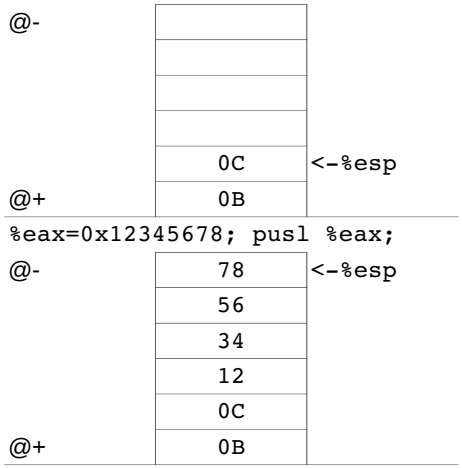
FONAMENTS D'ORDINADORS

TEMA 7: Subrutines

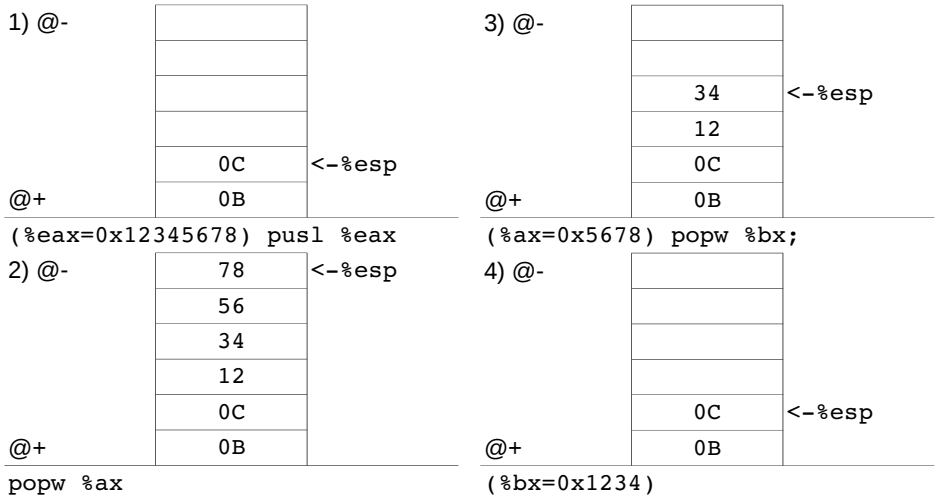
Manel Guerrero

[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|l] op1` | Equival a:
  - %esp = %esp - [2|4];
  - MEM[%esp] = op1;
- `pop[w|l] op1` | Equival a:
  - op1 = MEM[%esp];
  - %esp = %esp + [2|4];



La pila: exemple



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 depenent 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:
.data
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
# 12 Resultado en eax
 addl $12, %esp    # 13 Eliminar params
 popl %ecx   # 14 Restaurar
eax,ecx,edx
 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
```

[Opcional] Ex1: printf i scanf

```
t7_scanf.s
data
s: .asciz "Introdueix un num :"
is: .asciz "%d"
os: .asciz "Num és:%d.Scanf ret:%d.\n"
.bss
.comm n,4,4
.text
.global main
main:
 #printf(s);
 pushl $s
 call printf
 addl $4, %esp
 #scanf(is, &n);
 pushl $n
 pushl $is
 call scanf
 addl $8, %esp
 #printf(os,n,eax)
 pushl %eax
 pushl n
 pushl $os
 call printf
 addl $12, %esp
 #sys_exit
 movl $0, %ebx
 movl $1, %eax
 int $0x80
```

[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`
  - Retronar: `ret #popl %eip`

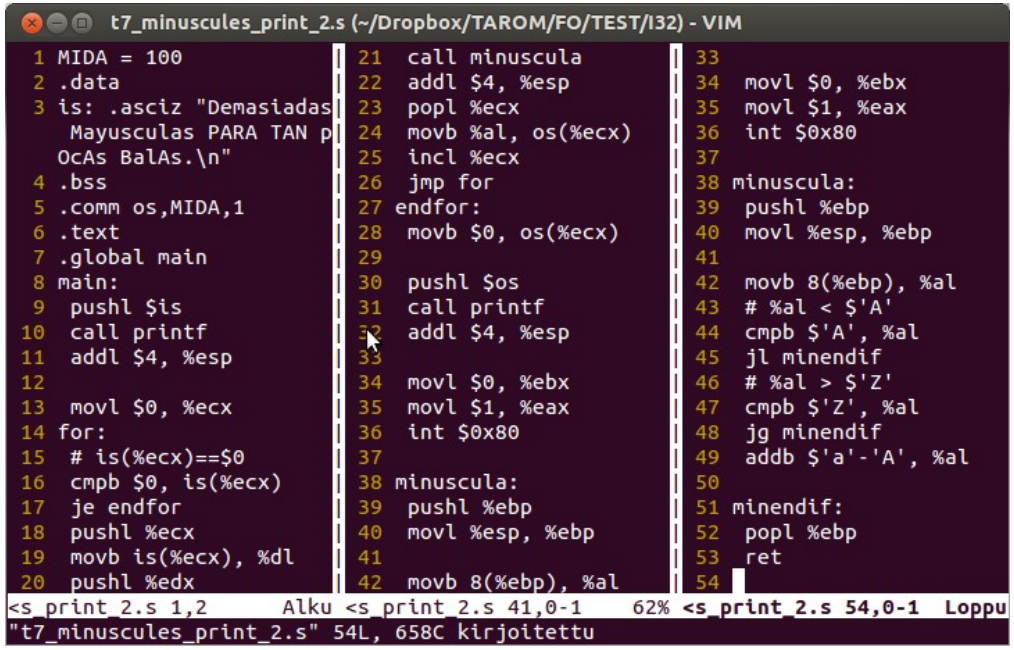
```
t7_minuscules_print.s:
minuscula: # Passa a minusc. un char
 pushl %ebp         # Establir enllaç
 movl %esp, %ebp    # dinamic
 movb 8(%ebp), %al  # Param1: 8(ebp)
 cmpb $'A', %al     # Funció:
 jl endif           # r = p1;
 cmpb $'Z', %al     # if ((r>='A')&&
 jg endif           #     (r<='Z'))
 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:
MIDA = 100
.data
is: .asciz "MaYuSCuLas.\n"
.bss
.comm os,MIDA,1
.text
.global main
main:
 pushl $is
 call printf
 addl $4, %esp
 movl $0, %ecx
for:
 cmpb $0, is(%ecx) #NO ,$0!
 je endfor # is(%ecx)==$0
 pushl %ecx
 movsbl is(%ecx), %dl
 pushl %edx
 call minuscula
 addl $4, %esp
 popl %ecx
 movb %al, os(%ecx)
 incl %ecx
 jmp for
endfor:
 movb $0, os(%ecx)

 pushl $os
 call printf
 addl $4, %esp
 movl $0, %ebx
 movl $1, %eax
 int $0x80
minuscula:
 pushl %ebp
 movl %esp, %ebp
 movb 8(%ebp), %al
 cmpb $'A', %al
 jl minendif # %al<$'A'
 cmpb $'Z', %al
 jg minendif # %al>$'Z'
 addb $'a'-'A', %al
minendif:
 popl %ebp
 ret
```

[H12] Repassem funcions



[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 referencia

- 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)".