

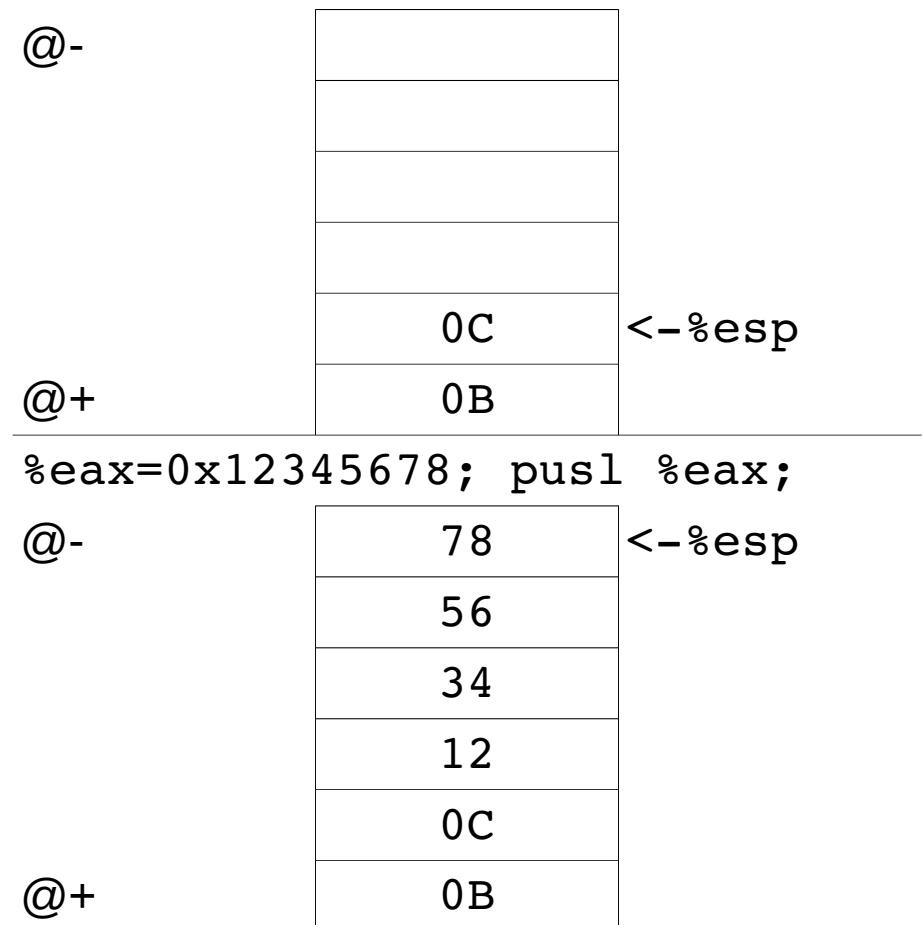
# FONAMENTS D'ORDINADORS

## TEMA 7: Subrutines

Manel Guerrero

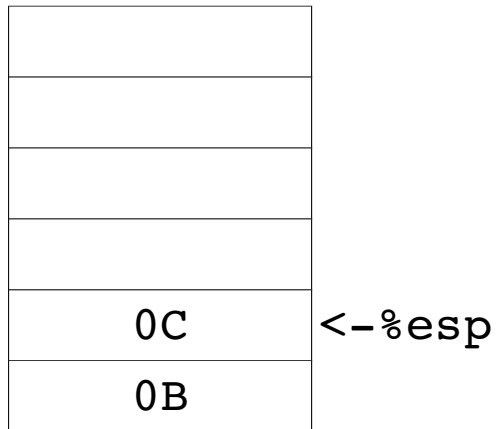
# [H10] La pila (stack)

- La pila és una pila de bytes LIFO: Last In, First Out.
- La pila té associat un punter al seu cim (%esp) que s'inicialitza pel SO al carregar el programa en memòria
- La pila creix en sentit decreixent de les adreces de memòria (anti-intuïtiu).
- Operacions:
- `push[w|1] op1` | Equival a:
  - `%esp = %esp - [2|4];`
  - `MEM[%esp] = op1;`
- `pop[w|1] op1` | Equival a:
  - `op1 = MEM[%esp];`
  - `%esp = %esp + [2|4];`



