Exercicis de la part teòrica

Comencem el treball de la part de teoria de la primera part del curs. Us recordem que és totalment recomanable dedicar el temps que requereix l'assignatura setmana a setmana, i que és el "secret" per a aprovar-la.

Durant pràcticament tot el curs anirem fent exercicis teòrics (E.Teo.1, E.Teo.2,…), que serviran per preparar les dues PACs i les proves finals.

En aquesta primera part de teòrica, dedicarem 4 setmanes a treballar els mòduls 1, 2, 3 i 7, estudiant la teoria i fent els exercicis que us proposarem, començant aquesta setmana amb el E.Teo.1 i acabarem, dedicant la darrera setmana a fer la PAC1, que serà molt semblant als exercicis que haurem fet.

Als mòduls 1, 2 i 3 es defineixen conceptes generals sobre arquitectures (llenguatge màquina) i per a això s'utilitzen exemples d'arquitectures comercials i també de la nostra arquitectura CISCA que es defineix formalment en el mòdul 7. Aquesta arquitectura, tot i ser semblant a l'arquitectura x86-64 que utilitzarem en la part pràctica, és força més simple, amb moltes menys instruccions i altres simplificacions. La farem servir en els exercicis teòrics, PACs i a les proves finals.

Els exercicis d'aquestes primera part dels curs tractaran el següent:

E-Teo-1:
Què fan les instruccions CISCA? Entendre com s'executa una instrucció o una petita seqüència d'instruccions CISCA i veure com modifiquen l'estat del computador (els registres, la memòria i els bits de resultat) per conèixer les instruccions i els modes d'adreçament més bàsics. (Mòduls 2 i 7).

També us proposarem escriure petits programes en llenguatge assemblador CISCA utilitzant tipus de dades bàsics, no estructurats. L'especificació del què ha de fer el codi assemblador es fa en el llenguatge C (i en algun cas, mitjançant un text). Això ens servirà per entendre cóm traduir de C a assemblador CISCA sentències condicionals i bucles. Per fer aquest exercicis heu de revisar l'apartat 2.4, Estructures de control, del mòdul 7 per a fer la traducció de sentencies de llenguatge C a llenguatge assemblador. Podeu revisar també l'apartat 4 del mòdul 6 on es fa una breu introducció al llenguatge C.

NOTA: Cal que repasseu com representar i operar nombres en hexadecimal i en complement a 2. Us serà útil per fer aquests exercicis. Això no està explicat en aquest materials tot i que es donen algunes indicacions.

E-Teo-2.
Escriure petits programes en llenguatge assemblador CISCA però amb accés a dades estructurades (bàsicament vectors i matrius) utilitzant modes d'adreçament indexats i relatius. Traducció de C a assemblador i d'assemblador a C.

Continuar revisant els mòduls 1 i 2 de teoria i l'apartat 4 del mòdul 6, sobre llenguatge C i l'apartat 2.4, Estructures de control, del mòdul 7 per a fer la traducció de sentencies de

llenguatge C a llenguatge assemblador però utilitzant tots els modes d'adreçament i totes les instruccions del joc.

També treballarem la codificació de les instruccions CISCA i l'especificació d'instruccions com a seqüència de micro-operacions. Això ens permetrà veure com funciona a més baix nivell el processador d'un computador.

Per fer aquest darrers exercicis cal revisar el mòdul 3 i estudiar els dos últims apartats, 3 i 4, del mòdul 7, sobre el format i codificació de les instruccions CISCA i sobre la execució de les instruccions en forma de micro-operacions.

Exercici Teo.1

Feu aquest exercici seguint la metodologia de treball que us hem proposat. Teniu dues setmana per fer aquest exercici, després us donarem la solució.

Aquest exercici no s'avalua, si no el podeu acabar tot no passa res, és per treballar els continguts de l'assignatura i perquè pugueu comentar amb la resta de companys els vostres resultat i els dubtes que us puguin sorgir.

Un cop que us facilitem les solucions, compareu-les amb les vostres, i verifiqueu que ho enteneu tot. Feu servir el fòrum de l'assignatura per tal de resoldre qualsevol dubte que us sorgeixi. Heu de tenir molt en compte que les solucions no són úniques.

Per facilitar la feina us donem resolt el primer exercici que és molt semblant al que us proposem fer a vosaltres. L'exercici s'enuncia a continuació i donem la resposta en blau.

