# Problemes de la Màquina Rudimentària

Fermín Sánchez Carracedo
Anna Maria del Corral González
Enric Pastor Llorens

# Problemes de llenguatge màquina

1. Traduïu a llenguatge assemblador de la MR les següents instruccions expressades en hexadecimal.

   a) 1006h
   b) 4E20h
   c) 80A1h
   d) EA64h
   e) C6F1h
   f) E846h
   g) A825h

2. Passeu a notació binària i hexadecimal les següents instruccions escrites en llenguatge assemblador de la MR.

   a) ADD R0, R1, R2
   b) SUBI R4, #1, R5
   c) BGE 43
   d) BR 12
   e) AND R1, R7, R6
   f) ASR R4, R2

3. Indiqueu quin és el resultat d'executar les instruccions següents, donant el contingut final dels registres i posicions de memòria que es modifiquen i el valor dels indicadors de condició. Se suposa per a cada cas que el contingut inicial dels registres i posicions de memòria és el següent:

   R1=2454h
   R2=5656h
   R3=FFFFh
   R4=0000h
   R5=0002h

   Memòria

| Adreça | Contingut |
|---|---|
| 00h | 0339h |
| 01h | 63AFh |
| 02h | EA00h |
| 03h | 3304h |
| 04h | 7834h |
| 05h | 54AAh |

   a) SUB R1,R2,R5
   b) ADDI R3, #1, R1
   c) LOAD 3(R5), R1
   d) STORE R4, 1(R4)
   e) ASR R2, R7

4. Indiqueu quina operació realitza la següent seqüència d'instruccions:
   D024h
   C864h
   D844h

5. La instrucció *SWAP a, b* és present en alguns llenguatges d'alt nivell. Aquesta instrucció intercanvia els valors de les variables *a* i *b*. Escriviu en llenguatge màquina de la MR un programa que es comporti com aquesta instrucció, suposant que les variables *a* i *b* són:
   a) la variable *a* en la posició de memòria 60h i la variable *b* en el registre R2 de la Unitat de Procés.
   b) la variable *a* en la posició de memòria 25h i la variable *b* en la posició 1Dh.

6. Escriviu en llenguatge màquina de la MR un programa que compti el nombre de bits que valen 1 en un nombre NUM emmagatzemat en memòria. Suposeu que el nombre NUM és a l'adreça 03h i que les instruccions s'emmagatzemen a partir de l'adreça 04h.

7. Escriviu en llenguatge màquina de la MR un programa que realitzi la divisió per $2^n$ d'un nombre NUM. L'exponent $n$ és un nombre positiu més petit que 16. Utilitzeu desplaçaments per a realitzar la divisió, i suposeu que:

   - el nombre NUM és a l'adreça 13h,
   - l'exponent $n$ és a l'adreça 17h,
   - les instruccions s'emmagatzemen a partir de l'adreça 2Ah.

8. Escriviu en llenguatge màquina de la MR un programa que indiqui si un nombre NUM és parell o senar. El nombre NUM és en la posició 03h de memòria. Utilitzeu la instrucció AND per a esbrinar el valor del bit de menys pes (0 parell, 1 senar).
   El programa ha de posar un 1 en la posició de memòria 02h si el nombre és senar, i un 0 si és parell. Suposeu que les instruccions estan emmagatzemades a partir de l'adreça 04h.

9. Traduïu el programa següent, escrit en llenguatge màquina de la MR (hexadecimal), a llenguatge assemblador. Indiqueu quina operació realitza, suposant que la primera instrucció executable és en l'adreça 02h i que en les posicions 00h i 01h hi ha dades.

   Memòria

| Adreça | Contingut |
|---|---|
| 00h | 0009h |
| 01h | XXXXh |
| 02h | 0800h |
| 03h | D004h |
| 04h | D824h |
| 05h | 8809h |
| 06h | D224h |
| 07h | DB09h |
| 08h | 8005h |
| 09h | 5001h |

10. Indiqueu quina operació realitza la següent seqüència d'instruccions

    CA24h
    D145h
    C945h

11. Donat el següent fragment de programa, escrit en un llenguatge d'alt nivell, traduïu-lo a llenguatge màquina de la MR suposant que està emmagatzemat a partir de l'adreça de memòria 00h.

    ***si*** (R1 > R2) ***llavors*** executar la instrucció de l'adreça 25h
    ***si no*** continuar executant en seqüència
    ***fsi***

12. Donat el següent fragment de programa, escrit en un llenguatge d'alt nivell, traduïu-lo a llenguatge màquina de la MR suposant que està emmagatzemat a partir de l'adreça de memòria A0h.

    ***si*** (R1 > R2) ***llavors*** executar la instrucció de l'adreça 25h
    ***si no*** executar la instrucció de l'adreça 64h
    ***fsi***

13. Donat el següent fragment de programa, escrit en un llenguatge d'alt nivell, traduïu-lo a llenguatge màquina de la MR suposant que està emmagatzemat a partir de l'adreça de memòria 38h.

    ***si*** ((R1 > R2) *o* (R3 < 0))
    ***llavors*** executar la instrucció de l'adreça 25h
    ***sinó*** executar la instrucció de l'adreça 64h
    ***fsi***

14. Donat el següent fragment de programa, escrit en un llenguatge d'alt nivell, traduïu-lo a llenguatge màquina de la MR suposant que està emmagatzemat a partir de l'adreça de memòria 22h.

    ***mentre*** (R1 > R2) ***i*** (R3 <= R4) ***fer***
    R1 := R1 – 1;

```
        R3 := R3 + 1;
fmentre
```

# Problemes de la Unitat de Procès

15. Indiqueu com afectaria als següents elements de la MR el fet d'augmentar de 8 a 16 el nombre de registres del Banc de Registres.

    a) Grandària de la instrucció.
    b) Amplària de la memòria.
    c) Busos interns de dades de la Unitat de Procés.
    d) Registres interns de la Unitat de Procés.

16. Indiqueu com afectaria als següents elements de la MR el fet d'augmentar la grandària de la memòria fins a tenir una memòria de 64 K paraules en comptes de 256 paraules.

    a) Registre Comptador de Programa (PC).
    b) Registre d'adreces (R@).
    c) Grandària de la instrucció (Registre IR) i amplària de la memòria.
    d) Amplària del Banc de Registres i busos interns.