# La pila: exemple

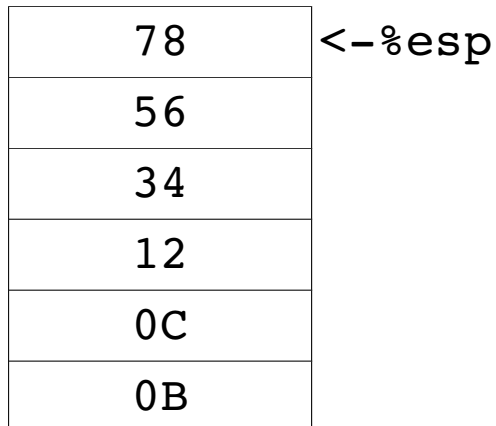
1) @-



@+

(%eax=0x12345678) pushl %eax

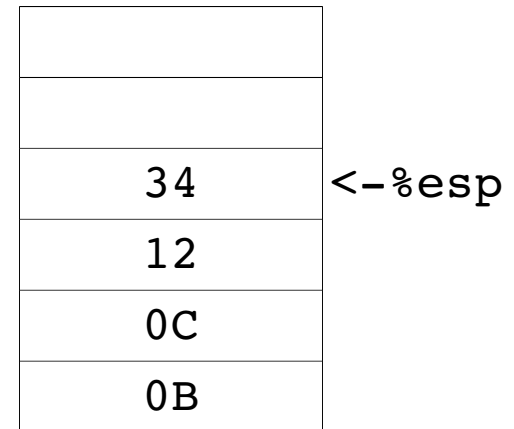
2) @-



@+

popw %ax

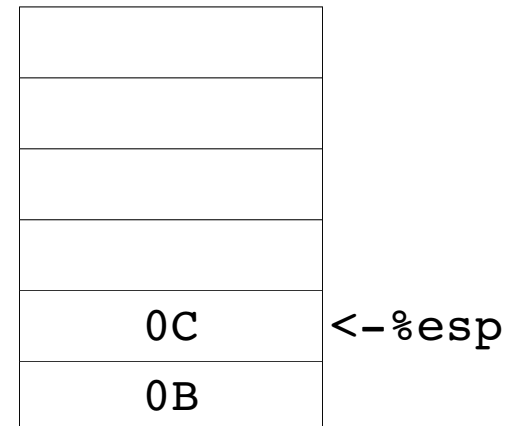
3) @-



@+

(%ax=0x5678) popw %bx;

4) @-



@+

(%bx=0x1234)

# Crida a una subrutina

- Si volem invocar una subrutina (com ara `printf("Hello World!\n");`) farem:
  - push dels paràmetres començant per la dreta (si el paràmetre és un vector, es passa el punter al vector).
  - call subrutina
  - Si la subrutina retorna algun enter el deixa a `%eax`.
  - Eliminar els paràmetres de la pila (com fer un pop però el paràmetre no es guarda en cap registre).
- `printf()` ha d'acabar en `'\n'` o no sortirà per pantalla (a diferencia de en C).

```
t7_ex_printf.s:  
.data  
s: .asciz "Hello World!\n"  
.text  
.globl main  
main:  
    pushl $s  
    # push param a pila (punter a s)  
    call printf # Cridar subrutina  
    addl $4, %esp  
    # Eliminar param de la pila  
    movl $1, %eax  
    movl $0, %ebx  
    int $0x80
```

# call versus jmp

- “call subrutina” equivaldria a:
  - `pushl %eip`
  - `jmp subrutina`
- `%eip` sent l'adreça de la següent instrucció després del 'call' (o adreça de retorn).
- En un codi no es permet operar directament amb `%eip`.
- L'instrucció per retornar (estarà dins de la funció cridada) “ret”, equival a:
  - `popl %eip`
- Això converteix una crida a funció en un salt després del qual podrem tornar al punt des del qual se l'ha cridat.