Exercici Teo.1.1.1 (resolt)

Per a cadascuna de les següents preguntes, suposeu que l'estat inicial del computador (el valor que contenen els registres, posicions de memòria i els bits de resultat) abans de l'execució dels fragments de codi és el següent:

- Registres: Ri= i*4, per exemple: R0=0, R1=4, R2=8, etc. (excepte el R15 o SP, que té el valor inicial 0).
- Memòria: M(i)=(i+16) per a i = 0, 4, 8,… (excepte en les posicions de memòria en las que es troba el codi del programa). Exemple: M(00001000h)=00001010h, etc. La notació M(i) es refereix a la paraula de 4 bytes M(i)..M(i+3) en little endian.
- Bits de resultat del registre d'estat inactius: Z=0, S=0, C=0, V=0
- Registres especials: El valor del PC i del SP no són necessaris per la resolució d'aquest exercici.

Quin serà l'estat del computador després d'executar cada una de les següents instruccions i fragments de codi?
Indiqueu només el contingut dels registres i posicions de memòria que s'hagin modificat a conseqüència de l'execució del codi. Indiqueu el valor final de tots els bits de resultat. En aquest exercici no és necessari saber el que ocupa cada una de les instruccions en llenguatge màquina, ni us demanem el valor del PC en executar el codi i no es necessari saber el seu valor per fer els exercicis.

Pregunta a)

```
ADD R3, R6
```

Valors inicials:
R3 = 3 * 4 = 12, R6 = 6 * 4 = 24

Operació:
R3 = R3+R6 = 12d + 24d = 36d,
Modifica els bits de resultat, però en aquest cas els valors generats són els mateixos que els valors originals (tot zero)

Resultat final (només registres i posicions de memòria que canvien) i bits de condició.
R3 = 36d, Z = 0, S = 0, C = 0, V = 0
- Z=0, perquè el resultat 12 + 24 és diferent de 0
- S=0, perquè el resultat de 32 bits és positiu, i no té signe
- C=0, perquè l'operació 12 + 24 no genera transport
- V=0, perquè la operació 12 + 24 no genera sobreeiximent

Pregunta b)

```
ADD R6, [FFFFFFF8h]
```

Valors inicials:
Adreça de memòria: FFFFFFF8h (32 bits),
[FFFFFFF8h] contingut de l'adreça FFFFFFF8h = Memòria(FFFFFFF8h)= FFFFFFF8h+16d = FFFFFFF8h+10h = 100000008h
però com que el valor necessita 33 bits per representar-lo, i els registres tenen una mida de 32 bits, es descarta el bit més significatiu, i el resultat és:
Memòria(FFFFFFF8h) = (FFFFFFF8h + 10h) mòdul 100000000h = 100000008h mòdul 100000000h = 00000008h = 8d

R6 = 6 * 4 = 24d

Operació:
R6 = R6 + Memòria(FFFFFFF8h) = 24d + 8d = 00000018h + 00000008h = 00000020h = 32d

Resultat:
R6 = 32d = 00000020h = 20h, Z = 0, S = 0, C = 0, V = 0
- Z=0, perquè el resultat 24 + 8 és diferent de 0
- S=0, perquè el resultat de 32 bits és positiu, i no té signe
- C=0, perquè l'operació 24 + 8 no genera transport
- V=0, perquè la operació 24 + 8 no genera sobreeiximent

Pregunta c)

```
ADD R4, -2
```

Valors inicials:
-2d = en complement a 2 (Ca2): FEh
Per fer el Ca2 de (-2): Agafem el valor binari positiu: 2d= 00000010b, el neguem: 11111101b i sumem 1: 111111101b + 1 = 11111110b = FEh = FFFFFFFEh

R4 = 4 * 4 = 16d = 0000000010h

Operació:
R4 = R4 + (-2) = 16d + (-2d) = 10h + FEh = 00000010h + FFFFFFFEh = 0000000Eh = 0Eh = 14d

Resultat:
R4 = 14d = 0000000Eh, Z = 0, S = 0, C = 1, V = 0
Z=0, perquè el resultat 16d + (-2d) és diferent de 0
S=0, perquè el resultat de 32 bits és positiu, i no té signe
C=1, perquè l'operació 16d + (-2d) genera transport (ens portem una).
V=0, perquè la operació 16d + (-2d) no genera sobreeiximent

Pregunta d)

Codi:

```
                MOV    R2, 3
                MOV    R1, R2
    Loop:       DEC    R2
                JE     End_loop
                MUL    R1, R2
                JMP    Loop
    End_loop:   MOV    [100], R1
```

