Exemples de programació en assemblador (1a part):

Material utilitzat del mòdul de programació en assemblador (x86-64):

3. El llenguatge d'assemblador per a l'arquitectura x86-64
4. Introducció al llenguatge C
5. Conceptes de programació en assemblador i C
   5.2. Operacions aritmètiques
   5.3. Control de flux
6. Annex: manual bàsic del joc d'instruccions (a mode de consulta)

Descripció:

En aquests exemples s'implementen un conjunt d'operacions en llenguatge assemblador.

S'utilitzen els fitxers e1Sol.asm i e1cSol.c que us donem. En aquests fitxers, cada subrutina d'assemblador i cada funció de C tenen una capçalera on s'explica breument que fan.

Les dades que s'utilitzen estan definides com a variables globals en el codi C i podem accedir-hi des de les subrutines d'assemblador (estan declarades com *extern* en el codi assemblador per a permetre l'accés), d'aquesta forma no és necessari definir més variables ni en el codi C, ni en el codi assemblador. Totes les dades definides són de tipus int (32 bits), per tant, per treballar amb aquest tipus de dades en el codi assemblador haurem d'utilitzar registres de 32 bits: eax, ebx, ecx, edx, esi, edi, r8d, ..., r15d.

Nota: Les variables R1, .. R5 declarades al codi C, són variables per representar registres (tipus CISCA). No són registres. Per aquest motiu, per accedir-hi des d'assemblador les posarem entre claudàtors [R1], ..., [R5].

El programa principal és la funció main() del codi C, en aquesta funció hi ha les crides a les subrutines en assemblador. El programa en assemblador no té sortida per pantalla, per tant, per comprovar el seu correcte funcionament, si no és suficient la sortida per pantalla implementada en el codi C, caldrà utilitzar el depurador (kdbg).

Totes les subrutines d'assemblador i funcions de C són les mateixes, o molt semblants, a les treballades a l'exercici de la part de teoria E-Teo-1. D'aquesta manera podreu posar en pràctica els coneixements treballats a la part de teoria i podreu validar si les vostres solucions també funcionen correctament. Us donem el fitxer e1Enun.asm, on només hi ha les capçaleres de les subrutines, per a que pugueu provar d'implementar vosaltres mateixos les diferents subrutines.

Les funcions de C tenen la mateixa funcionalitat que les subrutines d'assemblador, són una bona guia per entendre les subrutines d'assemblador, i en cas de dubte, les podeu executar per a comprovar que els resultats obtinguts amb les funcions de C són els mateixos que amb les subrutines d'assemblador.

Les crides a les funcions de C estan comentades a la funció main(). Per a executar-les només cal treure el símbol de comentari '//' i posar-lo a la crida de la subrutina d'assemblador de la següent forma:

```
summatory();                    // summatory();
// summatory_C();               summatory_C();
```

Si feu les dues crides, en els casos que es modifiquin les variables que utilitzem en aquella crida, els resultats poden ser diferents. Per exemple:

```
num=3
summatory()   ; el resultat serà sum=3+2+1=6
              ; però com aquesta crida deixa num=0
sumatory_C()  ; ara sum=0
```

Podeu comprovar el funcionament de cada subrutina per a diferents valors de les variables, que inicialitzem en la funció main() del codi C, per a validar diferents possibilitats.

```
;(e1.1) equivalent a l'exercici de teoria (E-Teo-1.1.2.a)=======================
; if (A>B) then A--; else B=B+4;
; B=B*A;
ifThenElse1:

       mov  eax, [A]
       mov  ebx, [B]
ifThenElse1_if:
       cmp  eax, ebx
       jle  ifThenElse1_else
ifThenElse1_then:
       dec  eax
       jmp  ifThenElse1_endif
ifThenElse1_else:
       add  ebx, 4
ifThenElse1_endif:
       imul ebx, eax
       mov  [A], eax
       mov  [B], ebx

       ret


;(e1.2) equivalent a l'exercici de teoria  (E-Teo-1.1.2.b)=======================
; sum = num + (num-1) + (num-2) + ... + 2 + 1
summatory:

; sum=0;
       mov eax, 0
; while (num>0) {
summatory_while:
       cmp DWORD [num], 0
       jle summatory_end_w
; sum=sum+num;
       add eax, [num]
; num--
       dec DWORD [num]
       jmp summatory_while
; }
summatory_end_w:
       mov [sum], eax

       ret


;(e1.3) equivalent a l'exercici de teoria (E-Teo-1.2.2.a)=======================
; if ( (R2>=R3) and (R4 < R5) ) then R1=1; else R1=0;
; R1: eax, R2:ebx, R3:ecx, R4:edx, R5:esi
ifThenElse2:
```