17. Indiqueu quines modificacions s'haurien de fer als següents elements de la MR si es vol treballar amb nombres a complement a dos de 32 bits, en comptes de fer-ho amb nombres de 16 bits.

    a) La UAL
    b) La memòria
    c) El Banc de Registres
    d) Els busos de dades interns de la Unitat de Procés
    e) Els registres interns de la Unitat de Procés

18. Realitzeu el disseny de la Unitat d'Avaluació de la Condició de Salt.

19. Redissenyeu la UAL per tal que, a més a més dels indicadors de condició Z i N, també calculi els indicadors:
    - V: sobreeiximent per a enters
    - C: sobreeiximent per a naturals

20. Contesteu els apartats següents:
    a) Redissenyeu la UAL perquè un únic bloc combinacional realitzi les operacions de suma i de resta que necessiten les instruccions ADD, SUB, ADDI i SUBI.
    b) Cal modificar la codificació de les instruccions?

21. Volem substituir la instrucció ASR de la MR per una instrucció que compari dos operands font (*CMP Rf1, Rf2*) i modifiqui l'indicador de condició Z tal com s'indica a continuació:
    - Z=1 si els dos operands són iguals
    - Z=0 si els dos operands són diferents

    L'indicador de condició N pot prendre qualsevol valor.

    a) Dissenyeu la nova UAL i redissenyeu la circuiteria que envolta la UAL.
    b) Quines implicacions té el fet d'intercanviar la instrucció ASR per la instrucció CMP en el format de les instruccions?

22. Suposant que el temps de resposta dels diferents elements de la Unitat de Procés és el següent:

    - 10 ns. per a cada multiplexor i descodificador
    - 50 ns. per al sumador d'adreces i l'incrementador connectat a l'entrada del PC
    - 40 ns. per al bloc que realitza l'avaluació de la Condició de Salt
    - 20 ns. per a llegir un registre del Banc de Registres
    - 100 ns. el temps de resposta de la UAL
    - 100 ns. el temps necessari per a llegir o escriure una dada a la memòria RAM

    Contesteu les preguntes següents, sense tenir en compte el temps que pugui tardar a reaccionar la Unitat de Control.

    a) Calculeu el temps necessari per a executar cadascuna de les fases de cada instrucció (cada estat del graf d'estats reduït).
    b) Indiqueu la freqüència màxima de rellotge a la qual pot funcionar la MR.

23. Suposant que s'inclou un registre RMEM entre el multiplexor SELDAT i la memòria, que no afecta el camí entre la sortida de la memòria i el registre IR, contesteu els apartats següents:

    a) Dissenyeu el nou graf d'estats per a executar les instruccions de la MR de forma que s'optimitzi el nombre d'estats necessari per a executar la instrucció LOAD. Indiqueu la taula de sortides dels nous estats
    b) Contesteu els apartats a) i b) del problema 22 usant el nou graf d'estats.
    c) Quina millora representa (en temps d'execució) el fet d'incloure el registre RMEM a la Unitat de Procés?

24. Volem dissenyar una màquina amb el mateix joc d'instruccions que la MR.

    a) Proposeu una codificació d'instruccions suposant que la màquina:

        - té 16 registres de propòsit general de 32 bits.
        - treballa amb nombres en complement a dos de 32 bits.
        - té una memòria de $2^{22}$ paraules de 32 bits.

        Quina és la grandària màxima de l'operand immediat?

    b) Quines modificacions s'haurien de fer a la resta de la Unitat de Procés?
    c) Com es pot augmentar el nombre d'instruccions aritmètiques fins a 64? Quines repercussions tindria aquest augment sobre la resta de paràmetres de les instruccions?
    d) Es podrien afegir més instruccions de salt sense modificar la resta de formats de les instruccions? Quantes i quines suggeriríeu, suposant que a la Unitat de Procés s'afegissin els indicadors de condició C (transport) i V (sobreeiximent)?

25. Contesteu les preguntes següents:

    a) Dissenyeu un Banc de Registres que tingui dos ports de lectura. El fet que un Banc de Registres tingui dos ports de lectura implica que es poden llegir dos registres al mateix temps.
    b) Quins canvis comportaria en el disseny de la Unitat de Procés de la MR substituir el seu Banc de Registres pel que es proposa en l'apartat a)? Hauria de canviar el format de les instruccions?

26. Volem afegir al joc d'instruccions de la MR una instrucció aritmètica que multipliqui per dos un operand font emmagatzemat en un registre i deixi el resultat en un registre destinació (*MULDOS Rf, Rd*). Volem una instrucció específica (aquesta operació es pot fer amb la instrucció *ADD Rf,Rf,Rd*, però no volem aquesta solució).

    a) Quines implicacions té per al format de les instruccions el fet d'afegir aquesta instrucció?
    b) Dissenyeu la nova UAL i redissenyeu la circuiteria que envolta la UAL.

# Problemes de la Unitat de Control

27. Donat el programa següent, escrit en llenguatge assemblador de la MR:

```
00h BR 03h
01h SUB R0, R0, R0
02h BR 05h
03h ADD R0, R0, R0
04h BGE 01h
```

Contesteu les preguntes següents:

a) En quin ordre s'executen les instruccions?
b) Quants cicles tarda a executar-se el programa (suposant el graf més simplificat per a la Unitat de Control)?
c) Indiqueu els diferents valors que pren el PC en cada una de les fases d'execució de les instruccions durant l'execució del programa.
d) Compareu la seqüència d'adreces emmagatzemades al PC amb la seqüència d'adreces corresponent a l'execució d'instruccions. Es corresponen totes dues seqüències? Justifiqueu la resposta.

28. Donat el bucle següent, escrit en llenguatge assemblador de la MR:

```
00h SUB R3, R2, R0
01h BL 08h
02h SUB R4, R2, R0
03h BNE 06h
04h ADDI R2, #1, R2
05h BR 07h
06h SUB R4, #2, R4
07h BG 00h
```

Suposant que inicialment R3 = $100_{10}$ i R2 = $49_{10}$, contesteu els apartats següents:

a) Descriviu què fa el cos del bucle.
b) Quants cops s'executa el bucle?

29. Que indiquen a la Unitat de Procés de la MR aquests senyals de control?

a) Ld_PC = 1, Ld_IR = 1 i PC/@ = 0
b) Ld_PC = 1, Ld_IR = 1 i PC/@ = 1
c) Ld_RA = 1 i CRf = 1
d) ERd = 1
e) Ld_RZ = 1, Ld_RN = 1, ERd = 1, OPERAR = 1 i CRf = 2

30. És vol dissenyar la Unitat de Control de la MR com un sistema lògic seqüencial. La funció G (transició d'estats) es realitza de forma mínima a tres nivells, i la funció de sortida (H) es realitza amb una memòria ROM. Dibuixeu la taula de transició d'estats i indiqueu el contingut i grandària de la ROM.

31. Contesteu els apartats següents:

a) Per què s'ha de realitzar la descodificació de la instrucció?
b) Per què s'intenta reduir el nombre d'estats de la Unitat de Control de la MR?