- Per cert, els paràmetres, (que es passen de dreta a esquerra), es guarden en @alineades a 4 bytes. Exemple:

```
movb $'a', %al
```

```
movsbl %al, %eax
```

```
pushl %eax
```

```
call subrutina
```

- El valor de retorn estarà a `%eax`, o `%ax`, o `%al` dependent de la mida.

# Crida a una subrutina 2

- Si no volem la subrutina que invoquem ens modifiqui els valors dels registres %eax, %ecx i %edx, a més a més, haurem de fer:
  - **push de %eax, %ecx, %edx.**
  - push dels paràmetres començant per la dreta (si el paràmetre és un vector, es passa el punter al vector).
  - call de la subrutina.
  - Si la subrutina retorna algun enter el deixa a %eax.
  - Eliminar els paràmetres de la pila (com fer un pop però el paràmetre no es guarda en cap registre).
  - **pop de %edx, %ecx, %eax.**
- Aquí podeu veure l'exemple t7\_printf.s que imprimeix per pantalla tots els elements del vector 'i' acabat en '0' excepte el '0'.

```
.data
v: .int 21, 15, 34, 11, 6, 50, 32, 0
s: .asciz "El numero es: %d\n"
.text
.globl main
main:
    movl $0, %ecx
loop:
    pushl %ecx          # 1 Salvar eax,ecx,edx
    pushl v(,%ecx,4)   # 2 Pasar params
    pushl $s           # 2 Pasar params
    call printf        # 3 Llamar subrutina
                        # 12 Resultado en eax
    addl $8, %esp      # 13 Eliminar params
    popl %ecx          # 14 Restaurar eax,ecx,edx
    incl %ecx
    cmpl $0, v(,%ecx,4)
    jne loop
# Finalitzar programa
```

# Ex2: Imprimir contingut vector

```
t7_printf2.s:                                     # 12 Resultado en eax
.data                                             # 13 Eliminar params
v: .int 21, 15, 34, 11, 0
s: .asciz "v[%d] = %d\n"
.text
.globl main
main:
    movl $0, %ecx
loop:
    pushl %ecx # 1 Salvar eax,ecx,edx
    pushl v(,%ecx,4) # 2 Pasar params
    pushl %ecx # 2 Pasar params
    pushl $s# 2 Pasar params
    call printf # 3 Llamar subrutina

    # 14 Restaurar
    popl %ecx
    incl %ecx
    cmpl $0, v(,%ecx,4)
    jne loop

    movl $1, %eax
    movl $0, %ebx
    int $0x80
=====
i[0] = 21
i[1] = 15
i[2] = 34
```



# [H11] Codi d'una subrutina

- Si la subrutina l'implementem nosaltres, farem:
  - Establir l'enllaç dinàmic:
    - `pushl %ebp`
    - `movl %esp, %ebp`
  - De manera que:
    - `(%ebp)`: Antic `%ebp`
    - `4(%ebp)`: Adreça de retorn
    - `8(%ebp)`: Param #1
    - `12(%ebp)`: Param #2
  - El codi del que ha de fer la funció.
  - Moure el resultat a `%eax`.
  - Desfer l'enllaç: `popl %ebp`
  - Retornar: `ret #popl %eip`

```
t7_minuscules_print.s:
minuscula: # Passa a minusc. un char
    pushl %ebp          # Establir enllaç
    movl %esp, %ebp    # dinamic
    movb 8(%ebp), %al  # Param1: 8(ebp)
    cmpb $'A', %al     # Funció:
    jl endif           # r = p1;
    cmpb $'Z', %al     # if ((r>='A')&&
    jg endif           #      (r<='Z'))
    addb $'a'-'A', %al # r=r+'a'-'A';
endif:                # resultat a %eax
    popl %ebp          # Desfer enllaç
    ret                # return r;
```