Desenvolupament instrucció a instrucció:
MOV R2,3
3d=03h=00000003h, valor que volem posar al registre R2.
R2 = 2 * 4 = 8d
Operació: R2 = 3d

MOV R1, R2
R2 = 3d (ara ja no val 2 * 4 = 8d ja que hem canviat el valor a la instrucció anterior).
R1 = 1 * 4 = 4d = 0000000004h
Operació: R1 = 3d = 03h = 00000003h

Loop: (1)
R2 = R2 – 1 = 3 – 1 = 2; Z=0, C=0, S=0, V=0
JE end_loop FALS (Z desactivat)
R1 = R1 * R2 = 3 * 2= 6
JMP loop (sempre salta)

Loop: (2)
R2 = R2 – 1 = 2 – 1 = 1; Z=0, C=0, S=0, V=0
JE end_loop FALS (Z desactivat)
R1 = R1 * R2 = 6 * 1 = 6

JMP loop (sempre salta)

Loop: (3)
R2 = R2 – 1 = 1 – 1 = 0; Z=1, C=0, S=0, V=0
JE end_loop **VERITAT** (Z activat)

end_loop:
M[100] = 6

Solució Final:
R1 = 6, R2 = 0, M[100] = 6
Z = 1, S = 0, C = 0, V = 0
(la darrera instrucció que ha modificat els bits de resultat ha estat DEC R2 en el Loop: (3)).

Exercici Teo.1.1.2 (resolt)

Escriviu un programa en llenguatge assemblador CISCA que tingui la funcionalitat que es descriu mitjançant un codi C en cada una de les següents preguntes.

Pregunta a)

Codi C:

```
if (A>B) then A-- else B=B+4;
B = B * A;
```

Codi assemblador:

```
            MOV   R0, [A]
if:         CMP   R0, [B]
            JLE   else
then:       DEC   R0
            JMP   endif
else:       ADD   [B], 4
endif:      MUL   [B], R0
            MOV   [A], R0
```

Suposeu que l'estat inicial del processador és el mateix que en l'exercici anterior i que les adreces simbòliques A i B valen 00000020h i 00000200h respectivament ([A] = [00000020h] i [B] = [00000200h]). M(i)=(i+8) para i = 0, 4, 8,…
Quin serà l'estat del computador després d'executar aquest fragment de codi?

Desenvolupament instrucció a instrucció:
MOV R0, [A]: R0 = Memòria(A) = Memòria(20h) = 20h + 8 = 28h
Les instruccions de transferència no modifiquen els bits de resultat i queden com estaven.

CMP R0, [B]: R0 - Memòria(B)= 28h - Memòria(200h) = 28h – (200h+8) = 28h-208h = -1E0h = FFFFFE20h (en Ca2)
Z=0, S=1, C=1, V=0 (El resultat genera signe i transport, però no sobreeiximent).

JLE else: CERT (S=1 implica resultat negatiu, Z desactivat implica que no és zero).
Saltem a l'etiqueta else: i executem la instrucció ADD [B],4
Les instruccions de salt no modifiquen els bits de resultat i queden com estaven.

else: ADD [B],4
Memòria(B) = Memòria(B) + 4 = Memòria(200h) + 4 = 208h + 4 = 20Ch
Memòria(200h) = 20Ch (es modifica el valor d'aquesta posició de memòria)
Z=0, S=0, C=0, V=0 (No s'activa cap bit de resultat).

endif: MUL [B], R0:
Memòria(B)= Memòria(B) x R0= Memòria(200h) x 28h = 20Ch x 28h =
(20Ch = (2 x 256 + 0 x 16 + 12 x 1) = 524d) (28h = 2 x 16 + 8 x 1 = 40d)
= 524 x 40 = 20960d = 051E0h
Memòria(200h) = 051E0h (es modifica el valor d'aquesta posició de memòria)
Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

MOV [A], R0
Memòria(A) = R0 = 28h ; Memòria(20h) = 28 h
Les instruccions de transferència no modifiquen els bits de resultat i queden com estaven.

Solució Final:
R0 = 28h,
M[A] = M[20h] = 28h (com abans d'executar),
M[B] = M[200h] = 051E0h
Z = 0, S = 0, C = 0, V = 0 (com s'han deixar el bits de resultat a la darrera instrucció que els ha modificat, en aquest cas el MUL [B], R0).

Quan hauria de valdre [A] inicialment perquè s'executés la instrucció DEC R0?

[A] = 209h

CMP R0, [B]: R0 - Memòria(B)=209h - Memòria(200h)=209h – (200h+8)=209h-208h=1
Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat i el JLE no saltaria).

