ebx ← am pus %ebx pe stivă, astfel, vechea lui val. salvată

pushl %esi ← am pus %esi pe stivă pt. că vreau să-l utilizez în cadrul procedurii

...

popl %esi ← am restaurat vechea val. a lui %esi

popl %ebx ← descarc în %ebx conținutul din vf. stivei, în acest mod, vechea val. a lui %ebx



O completare pt. %ebp

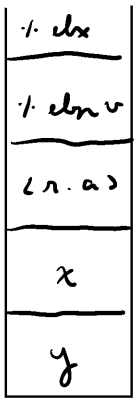
Pentru a-l putea utiliza, avem nevoie de urm. pași:

1. Se face push cu vechea val. a lui %ebp pe stivă fix la intrarea în codul de apel

2. Se face %ebp pointer în codul de apel, a.î. elementele de pe stivă să fie accesate în raport cu acesta

Cum arată stiva pentru suma a două nr:

suma (x, y)



← %esp

← %ebp

-4 (%ebp) sau 0(%esp)

0 (%ebp)

4 (%ebp)

8 (%ebp)

12 (%ebp)

Respectarea convențiilor asigură și posibilitatea de implementare a procedurilor recursive și a call-urilor imbricate. (de exemplu, în procedura care det. câte elemente maxime avem într-un array, putem face call către o procedură care ne dă maximul din array-ul respectiv)

Returnarea valorilor se face prin reg. %eax

Dacă implementez suma a două numere, mă aștept ca, la revenirea din apel, să găsesc rez. stocat în registrul %eax

%eax = suma (x, y)

```
long suma ( long x, long y)
  {
      return %eax
  }
```

Aplicație

Să se implementeze o procedură care să verifice dacă un argument este un număr perfect (egal cu suma divizorilor până la jumătate

perfect (long x) care returnează prin %eax
→ 1 dacă x este perf
→ 0 altfel

perfect (28) → 1 în %eax
perfect (13) → 0 în %eax

și afișați pe ecran " Numarul ... dat ca argument este / nu este nr. perfect "