# Ex: Passar vector a minúscules

```
t7_minuscules_print_2.s:      pushl %ecx                                movl $0, %ebx
MIDA = 100                   movsbl is(%ecx), %dl                       movl $1, %eax
.data                        pushl %edx                                int $0x80
is: .asciz "MaYuSCuLas.\n"   call minuscula                            minuscula:
.bss                         addl $4, %esp                               pushl %ebp
.comm os,MIDA,1              popl %ecx                                  movl %esp, %ebp
.text                        movb %al, os(%ecx)                         movb 8(%ebp), %al
.global main                  incl %ecx                                  cmpb $'A', %al
main:                         jmp for                                     jl minendif # %al<$'A'
    pushl $is                               endfor:                                    cmpb $'Z', %al
    call printf                               movb $0, os(%ecx)                         jg minendif # %al>$'Z'
    addl $4, %esp                             pushl $os                                  addb $'a'-'A', %al
    movl $0, %ecx                             call printf                                minendif:
for:                           addl $4, %esp                               popl %ebp
    cmpb $0, is(%ecx) #NO , $0!              ret
    je endfor # is(%ecx)==$0
```

# [H12] Repassem funciones

```
t7_minuscules_print_2.s (~/.Dropbox/TAROM/FO/TEST/I32) - VIM
1 MIDA = 100
2 .data
3 is: .asciz "Demasiadas
   Mayusculas PARA TAN p
   OcAs BalAs.\n"
4 .bss
5 .comm os,MIDA,1
6 .text
7 .global main
8 main:
9  pushl $is
10 call printf
11  addl $4, %esp
12
13  movl $0, %ecx
14 for:
15  # is(%ecx)==$0
16  cmpb $0, is(%ecx)
17  je endfor
18  pushl %ecx
19  movb is(%ecx), %dl
20  pushl %edx
21  call minuscula
22  addl $4, %esp
23  popl %ecx
24  movb %al, os(%ecx)
25  incl %ecx
26  jmp for
27 endfor:
28  movb $0, os(%ecx)
29
30  pushl $os
31  call printf
32  addl $4, %esp
33
34  movl $0, %ebx
35  movl $1, %eax
36  int $0x80
37
38 minuscula:
39  pushl %ebp
40  movl %esp, %ebp
41
42  movb 8(%ebp), %al
43  # %al < '$A'
44  cmpb '$A', %al
45  jl minendif
46  # %al > '$Z'
47  cmpb '$Z', %al
48  jg minendif
49  addb '$a'-'A', %al
50
51 minendif:
52  popl %ebp
53  ret
54

<s print 2.s 1,2      Alku <s print 2.s 41,0-1      62% <s print 2.s 54,0-1      Loppu
"t7_minuscules_print_2.s" 54L, 658C kirjoitettu
```

# [H13] Salvar regs %ebx, %esi, %edi

- Establir l'enllaç dinàmic:
  - `pushl %ebp`
  - `movl %esp, %ebp`
- De manera que:
  - `(%ebp)`: Antic `%ebp`
  - `4(%ebp)`: Adreça de retorn
  - `8(%ebp)`: Param #1
  - `12(%ebp)`: Param #2
- **Salvar regs %ebx, %esi, %edi**
- El codi del que ha de fer la funció.
- Moure el resultat a `%eax`.
- **Restaurar regs %edi, %esi, %ebx**
- Desfer l'enllaç: `popl %ebp`
- Retronar: `ret`

# Subrutina amb variables locals

- Establir l'enllaç dinàmic:
  - `pushl %ebp | movl %esp, %ebp`
- **Reservar espai per variables locals**
- Salvar regs `%ebx, %esi, %edi`
- El codi del que ha de fer la funció.
- Moure el resultat a `%eax`.
- Restaurar regs `%edi, %esi, %ebx`
- **Alliberar l'espai de les variables locals**
- Desfer l'enllaç: `popl %ebp`
- Retronar: `ret`
- De manera que el Bloc d'Activació és:
  - `-12(%ebp): %ebx, %esi, %edi`
  - `-8(%ebp): LVar#2<-%esp`
  - `-4(%ebp): LVar #1`
  - `(%ebp): Antic %ebp`
  - `4(%ebp): Adreça de retorn de la funció`
  - `8(%ebp): Param #1`
  - `12(%ebp): Param #2`
  - `16(%ebp): %eax, %ecx, %edx`
- I `%esp` apunta a l'última variable local (o a l'últim registre ficat per push).