32. Si el Banc de Registres de la Unitat de Procés de la MR tingués dos ports de lectura (vegeu el problema 25):

a) Podria ser més curt el temps d'execució d'alguna instrucció?
b) Milloraria el rendiment global de la MR durant l'execució d'instruccions?

33. Si la UAL de la MR, a més a més dels indicadors de condició Z i N, també calculés els indicadors

- V: sobreeiximent per a enters
- C: sobreeiximent per a naturals

seria interessant tenir més instruccions de salt.

a) Quines instruccions de salt es podrien afegir?
b) Caldria canviar el format de les instruccions?

# Problemes bàsics de Llenguatge assemblador

34. Traduïu el programa següent, escrit en un llenguatge d'alt nivell, a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que:

- s'emmagatzema a partir de l'adreça 00h,
- les variables *a* i *b* són a les posicions de memòria 00h i 01h respectivament,
- la primera instrucció executable del programa és a l'adreça 02h,
- la instrucció *swap a, b* intercanvia els valors de les variables *a* i *b.*

```
programa exemple1;
var a, b: enter;
a := 13;
b := 16;
mentre (a > 10) fer
        a := a –1;
        b := b +2;
fmentre;
si (a < b)      llavors swap (a, b)
                si no b := a –1
fsi;
fprograma.
```

35. Traduïu el programa següent, escrit en un llenguatge d'alt nivell, a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que s'emmagatzema a partir de l'adreça 00h.

```
programa exemple2;
var a, b, c : enter;
a := 13;
b := 1;
c := 0;
mentre (a > 10) o (b < 20) fer
        si ( a < 11 ) i ( b > 18 ) llavors c := c + 3 fsi;
        a := a – b;
fmentre;
fprograma.
```

36. El programa següent calcula el màxim comú divisor de dos nombres *a* i *b* segons l'algorisme de residus d'Euclides:

```
programa exemple3;
var a = 5, b = 15, mcd: enter;
mentre (a <> b) fer
        si (a > b)      llavors a := a – b
                        si no b := b – a
        fsi;
fmentre;
mcd := a;
fprograma.
```

a) Feu-ne la traducció a llenguatge assemblador de la MR.
b) Traduïu-lo a llenguatge màquina, expressant-lo tant en binari com en hexadecimal, suposant que:

- s'emmagatzema a partir de l'adreça 0Fh,
- les variables *a* i *b* s'emmagatzemen a les posicions 0Fh i 10h respectivament,
- la variable *mcd* s'emmagatzema a la posició 11h,
- la primera instrucció executable del programa és a l'adreça 12h.

# Problemes avançats de Llenguatge assemblador

37. Escriviu en un llenguatge d'alt nivell un programa que calculi el quadrat d'un nombre positiu *a* mitjançant la suma acumulada *a* vegades. Feu-ne la traducció a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que s'emmagatzema a partir de l'adreça 00h.

38. L'algorisme següent, descrit en un llenguatge d'alt nivell, calcula el factorial d'un nombre *a*. Feu-ne la traducció a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que s'emmagatzema a partir de l'adreça 00h. Indiqueu el contingut de la taula de símbols.

```
Programa FACTORIAL;
var n, ind1, ind2, asumar, acum, fact: enter;
n:= 4;
ind1:= n;
asumar := n;
acum:= 0;
mentre ( ind1 > 2) fer
        ind2 := ind1 - 1;
        mentre ( ind2 > 0 ) fer
                acum := acum + asumar;
                ind2 := ind2 – 1;
        fmentre;
        asumar := acum;
        acum := 0;
        ind1 := ind1 - 1;
fmentre;
fact := asumar;
fprograma.
```

39. Escriviu en un llenguatge d'alt nivell un programa que calculi el nombre de lletres "*a*" d'una frase acabada en punt. Traduïu el programa a llenguatge assemblador i llenguatge màquina de la MR, suposant que:

- El programa es troba emmagatzemat a partir de l'adreça 80h.
- La frase es troba emmagatzemada a partir de l'adreça 02h en posicions consecutives de memòria.
- Cada lletra ocupa una posició de memòria i està codificada utilitzant el codi ASCII. En aquest sistema de codificació, cada símbol es codifica amb 7 bits; en aquest cas, en els 7 bits de menys pes de la posició de memòria que ocupa. Els 9 bits de més pes són a 0. La lletra "*a*" es codifica mitjançant el nombre 61h, i el caràcter "." mitjançant el nombre 2Eh.
- Els valors 61h i 2Eh es poden emmagatzemar a les adreces 00h i 01h. La frase mai no té més de 125 lletres, i sempre acaba en ".".

40. Escriviu en un llenguatge d'alt nivell un programa que calculi la suma dels elements d'un vector al mateix temps que busca els elements màxim i mínim. Feu la traducció a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que les dades s'emmagatzemen a partir de l'adreça 00h i les instruccions a partir de l'adreça 3Ah.

41. Escriviu en un llenguatge d'alt nivell un programa que sumi, element a element, dos vectors A i B, deixant el resultat en un tercer vector C (C[i] := A[i] + B[i]). La grandària dels vectors és de 10 elements. Traduiu el programa a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que els vectors s'emmagatzemen a partir de l'adreça de memòria 00h i el programa s'emmagatzema a partir de l'adreça 80h.

42. Escriviu en llenguatge assemblador de la MR les macros que permeten d'executar les instruccions següents, pertanyents als jocs d'instruccions d'altres computadors:

| | | |
|---|---|---|
| a) | clr Rd | Posa un 0 al registre destinació |
| b) | clr A(Ri) | Posa un 0 a la posició de memòria M[A+(Ri)] |
| c) | inc Rd | Incrementa en una unitat el valor del registre Rd |
| d) | dec Rd | Decrementa en una unitat el valor del registre Rd |

e) add3 Rf1, Rf2, Rd — Suma el contingut de 3 registres del Banc de Registres i deixa el resultat en un d'ells (Rd)
f) asl Rd — Desplaça un bit a l'esquerra el registre Rd
g) mov Rf, Rd — Copieu el valor del registre Rf en el registre Rd
h) mov Rf, A(Ri) — Copia el valor del registre Rf en la posició de memòria M[A+(Ri)]
i) mov A(Ri), B(Rj) — Copia el valor de la posició de memòria M[A+(Ri)] en la posició de memòria M[B+(Rj)]
j) swap Rf1, Rf2 — Intercanvieu els valors dels registres Rf1 i Rf2
k) swap A(Ri), B(Rj) — Intercanvia els valors de les posicions de memòria M[A+(Ri)] i M[B+(Rj)]
l) asrn #n, Rd — Desplaça *n* bits a la dreta el registre Rd. El nombre *n* ha d'ésser més gran que 0 i més petit que 16
m) asln #n, Rd — Desplaça *n* bits a l'esquerra el registre Rd. El nombre *n* ha d'ésser més gran que 0 i més petit que 16
n) inc A(Ri) — Incrementa el contingut de la posició de memòria M[A+(Ri)]

43. Escriviu en un llenguatge d'alt nivell un programa que calculi la suma dels elements d'un vector V que compleixen el fet d'ésser més grans que un cert valor MÍNIM i més petits que un cert valor MÀXIM (MÍNIM < V[i] < MÀXIM). Feu-ne la traducció a llenguatge assemblador de la MR i posteriorment a llenguatge màquina, suposant que el programa s'emmagatzema a partir de l'adreça 3Ah.