Pregunta b)

Codi C:

```
sum = 0;
While (num > 0) {
        sum = sum + num;
        num--;
}
```

Codi assemblador:

```
                MOV   R1, 0
while:          CMP   [num], 0
                JLE   end_w
                ADD   R1, [num]
                DEC   [num]
                JMP   while
end_w:          MOV   [sum], R1
```

Suposeu que l'estat inicial del processador és el mateix que en l'exercici anterior i que les adreces simbòliques num i sum valen 00000004h i 00000008h respectivament ([num] = [00000004h] i [sum] = [00000008h]). M(i)=(i+8) para i = 0, 4, 8,…

Quin serà l'estat del computador després d'executar aquest fragment de codi?

Desenvolupament instrucció a instrucció:

MOV R1, 0: R1 = 0

Les instruccions de transferència no modifiquen els bits de resultat i queden com estaven.

(Recordeu que [num]=[00000004h]=4h + 8=12d i [sum]=[00000008h]=8h + 8=16d).

Loop: (1)

while: CMP [num], 0: Memòria(num) – 0 = Memòria(4h) – 0 = 12 – 0 = 12

Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

JLE end_w: (no hi ha cap bit de resultat actiu i el JLE no saltaria).

Les instruccions de salt no modifiquen els bits de resultat i queden com estaven.

ADD R1, [num]: R1 = R1 + Memòria(num)= 0 + Memòria(4)= 0 + 12 = 12

R1=12; Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

DEC [num]: [num] = [num] – 1 = Memòria(num) - 1 = Memòria(4) - 1 = 12-1 = 11

[num] = 11; Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

JMP while: (salta sempre a l'etiqueta especificada i es torna a executar la instrucció que hi ha la etiqueta 'while' CMP [num], 0).

Loop: (2)

while: CMP [num], 0: Memòria(num) – 0 = Memòria(4h) – 0 = 11 – 0 = 11

Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

JLE end_w: (no hi ha cap bit de resultat actiu i el JLE no saltaria).

Les instruccions de salt no modifiquen els bits de resultat i queden com estaven.

ADD R1, [num]: R1 = R1 + Memòria(num)= 0 + Memòria(4)= 12 + 11 = 23

R1=23; Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

DEC [num]: [num] = [num] – 1 = Memòria(num) - 1 = Memòria(4) - 1 = 11-1 = 10

[num] = 10; Z=0, S=0, C=0, V=0 (no s'activa cap bit de resultat).

JMP while: (salta sempre a l'etiqueta especificada i es torna a executar la instrucció que hi ha la etiqueta 'while' CMP [num], 0).

...

A cada iteració del bucle canvien R1 i [num]. Farem una llista amb els valors que van prenent cada cop que s'executa la instrucció després de while:

Loop: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

R1: 0, 12, 23, 33, 42, 50, 57, 63, 68, 72, 75, 77, 78

[num]: 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Quan comença la iteració 13, la condició de sortida del bucle canvia:

Loop: (13)

while: CMP [num], 0: Memòria(num) – 0 = Memòria(4h) – 0 = 0 – 0 = 0

Z=1, S=0, C=0, V=0 (s'activa el bit de zero).

JLE end_w: (com hi ha el bit de zero actiu, el JLE saltaria a end_w).

Les instruccions de salt no modifiquen els bits de resultat i queden com estaven.

end_w: MOV [sum], R1: [sum] = Memòria(sum) = Memòria(8h) = R1 = 78
Les instruccions de transferència no modifiquen els bits de resultat i queden com estaven.

Solució Final:
R1 = 78 = (12+11+10+9+8+7+6+5+4+3+2+1)
M[num] = M[4h] = 0
M[sum] = M[8h] = 78
Z = 1, S = 0, C = 0, V = 0

Exercici Teo.1.2.1

Per a cadascuna de les següents preguntes, suposeu que l'estat inicial del computador (el valor que contenen els registres, posicions de memòria i els bits de resultat abans de l'execució dels fragments de codi de cada pregunta) és el següent:

- Registres: Ri= i*2, per exemple: R0=0, R1=2, R2=4, etc. (excepte el R15 o SP, que té el valor inicial 0)
- Memòria: M(i)=(i+8) para i = 0, 4, 8,… (excepte en les posicions de memòria en las que es troba el codi del programa). La notació M(i) es refereix a la paraula de 4 bytes M(i)..M(i+3) en little endian.
- Bits de resultat del registre d'estat inactius: Z=0, S=0, C=0, V=0
- Registres especials: El valor del PC i del SP no són necessaris per la resolució d'aquest exercici.