# Ex: amb variables locals

multiplica.c:

```
#include<stdio.h>
```

```
main() {
```

```
    printf("%d\n",
```

```
        multiplica(2,3) +
```

```
        multiplica(5,2));
```

```
}
```

```
int multiplica(int a, int b)
```

```
{
```

```
    int m=0;
```

```
    while(b!=0) {
```

```
        m=m+a;
```

```
        b--;
```

```
    }
```

```
    return m;
```

```
}
```

# Ex: amb variables locals

```
multiplica.s: # 2*3+5*2=16
```

```
.text
.global main
main:
    pushl $3
    pushl $2
    call multiplica
    addl $8, %esp
    movl %eax, %ebx
    pushl $2
    pushl $5
    call multiplica
    addl $8, %esp
    addl %eax, %ebx
    movl $1, %eax
    int $0x80
```

```
multiplica:
    pushl %ebp                #Salvar antic base pointer
    movl %esp, %ebp          #Stack pointer -> base pointer
    subl $4, %esp            #Reservar espai per var local
    pushl %ebx                #Salvar ebx, esi, edi
    movl 8(%ebp), %eax        #Posar primer arg a %eax
    movl 12(%ebp), %ebx     #Posar segon arg a %ebx
    movl $0, -4(%ebp)        #Inicialitzar var local
multiplica_loop:
    cmpl $0, %ebx
    je multiplica_endloop
    addl %eax, -4(%ebp)
    decl %ebx
    jmp multiplica_loop
multiplica_endloop:
    movl -4(%ebp), %eax    #Posar valor de retorn a %eax
    popl %ebx                #Restaurar edi, esi, ebx
    addl $4, %esp            #Restaurar stack pointer
    # movl %ebp, %esp        #La línia anterior equival a aquesta
    popl %ebp                #Restaurar base pointer
    ret
```

# Resum de crida a subrutina

## Subrutina 1

- 1- Salvar los regs %eax, %ecx, %edx
- 2- Pasar los parámetros
- 3- Llamar a la subrutina 2
- 12- Recoger resultado
- 13- Eliminar parámetros
- 14- Restaurar regs %eax, %ecx, %edx

## Subrutina2

- 4- Establecer enlace dinámico
- 5- Reservar espacio para var locales
- 6- Salvar registros %ebx, %esi, %edi  
(Ejecución de la subrutina 2)
- 7- Devolver resultado
- 8- Restaurar regs %ebx, %esi, %edi
- 9- Liberar el espacio de var locales
- 10- Deshacer enlace dinámico
- 11- Retorno

## [H14] Ex: amb variables locals

- t7\_signes.s: Diu quants zeros, positius i negatius hi han en un vector d'enters (tres variables locals).

# [H15] Pas paràmetres per referència

- `t7_complex_swap.s`: Programa on una funció crida a una altra funció passant com a paràmetres un paràmetre seu i una variable local. (En realitat es faria amb la instrucció LEA, però no està dins del temari de FO).
- Si us pica la curiositat saber com funciona LEA mireu-vos `t7_complex_swap_with_lea.s`

# [H16] Grande Finale

- t7\_cuento\_minusculas\_b.s Diu quantes vegades apareix cada lletra en una cadena ascii. Ensenya com accedir a posicions de vectors que són passats com a paràmetres de funció o variables locals (adreçaments amb registre base de l'estil "OFFSET\_VAR(%ebp,%ecx,4)").
- t7\_cuento\_minusculas.s [Opcional] Versió que accedeix al vector declarat com variable local amb adreçaments de l'estil "(%esi,%ecx)" o "(%ebx,%ecx,4)").

## [H17] Si queda temps...

- t7\_fibonacci.s Un altre exemple d'adreçament del tipus "-4(%ebx,%ecx,4)".
- t7\_fibonacci\_2.s Un altre exemple d'adreçament del tipus "v(%esi,%ecx,4)".