44. Traduïu a llenguatge assemblador de la MR el programa següent, escrit en un llenguatge d'alt nivell. El programa calcula el factorial d'un nombre "a" (en l'exemple, a=8):

```
programa FACTORIAL;
var a, cont, factorial: enter;
a:= 8;
cont := 1;
factorial := 1;
mentre ( cont < A ) fer
        cont := cont + 1;
        factorial := factorial * cont;
fmentre;
fprograma.
```

a) Dissenyeu i useu la macro *mul $1, $2, $3*, que multiplica el contingut de dos registres ($1, $2) i deixa el resultat en un tercer registre ($3). Els tres registres han d'ésser diferents.
b) Feu el preassemblatge (expansió de macros) i el postassemblatge (creació de la taula de símbols i generació de codi) del programa, suposant que el programa s'emmagatzema a partir de l'adreça de memòria 00h.

45. Donat el programa següent, escrit en llenguatge assemblador de la MR:

```
N:      .DW 4
RESUL:.RW 1
        .DEF cmp $1, $2
                SUB $1, $2, R0
        .ENDDEF
        .DEF cmp $1, $i2
                SUBI $1, $i2, R0
        .ENDDEF
        .DEF clr $1
                ADD R0, R0, $1
        .ENDDEF
        .DEF mov $1, $2
                ADD R0, $1, $2
        .ENDDEF
        .DEF mov $i1, $2
                ADDI R0, $i1, $2
        .ENDDEF
        .DEF mov $d1, $2
                LOAD $d1, $2
        .ENDDEF
        .DEF mov $1, $d2
                STORE $1, $d2
```

```
        .ENDDEF
        .DEF inc $1
                ADDI $1, #1. $1
        .ENDDEF
        .DEF dec $1
                SUBI $1, #1. $1
        .ENDDEF
        .BEGIN INI
INI:    MOV N(R0), R1
        MOV R1, R3
        clr R4
M1:     cmp R1, #2
        BLE FM1
        SUBI R1, #1, R2
M2:     cmp R2, #0
        BLE FM2
        ADD R4, R3, R4
        dec R2
        BR M2
FM2:    mov R4, R3
        clr R4
        dec R1
        BR M1
FM1:    mov R3, RESUL(R0)
        .END
```

a) Realitzeu-ne el preassemblatge (expansió de macros).
b) Realitzeu-ne el postassemblatge (generació de la taula de símbols i traducció a llenguatge màquina) suposant que el programa s'emmagatzema a partir de la posició 00h de memòria.
c) Traduïu el programa a alt nivell, indicant quines operacions realitza. Per a fer això, intenteu analitzar quines operacions executa cada macro.

# Solucionari

**1.**

a) 1006h= 0001 0000 0000 0110= 00 010 000 00000110= **LOAD 6(R0), R2**

b) 4E20h= 0100 1110 0010 0000= 01 001 110 00100000= **STORE R1,2Oh(R6)**

c) 80A1h= 1000 0000 10100001= 10 000 000 10100001= **BR A1h**

d) EA64h= 1110 1010 0110 0100= 11 101 010 011 00 100= **ADD R2,R3,R5**

e) C6F1h= 1100 0110 1111 0001= 11 000 110 11110 001= **SUBI R6,#–2,R0**

f) E846h= 1110 1000 0100 0110= 11 101 000 010 00 110= **ASR R2, R5**

g) A825h= 1010 1000 0010 0101= 10 101 000 00100101= **BNE 25h**

**2.**

a) ADD R0,R1,R2= 11 010 000 001 00 100= 1101 0000 0010 0100= **D024h**

b) SUBI R4,#1,R5= 11 101 100 00001 001= 1110 1100 0000 1001= **EC09h**

c) BGE 43= 10 110 000 00101011= 1011 0000 0010 1011= **B02Bh**

d) BR 12= 10 000 000 0000 1100= 1000 0000 0000 1100= **800Ch**

e) AND R1,R7,R6= 11 110 001 111 00 111= 1111 0001 1110 0111= **F1E7h**

f) ASR R4,R2= 11 010 xxx 100 00 110= 1101 0xxx 1000 0110= **D086h** (per exemple)

**3.**

a) SUB R1,R2,R5

La instrucció realitza l'operació R5 := R1 – R2
Després d'executar la instrucció, R5 val 2454h – 5656h = CDFEh. Els registres R1 i R2 no varien. Els indicadors de condició s'activen segons el valor escrit a R5 i, per tant, Z=0 i N=1.

b) ADDI R3, #1, R1

La instrucció realitza l'operació R1 := R3 + 1
Després d'executar la instrucció, R1 val FFFFh + 0001h = 0000h (el sumador de la UAL no té en compte el bit de transport).
El registre R3 no varia.
Els indicadors de condició s'activen segons el valor escrit a R1 i, per tant, Z=1 i N=0.

c) LOAD 3(R5), R1

La instrucció realitza l'operació R1 := M[(R5)+3] = M[2+3] = M[5] = 54Aah
Després d'executar la instrucció, el registre R5 no varia.
Els indicadors de condició s'activen segons el valor escrit a R1 i, per tant, Z=0 i N=0.

d) STORE R4, 1(R4)

La instrucció realitza l'operació M[(R4)+1] := R4; per tant, M[1] := 0000h
Després d'executar la instrucció, la posició de memòria 1 passa a contenir un 0.
El registre R4 no varia.
Els indicadors de condició no varien.

e) ASR R2, R7

La instrucció realitza l'operació R7 := R2 >> 1
Com que R2=0101 0110 0101 0110, el resultat de desplaçar un bit a la dreta mantenint el signe és 0010 1011 0010 1011 = 2B2Bh.

Després d'executar la instrucció, R7 val 2B2Bh.
El registre R2 no varia.
Els indicadors de condició s'activen segons el valor escrit a R7 i, per tant, Z=0 i N=0.

