Laborator 0x08

25 Nov 2024

1. Recap
2. Funcție : suma el. dintr-un array
3. Funcție : lungimea unui șir de caractere
4. Structura stivei în cazul apelurilor imbricate

Recap

- configurația stivei
- implementarea procedurilor

Convenții

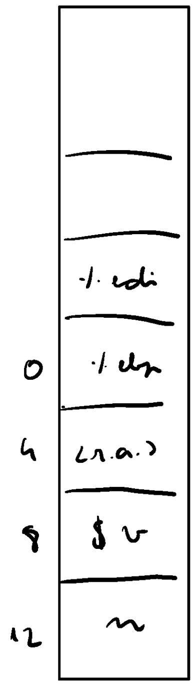
1. Arg se încarcă în ordine inversă pe stivă
2. apel prin call / ret ; call ::= jump et link
   - se trece ‹r.a.› pe stivă
   - jump
3. %eax, %ecx, %edx nu garantează restaurarea valorii returnate
4. %ebx, %esi, %esp sunt restaurați
5. accesarea în cadrul de apel se face relativ la %ebp

ex 1

Să se implementeze procedura <u>sumArg</u> care să calculeze suma el. dintr-un array din întregi

```
.data
    v: .long 10, 20, 30, 15, 7
    n: .long 5
    formatPrintf: .asciz "Sum %d"
.text
```

```
.global main

sumArr :

    push %ebp
    mov %esp, %ebp
        xor %eax, %eax      # suma
        xor %ecx, %ecx      # index

        push %edi
        mov 8(%ebp), %edi
        push %ebx
        mov 12(%ebp), %ebx      # n

SumArr_loop:
        cmp %ecx, %ebx
        je sumArr_exit

        mov (%edi, %ecx, 4), %edx
        add %edx, %eax
        inc %ecx

        jmp sumArr_loop
sumArr_exit
        pop %ebx
        pop %edi
        pop %ebp
        ret
```



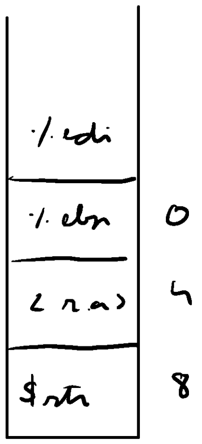
```
main :

    push   n
    push   $v
    call   sumArr
    add    $8, %esp

    push   %eax
    push   $formatPrintf
    call   printf
    add    $8, %esp

    mov    $1, %eax
    mov    $0, %ebx
    int    $0x80
```

ex 2

Scrieți o funcție **stringlen** care primește adresa unui șir de caractere și returnează lungimea acestui șir

```
    # long stringlen ( char *str)
```

```
. data
      n : .long 0
      v : .asciz "lungime \n"
. text
. global main
```



```
main :

    push  $ str
    call  stringlen
    add   $4, %ebp

    movl  %eax, n

    mov   $1, %eax
    mov   $0, %ebx
    int   $0x80
```

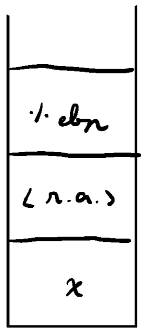
```
stringlen :

    push  %ebp
    mov   %esp, %ebp
    push  %edi
    mov   8(%ebp), %edi
    xor   %eax, %eax     # counter
                         # result

stringlen_loop :
    movb  (%edi, %eax, 1), %cl
    cmp   $0, %cl
    je    stringlen_exit
    inc   %eax
    jmp   stringlen_exit

stringlen_exit
    pop   %edi
    pop   %ebp
    ret
```