Escriu en R1 un 1 si el contingut de R2 és més gran o igual que el de R3 i a més el de R4 és més petit que el de R5. Si no passa tot l'anterior, escriu un 0.

```
        mov  ebx, [R2]
        mov  ecx, [R3]
        mov  edx, [R4]
        mov  esi, [R5]

        mov  eax,0;       ;else: condició per defecte.
ifThenElse2_if:
        mov  eax, 0
        cmp  ebx, ecx
        jl   ifThenElse2_endif
        cmp  edx, esi
        jge  ifThenElse2_endif
ifThenElse2_then:
        mov  eax, 1
ifThenElse2_endif:

        mov  [R1], eax    ;només modifiquem R1

        ret
```

```
;(e1.4) equivalent a l'exercici de teoria (E-Teo-1.2.2.b)=======================
; if ( (R2!=0) or (R4 <= R5) ) then R1=1; else R1=0
; R1: eax, R2:ebx, R3:ecx, R4:edx, R5:esi
ifThenElse3:
```

Escriu en R1 un 1 si es compleix una qualsevol de les dues condicions següents o be si es compleixen les dues:

- el contingut de R2 es diferent de 0
- el contingut de R4 es més petit o igual que el de R5.

En qualsevol altre cas escriu en R1 un 0.

```
ifThenElse3:

        mov  ebx, [R2]
        mov  ecx, [R3]
        mov  edx, [R4]
        mov  esi, [R5]

        mov  eax, 0             ;else: condició per defecte.
ifThenElse3_if:

        cmp  ebx, 0
        jne  ifThenElse3_then   ; si es compleix (R2 != 0) fer el cos del if
        cmp  edx, esi
        jg   ifThenElse3_endif  ; si no es compleix (R4<=R5) sortir del if
ifThenElse3_then:
        mov  eax, 1
ifThenElse3_endif:

        mov  [R1], eax          ;només modifiquem R1

        ret


;(e1.5) equivalent a l'exercici de teoria (E-Teo-1.2.2.b)=======================
; el contrari de ( (R2!=0) or (R4 <= R5) )  és ( (R2==0) and (R4 > R5) )
; if ( (R2==0) and (R4 > R5) ) then R1=0; else R1=1;
; és el mateix que fer: if ( (R2!=0) or (R4 <= R5) ) then R1=1; else R1=0
; R1:eax, R2:ebx, R3:ecx, R4:edx, R5:esi
ifThenElse4:
```

Aquesta subrutina fa exactament el mateix que l'anterior, només canviem la forma d'implementar la condició. En l'exemple anterior tenim una condició de la forma (CondA or CondB) i en aquest exemple implementem la condició negada: not (CondA or CondB) = not(CondA) and not(CondB).

```
ifThenElse4:

        mov  ebx, [R2]
        mov  ecx, [R3]
        mov  edx, [R4]
        mov  esi, [R5]

        mov  eax, 1             ;else: condició per defecte.
ifThenElse4_if:
        cmp  ebx, 0
        jne  ifThenElse4_endif  ;si no es compleix ( R2 != 0 ) sortir del if
        cmp  edx, esi
        jle  ifThenElse4_endif  ;si no es compleix ( R4 > R5 ) sortir del if
ifThenElse4_then:
        mov  eax, 0
ifThenElse4_endif:

        mov  [R1], eax          ;només modifiquem R1

       ret
```