**4.**
La traducció a llenguatge assemblador és la següent:

D024h = 1101 0000 0010 0100 = 11 010 000 001 00 100 = ADD R0, R1, R2
C864h = 1100 1000 0110 0100 = 11 001 000 011 00 100 = ADD R0, R3, R1
D844h = 1101 1000 0100 0100 = 11 011 000 010 00 100 = ADD R0, R2, R3

La seqüència d'instruccions intercanvia el valor dels registres R1 i R3, usant el registre R2 com a registre auxiliar.

R2 := R0 + R1 = R1 (ja que R0=0 sempre)
R1 := R0 + R3 = R3 (per la mateixa raó)
R3 := R0 + R2 = R2

**5.**
Per a realitzar l'intercanvi de dues variables usarem els registres addicionals que considerem necessaris. Els programes en llenguatge màquina són els següents:

a) Usem el registre R1 com a variable auxiliar per a emmagatzemar temporalment el valor llegit de la memòria.

```
LOAD 60h(R0), R1
STORE R2, 60h(R0)
ADD R0, R1, R2
```

b) En aquest cas necessitem els registres R1 i R2 per a emmagatzemar temporalment els dos valores llegits de la memòria.

```
LOAD 25h(R0), R1
LOAD 1Dh(R0), R2
STORE R1, 1Dh(R0)
STORE R2, 25h(R0)
```

**6.**
La solució proposada itera 16 cops sobre el nombre NUM. Per a detectar si el darrer bit és 0 o 1 calcula l'AND entre NUM i la constant 1. El resultat d'aquesta operació es pot acumular directament sobre un registre que emmagatzema el nombre d'uns (1) existents. El bucle també ha d'incrementar l'índex de la iteració, i així mateix desplaçar NUM un bit a la dreta per tal d'analitzar el bit següent.

R1 guarda nombre NUM.
R2 s'usarà com a índex de la iteració.
R3 emmagatzema el nombre d'uns a NUM.
R4 emmagatzema la constant "1".

| Adreça | Instrucció | Contingut | Hexadecimal |
|---|---|---|---|
| 04h | LOAD 03h(R0), R1 | 00 001 000 0000 0011 | 0803 |
| 05h | ADD R0, R0, R2 | 11 010 000 000 00 100 | D004 |
| 06h | ADD R0, R0, R3 | 11 011 000 000 00 100 | D804 |
| 07h | ADDI R0, #1, R4 | 11 100 000 00001 000 | E008 |
| 08h | ADDI R2, #-16, R0 | 11 000 010 10000 000 | C280 |
| 09h | BEQ 0Fh | 10 001 000 00001111 | 880F |
| 0Ah | AND R1, R4, R5 | 11 101 001 100 00 111 | E987 |
| 0Bh | ADD R3, R5, R3 | 11 011 011 101 00 100 | DBA4 |
| 0Ch | ASR R1, R1 | 11 001 000 001 00 110 | C826 |
| 0Dh | ADDI R2, #1, R2 | 11 010 010 00001 000 | D208 |
| 0Eh | BR 08h | 10 000 000 00001000 | 8008 |

**7.**
La solució proposada itera sobre el nombre NUM tants cops com indica l'exponent. A cada iteració el nombre és desplaçat un bit a la dreta per a implementar una divisió per dos.

R1 guarda nombre NUM.
R2 guarda l'exponent N.
R3 emmagatzema l'índex de la iteració.

| Adreça | Instrucció | Contingut | Hexadecimal |
|---|---|---|---|
| 2Ah | LOAD 13h(R0), R1 | 00 001 000 00010011 | 0813 |
| 2Bh | LOAD 17h(R0), R2 | 00 010 000 00010111 | 1017 |
| 2Ch | ADD R0, R0, R3 | 11 011 000 000 00 100 | D804 |
| 2Dh | SUB R2, R3, R0 | 11 000 010 011 00 101 | C265 |
| 2Eh | BGE 32h | 10 110 000 00110010 | B032 |
| 2Fh | ASR R1, R1 | 11 001 000 001 00 110 | C826 |
| 30h | ADDI R3, #1, R3 | 11 011 011 00001 000 | DB08 |
| 31h | BR 2Dh | 10 000 000 00101101 | 802D |

**8.**
La solució proposada realitza l'AND del nombre NUM amb la constant "1". Si el resultat és 1, NUM és un nombre senar; si el resultat és 0, NUM és un nombre parell. Com que aquest valor correspon al valor que s'ha d'emmagatzemar en l'adreça 02h, el resultat de l'operació AND s'emmagatzema directament en aquella posició de memòria.

R1 guarda el nombre NUM.
R2 guarda la constant "1".
R3 emmagatzema el resultat de l'operació AND.

| Adreça | Instrucció | Contingut | Hexadecimal |
|---|---|---|---|
| 04h | LOAD 03h(R0), R1 | 00 001 000 00000011 | 0803 |
| 05h | ADDI R0, #1, R2 | 11 010 000 00001 000 | D008 |
| 06h | AND R1, R2, R3 | 11 011 001 010 00 111 | D947 |
| 07h | STORE R3, 02h(R0) | 01 011 000 00000010 | 5802 |

**9.**
La traducció a llenguatge assemblador és la següent:

0800h= 0000 1000 0000 0000= 00 001 000 00000000=02h LOAD 00h(R0),R1
D004h= 1101 0000 0000 0100= 11 010 000 000 00 100= 03h ADD R0,R0,R2
D824h= 1101 1000 0010 0100= 11 011 000 001 00 100= 04h ADD R0,R1,R3
8809h= 1000 1000 0000 1001= 10 001 000 0000 1001= 05h BEQ 09h
D224h= 1101 0010 0010 0100= 11 010 010 001 00 100= 06h ADD R2, R1,R2
DB09h= 1101 1011 0000 1001= 11 011 011 00001 001= 07h SUBI R3, #1, R3
8005h= 1000 0000 0000 0101= 10 000 000 0000 0101= 08h BR 05h
5001h= 0101 0000 0000 0001= 01 010 000 00000001=09h STORE R2,01h(R0)

La seqüència d'instruccions acumula sobre el registre R3 el valor emmagatzemat a l'adreça 02h tants cops com indica aquesta mateixa adreça. És a dir, calcula el quadrat del valor emmagatzemat en la posició 02h. El resultat s'emmagatzema a l'adreça 01h.