Structura stivei în cazul apelurilor imbricate

$$g(x) = x + 1$$
$$f(x) = 2 \cdot g(x)$$

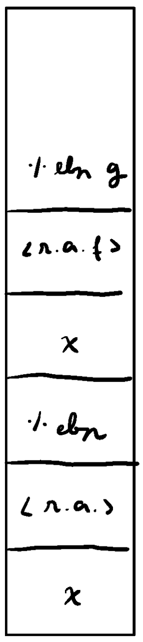
g :

```
push %ebp
mov %esp, %ebp

  mov 8(%ebp), %eax
  inc %eax
  pop %ebp
   ret
```



f :

```
push %ebp
mov %esp, %ebp

    mov 8(%ebp), %eax
    push %eax
    call g
<r.a. f.> ⟶    add $4, %esp
    # eax ← g(x)
    add %eax, %eax
    pop %ebp
    ret
```



ex 3

a) Scrieți o procedura **longMax** (long * v, long n)

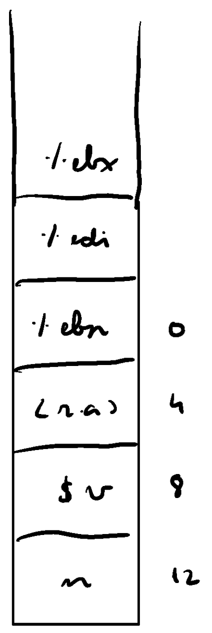
=> det max din Arr

b) Scrieți o procedură **nrAparMax** (long * v, long n)

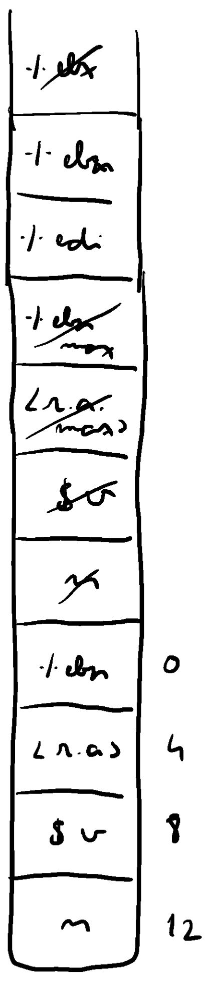
=> det nr. apariții al elem. max

In nrAparMax efectuați un apel la max

```
.data
    n: .long 6
    v: .long   20, 5, 20, 3 , 6, 20
.text
.global main

  max :
        push %ebp
        mov %esp, %ebp
          mov $0, %ecx
          mov $0, %eax
            push %edi
            mov 8(%ebp), %edi
            push %ebx
            mov 12(%ebp), %ebx
```



```
max_loop :
        cmp  %ecx, %ebx
        je   max_exit

        push  %edx
        mov   (%edi, %ecx, 4), %edx

        cmp  %eax , %edx
        jg    change_max
        jmp   cont_max
change_max :
            mov  %edx, %eax
  cont_max :
              inc  %ecx
              pop  %edx
              jmp  max_loop
  max_exit :
              pop  %ebx
              pop  %edi
              pop  %ebp
              ret


mArrayMax :
        push %ebp
        mov  %esp, %ebp
```



```
        mov   12(%ebp), %eax
        push  %eax
        mov   8(%ebp), %eax
        call  max
        add   $8, %esp

        %eax ← max
        mov   %eax, %edx
        %edx ← max
        mov   $0, %eax

        push  %edi
        mov   8(%ebp, %edi)
        push  %ebx
        mov   12(%ebp), %ebx
        mov   $0, %ecx

nrAparMax_loop:
        cmp   %ecx, %ebx
        je    nrAparMax_exit

        push  %ebx

        mov   (%edi, %ecx, 4), %ebx
        cmp   %ebx, %edx
        je    et_adaugă
        jmp   et_cont_loop
```

```
et_adaugă:
        inc %eax
 et_cont_loop:
        pop %ebx
         inc %ecx
          jmp nAparMax_loop

nAparMax_exit:
          pop %ebx
          pop %edi
          pop %ebp
           ret


main:
      push n
      push $v
        call nAparMax
        add $8, %esp

         push %eax
          push $formatPrint
          call printf
           add $8, %esp
et_exit:
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```