Quin serà l'estat del computador després d'executar cada una de les següents instruccions i fragments de codi?

Indiqueu només el contingut dels registres i posicions de memòria que s'hagin modificat a conseqüència de l'execució del codi. Indiqueu el valor final de tots els bits de resultat. En aquest exercici no és necessari saber el que ocupa cada una de les instruccions en llenguatge màquina, ni us demanem el valor del PC en executar el codi i no es necessari saber el seu valor per fer els exercicis.

Pregunta a)

```
ADD R5, R6
```

Valors inicials:
R5 = 5 * 2 = 10, R6 = 6 * 2 = 12

Operació:
R5 = R5+R6 = 10 + 12 = 22d = 16h

Resultat:
R5 = 22d, Z = 0, S = 0, C = 0, V = 0
Z=0, perquè el resultat 10 + 12 és diferent de 0
S=0, perquè el resultat de 32 bits és positiu, i no té signe
C=0, perquè l'operació 10 + 12 no genera transport
V=0, perquè la operació 10 + 12 no genera sobreeiximent

Pregunta b)

```
SUB [A13F00FCh], R6
```

Valors inicials:
R6 = 6 * 2 = 12
Adreça de memòria: A13F00FCh,
[A13F00FCh] Contingut de l'adreça A13F00FCh= Memòria(A13F00FCh)= A13F00FCh+8 = A13F0104h

Operació:
Memòria(A13F00FCh) – R6 = A13F0104h – 12 = A13F0104h – 0Ch = A13F00F8h

Resultat:
[A13F00FCh] = Memòria(A13F00FCh) = A13F00F8h
Z = 0, S = 1, C = 0, V = 0
El bit S s'activa, perquè el bit més significatiu del resultat (bit 31) és 1
A13F00F8h = **1**010 0001 0011 1111 0000 0000 1111 1000b

Pregunta c)

```
AND R6, FFFFFFF8h
```

Valors inicials:
FFFFFFF8h = 1111 1111 1111 1111 1111 1111 1111 1000b
R6 = 6*2 = 12d = 0Ch = 0000 0000 0000 0000 0000 0000 0000 1100b

Operació:
0000 0000 0000 0000 0000 0000 0000 1100b (000000Ch)
1111 1111 1111 1111 1111 1111 1111 1000b (FFFFFFF8h) operació AND

0000 0000 0000 0000 0000 0000 0000 1000b (0000008h)

Resultat:
R6 = 00000008h = 08h =8d, Z = 0, S = 0, C = 0, V = 0
(no s'activa cap bit de resultat)

Pregunta d)

```
MOV [256], 3FA0h
```

Valors inicials:
3FA0h = 00003FA0h
[256] = 256+ 8 =264 (aquest valor no és necessari)

Resultat:

[256] = Memòria (256d) = 3FA0h, Z = 0, S = 0, C = 0, V = 0
(com que no es fa cap operació, els bits de resultat no es modifiquen i el valor original (264) de [256] es perd).

Pregunta e)

```
CMP R2, R7
```

Valors inicials:
R7 = 7 * 2 = 14 =0Eh = 0000000Eh
R2 = 2 * 2 = 4 = 04h = 00000004h

Operació:
R2-R7 = 4 - 14 = -10
00000004h
0000000Eh (CMP: resta)

FFFFFFF6h = (-10)

Resultat final:
R2 i R7 no es modifiquen, Z = 0, S = 1, C = 1, V = 0
Z=0, perquè el resultat -10 és diferent de 0
S=1, perquè el resultat de 32 bits és negatiu, té signe.
C=1, perquè la operació 4 - 14 ha genera transport final
V=0, perquè la operació 4 - 14 no genera sobreeiximent

Exercici Teo.1.2.2

Escriviu un programa en llenguatge assemblador CISCA que tingui la funcionalitat que es descriu mitjançant un text en cada una de les següents preguntes.

Pregunta a)

Escriviu 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, escriviu un 0.

Solució:

Codi C:

```
if ( (R2>=R3) and (R4 < R5) ) R1=1; else R1=0;
```

Codi assemblador:

```
            MOV   R1, 0          ;else: condició per defecte.
if:         MOV   R1, 0
            CMP   R2, R3
            JL    endif          ; Condició Negada  ( L: Lower than )
            CMP   R4, R5
            JGE   endif          ; Condició Negada  ( GE: Greater or Equal )
then        MOV   R1, 1
endif:
```

Pregunta b)

Escriviu en R1 un 1 si es compleix una i només una de les dues condicions següents:
- el contingut de R2 es més gran o igual que el de R3
- el contingut de R4 es més petit que el de R5.