**10.**
La traducció a llenguatge assemblador és la següent:

CA24h = 1100 1010 0010 0100 = 11 001 010 001 00 100 = ADD R2, R1, R1
D145h = 1101 0001 0100 0101 = 11 010 001 010 00 101 = SUB R1, R2, R2
C945h = 1100 1001 0100 0101 = 11 001 001 010 00 101 = SUB R1, R2, R1

La seqüència d'instruccions intercanvia el valor dels registres R1 i R2 sense fer servir cap registre auxiliar. Suposem que R1=x, R2=y:

R1 := R2 + R1 = i + x
R2 := R1 - R2 = (y + x) – i = x
R1 := R1 - R2 = (y + x) – x = y

**11.**

| Adreça | Instrucció | Contingut |
|---|---|---|
| 00h | SUB R1,R2,R0 | 11 000 001 010 00 101 |
| 01h | BG 25h | 10 111 000 0010 0101 |

**12.**

| Adreça | Instrucció | Contingut |
|---|---|---|
| A0h | SUB R1,R2,R0 | 11 000 001 010 00 101 |
| A1h | BLE 64h | 10 011 000 0110 0100 |
| A2h | BR 25h | 10 000 000 0010 0101 |

**13.**

| Adreça | Instrucció | Contingut |
|---|---|---|
| 38h | SUB R1,R2,R0 | 11 000 001 010 00 101 |
| 39h | BG 25h | 10 111 000 0010 0101 |
| 3Ah | SUBI R3,#0,R0 | 11 000 011 00000 001 |
| 3Bh | BL 25h | 10 010 000 0010 0101 |
| 3Ch | BR 64h | 10 000 000 0110 0100 |

**14.**

| Adreça | Instrucció | Contingut |
|---|---|---|
| 22h | SUB R1,R2,R0 | 11 000 001 010 00 101 |
| 23h | BLE 29h | 10 011 000 00101001 |
| 24h | SUB R3,R4,R0 | 11 000 011 100 00 101 |
| 25h | BG 29h | 10 111 000 00101001 |
| 26h | SUBI R1,#1,R1 | 11 001 001 00001 001 |
| 27h | ADDI R3,#1,R3 | 11 011 011 00001 000 |
| 28h | BR 22h | 10 000 000 00100010 |

**15.**

a) Com que disposem de 16 registres, calen 4 bits per a codificar a quin registre es vol accedir a cada moment. Per tant, per a cada registre usat en cada instrucció s'ha d'augmentar en un bit la magnitud de la instrucció.

- Instruccions aritmètiques amb un operand immediat:

  2 bits de CO
  4 bits Rd
  4 bits Rf1
  5 bits operand immediat
  3 bits d'operació

  **TOTAL: 18 BITS**

- Instruccions aritmètiques amb operands a registres:

  2 bits de CO
  4 bits Rd
  4 bits Rf1
  4 bits Rf2
  3 bits de operació

  **TOTAL: 17 BITS**

- Instruccions de salt: els bits 10:8 codifiquen el registre R0 per tal que el salt es faci correctament. Amb un Banc de Registres de 16 registres farien falta 4 bits per a aquest camp.

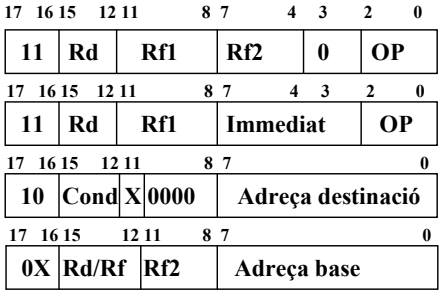
  **TOTAL: 17 BITS**

- Instruccions d'accés a memòria:

  2 bits CO
  4 bits Rf (Store) o Rd (Load)
  4 bits Ri
  8 bits adreça base

  **TOTAL: 18 BITS**

Per tant, la grandària de la instrucció es veuria incrementada fins a 18 bits. Una possible nova codificació per a les instruccions seria la següent:




b) En augmentar la grandària de la instrucció, també caldria augmentar l'amplària de la memòria fins a 18 bits, amb l'objectiu que cada instrucció pogués ser emmagatzemada en una posició de memòria.
Com a efecte lateral, en codificar les instruccions tal com indica la figura s'observa que sobra un bit per a les instruccions de salt i per a les instruccions aritmètiques i lògiques registre-registre. Aquest bit pot ser usat per a incrementar el nombre d'instruccions si s'afegeix al camp *Cond*. Per exemple, es podrien tenir set instruccions que salten suposant que els nombres estan escrits en complement a dos, i unes altres set que saltessin suposant que són nombres naturals, i encara sobrarien dues codificacions. Aquesta és una distinció molt comuna, i present en pràcticament tots els computadors.

c) El bus que va des de la memòria al registre IR hauria d'ésser de 18 bits. Els 16 bits de menys pes es connectarien a les entrades 0 i 1 del multiplexor SELDAT. La resta de busos no es veurien alterats.

d) Solament s'hauria d'augmentar la grandària del registre IR fins a 18 bits.

ACLARIMENT: En les respostes d'aquest problema s'ha considerat que la grandària dels operands continua essent de 16 bits (i no pas de 18). Si els operands fossin de 18 bits s'haurien de canviar les respostes dels apartats c) i d).

**16.**

a) Per a adreçar 64 Kbytes fan falta 16 bits. Per tant, el PC hauria d'ésser de 16 bits.

b) El registre R@ conté l'adreça de salt quan s'executen instruccions de salt o l'adreça d'una dada quan s'executen instruccions d'accés a memòria. En qualsevol cas, i per la mateixa raó que en l'apartat a), ha d'ésser de 16 bits.

c) En augmentar la grandària d'una adreça a 16 bits, cal canviar el format de les instruccions de salt perquè continguin una adreça de 16 bits en comptes d'una adreça de 8 bits (si es vol mantenir l'adreçament absolut). Per tant, aquestes instruccions passarien a tenir 24 bits, cosa que implica que l'amplària de la memòria i del registre d'instrucció seria també de 24 bits.

Les instruccions aritmètiques i lògiques i les d'accés a memòria no es veurien modificades. En el cas de les instruccions d'accés a memòria, l'adreça es calcularia sumant el contingut del registre índex (els seus 16 bits permeten d'accedir a qualsevol posició de memòria) amb els 8 bits de menys pes de la instrucció. De tota manera, i aprofitant que la grandària de la instrucció s'ha incrementat a causa de les instruccions de salt, es podria disposar d'un desplaçament de 16 bits a les instruccions Load i Store.

Una altra possible solució, que no necessitaria efectuar cap canvi a la Unitat de Procés, consistiria a usar el mode relatiu a registre base (base + desplaçament) també a les instruccions de salt en comptes del mode absolut. Això implicaria que l'adreça de salt es calcularia igual que l'adreça d'accés a memòria en una instrucció Load o Store. Per a fer-ho, n'hi ha prou de substituir els bits 10:8 de la instrucció de salt (que contenen un zero) per la codificació del registre índex que s'usaria per a calcular l'adreça de salt. Aquest canvi en la forma de calcular les adreces de salt no requereix cap modificació de la Unitat de Procés. La grandària de la instrucció, i per tant l'amplària de la memòria, continuaria essent de 16 bits.

d) Un registre del Banc de Registres és de 16 bits i pot adreçar fins a 64 Kbytes. Per tant, no es requereix modificar el Banc de Registres. De tota manera, el bus de sortida del Banc de Registres que està connectat amb l'entrada del sumador hauria d'ésser de 16 bits, en comptes de 8 bits.

**17.**

a) La UAL hauria de tenir operadors de 32 bits (un sumador de 32 bits, un restador de 32 bits i 32 portes AND per a la instrucció AND).

b) L'amplària de la memòria hauria d'ésser també de 32 bits, per tal que cada dada pogués estar continguda en una única adreça. Les instruccions poden emmagatzemar-se, per exemple, als 16 bits de menys pes de cada posició de memòria.

c) El Banc de Registres hauria de tenir registres de 32 bits per a emmagatzemar els operands.

d) Tots els busos de dades que es comuniquen amb el Banc de Registres i amb la memòria haurien d'ésser de 32 bits.

e) El registre RA hauria d'ésser de 32 bits per a contenir el primer operand de les instruccions aritmètiques i lògiques. De tota manera, el PC i el R@ continuarien essent de 8 bits, ja que el bus d'adreces de la memòria continua essent d'aquesta grandària. El registre d'instruccions tampoc no es veuria alterat, ja que les instruccions continuen essent de 16 bits.

**18.**

Hi ha diverses opcions per a dissenyar la Unitat d'Avaluació de la Condició de Salt. Vegem com es dissenyaria amb portes lògiques.

El valor del senyal *Cond* es pot expressar amb la funció:

$$Cond = \overline{IR_{13}}.\overline{IR_{12}}.\overline{IR_{11}} + (\overline{IR_{13}}..\overline{IR_{12}}.IR_{11}).Z + (\overline{IR_{13}}.IR_{12}.\overline{IR_{11}}).N + (\overline{IR_{13}}.IR_{12}.IR_{11}).(N+Z) +$$
$$+ (IR_{13}.\overline{IR_{12}}.IR_{11}).\overline{Z} + (IR_{13}.IR_{12}.\overline{IR_{11}}).\overline{N} + (IR_{13}.IR_{\cdot 12}\,IR_{11}).(\overline{N+Z})$$

Simplificant per Karnaugh a dos nivells obtenim l'expressió:

$$Cond = \overline{IR_{12}}.\overline{IR_{11}} + \overline{IR_{13}}.IR_{11}.Z + \overline{IR_{13}}.IR_{12}.N + IR_{13}.\overline{IR_{12}}.\overline{Z} + IR_{13}.\overline{IR_{11}}.\overline{N} + IR_{13}.\overline{N}.\overline{Z}$$

**19.**

**V: sobreeiximent per a enters representats en complement a dos**

El sobreeiximent per a la suma de nombres enters es produeix quan els dos operands que cal sumar són del mateix signe i el resultat és de signe contrari. El sobreeiximent a la resta de nombres enters es produeix quan els dos operands a restar són de diferent signe i el resultat té el signe del subtrahend. Per tant:

$$V = OPERAR.\overline{IR_1}.\overline{IR_0}(A_{15}.B_{15}.\overline{O_{15}} + \overline{A_{15}}.\overline{B_{15}}.O_{15}) + OPERAR.\overline{IR_1}.IR_0(A_{15}.\overline{B_{15}}.\overline{O_{15}} + \overline{A_{15}}.B_{15}O_{15})$$

**C: sobreeiximent per a naturals**

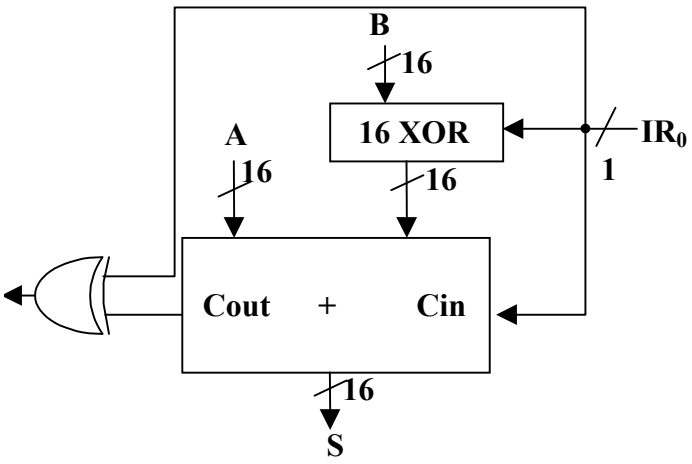
El sobreeiximent a la suma i a la resta de nombres naturals es comprova amb el bit del transport de l'operació realitzada. Per tant:

$$C = OPERAR.\overline{IR_1}.\overline{IR_0}Cout\_suma + OPERAR.\overline{IR_1}.IR_0.Cout\_resta$$

La realització de V i de C es pot fer amb qualsevol dels mètodes estudiats per a dissenyar un sistema lògic combinacional.