En qualsevol altre cas escriviu en R1 un 0.

Solució:
Si només s'ha de complir una condició, la clau es fer servir la operació XOR

Codi C:

```
if ( (R2>=R3) xor (R4 < R5) ) R1=1; else R1=0;
```

Codi assemblador:

```
Cond1:    MOV   R1, 1        ; codificar en R1 la primera condició ( 0: fals,  1: cert )
          CMP   R2, R3
          JGE   Cond2
          MOV   R1,0
Cond2:    MOV   R7, 1        ; codificar en R7 la segona condició ( 0: fals,  1: cert )
          CMP   R4, R5
          JL    evalConds
          MOV   R7, 0

evalConds: XOR  R1, R7       ; aplicar la operació XOR a les dues condicions
```

Exercici Teo.1.2.3

Escriviu un programa en llenguatge assemblador CISCA que tingui la funcionalitat que es descriu mitjançant un codi C en cada una de les següents preguntes.

Pregunta a)

Codi C:

```
if (A>B)  C = C + 3 else C = C – 1;
```

Codi assemblador:

```
if:       MOV R1, [A]      ; la etiqueta no és necessària, es fa servir per claredat
          CMP R1, [B]
          JLE else         ; JLE = Lower or Equal  (condició negada!)
then:     ADD [C], 3       ; part de codi del THEN
          JMP endif        ; saltem la part de codi del ELSE
else:     SUB [C], 1       ; part de codi del ELSE
endif:
```

Es podria fer servir qualsevol altre nom per a les etiquetes, però és preferible fer servir noms que indiquin de forma mnemotècnica l'objectiu del codi que ve a continuació.

Pregunta b)

Codi C:

Càlcul del Màxim Comú Divisor per l'algorisme d'Euclides

```
while (A != B) {
  if (A > B)  A = A – B;
    else   B = B – A;
}
MCD = A;
```

Codi assemblador:

```
            MOV R1, [A]
            MOV R2, [B]

while       CMP R1, R2
            JE endwhile          ; La negació de ( A != B ) és ( A == B )
if:         CMP R1, R2          ; Aquest CMP no és necessari fer-lo, ja l'hem fet i
            JLE else             ; no es modifiquen els registres R1 i R2.
            SUB R1, R2
            JMP while
else:       SUB R2, R1
            JMP while

endwhile:   MOV [MCD], R1
            MOV [A], R1
            MOV [B], R2
```

Pregunta c)

Codi C:

Càlcul del terme 'n-èssim' (Sn) de la successió de Fibonacci. S0=0, S1=1, Sn=Sn-1+Sn-2.
Sn: next; Sn-1: second; Sn-2: first.

```
first = 0;
second = 1;
i = 1;
do {

  if (n<2) then {
    next=n;
  }
  else {
    next   = first+second;
    first  = second;
    second = next;
  }
```

```
    i++;

} while (i<n);
```

Codi assemblador:

```
        MOV   R1, 0
        MOV   R2, 1
        MOV   R0, [n]
        MOV   R10,1          ;on guardarem 'i'.
                             ;R3: next; R2: second; R1: first.
do:                          ;do: i if: són realment la mateixa etiqueta.
    if:     CMP   R0, 2
            jge   else
    then:   MOV   R3, R0
            JMP   endif
    else:   MOV   R3, R1
            ADD   R3, R2
            MOV   R1, R2
            MOV   R2, R3
    endif:  INC   R10

while:      CMP   R10, R0
            JL    do

        MOV   [next], R3
        MOV   [second], R2
        MOV   [first], R1
```

Pregunta d)

Codi C:

Càlcul de power = b^e (b^e)

```
power=1;
for (i=e; i>0; i--) {
   power = power * b;
}
```

Codi assemblador:

```
            MOV   R0, 1          ;on guardarem power.
            MOV   R1, [b]        ;agafem la base.
            MOV   R2, [e]        ;agafem l'exponent, R2 farà d'índex 'i'.

            CMP   R2, 0
            JLE   endfor         ;si e=0, b^0=1, no cal fer res.
for:        MUL   R0, R1         ;R0=R0*R1
            DEC   R2
            JG    for            ;Salta en funció dels bits de resultat del DEC.

endfor:     MOV   [power], R0
```