**20.**

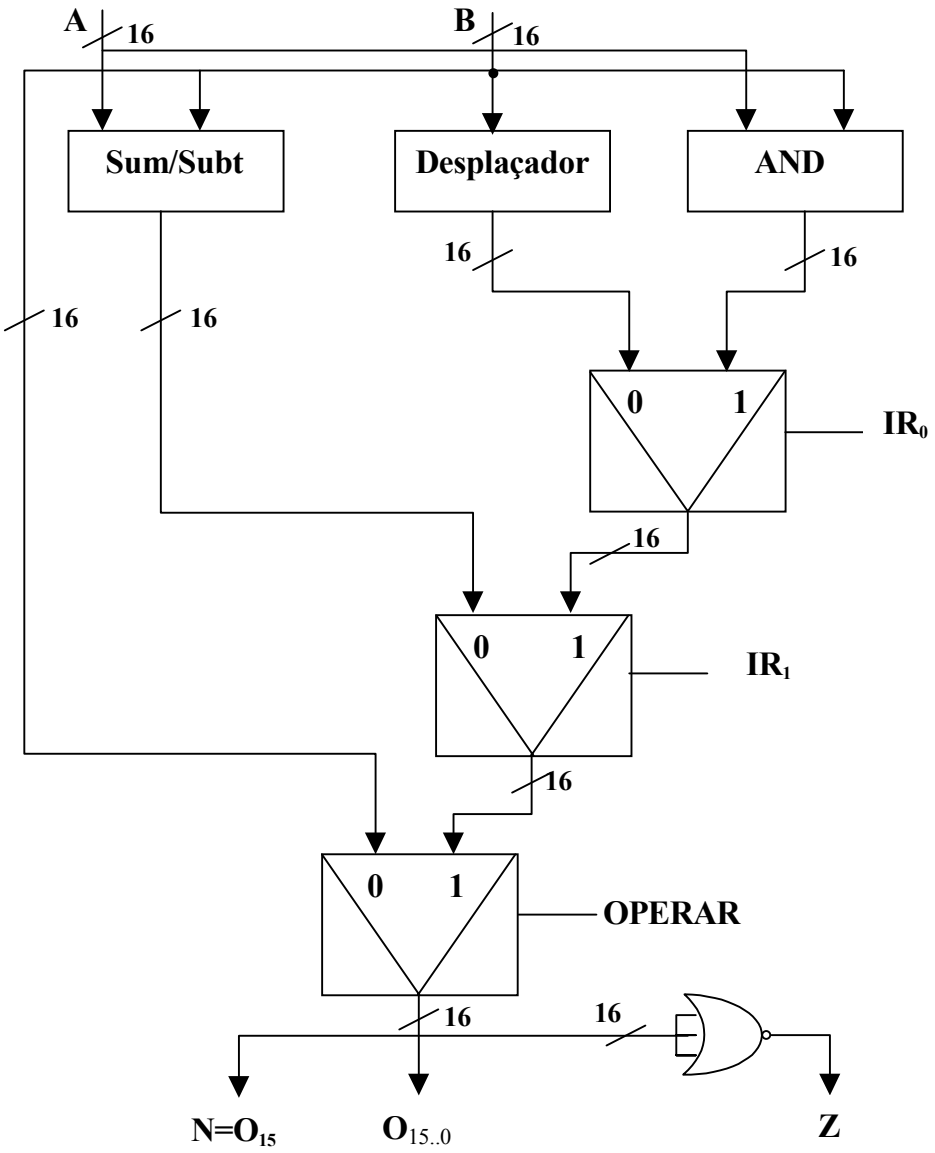
a) El circuit que realitza la Suma/Resta és:




El bit $IR_0$ permet de seleccionar de forma automàtica l'operació a realitzar. Quan val 0 es realitza una suma, i quan val 1 es fa una resta.

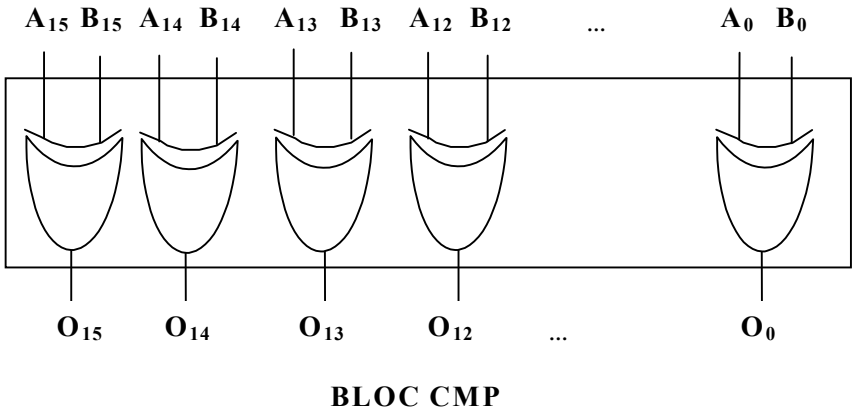
b) El format de les instruccions no ha de canviar, ja que, com s'ha vist en l'apartat a), la selecció de l'operació que cal realitzar al sumador/restador es fa de forma automàtica mitjançant el bit $IR_0$.

El disseny final de la UAL és el que es mostra en la figura següent:



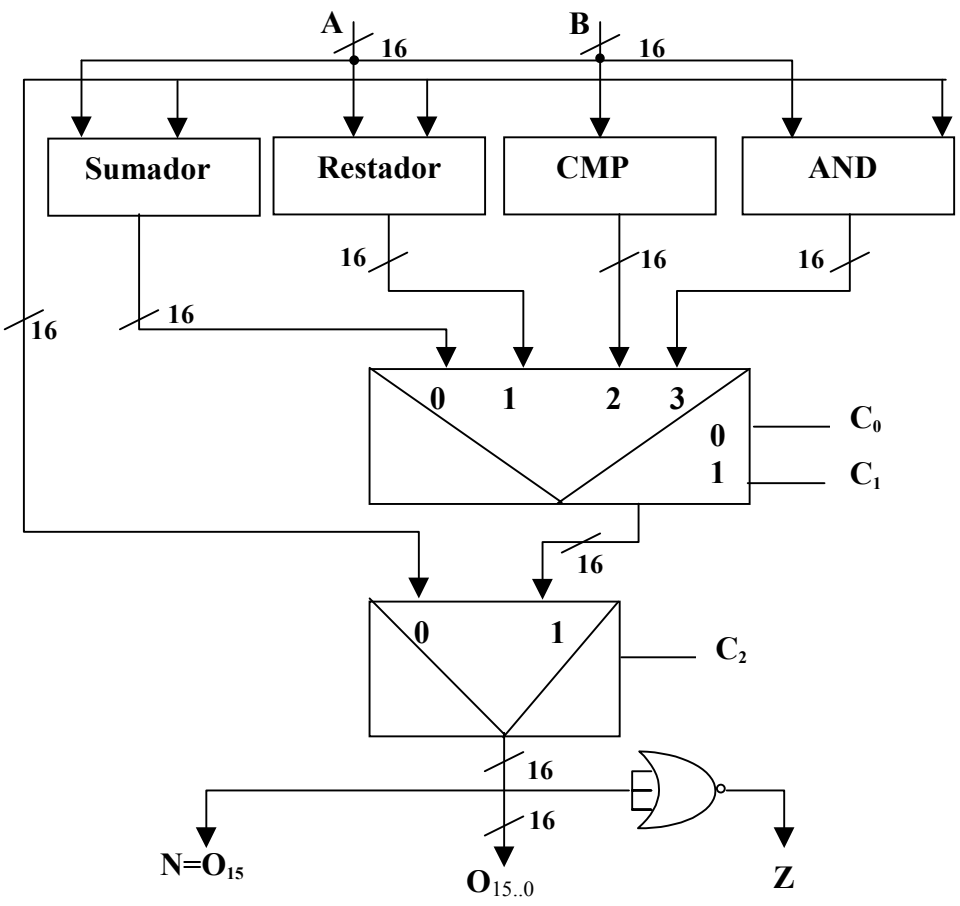

**21.**

a) La nova operació de comparar es pot realitzar mitjançant el circuit següent:




**BLOC CMP**

Usant aquest bloc, la nova UAL quedaria dissenyada de la forma següent:




b)

Solució 1

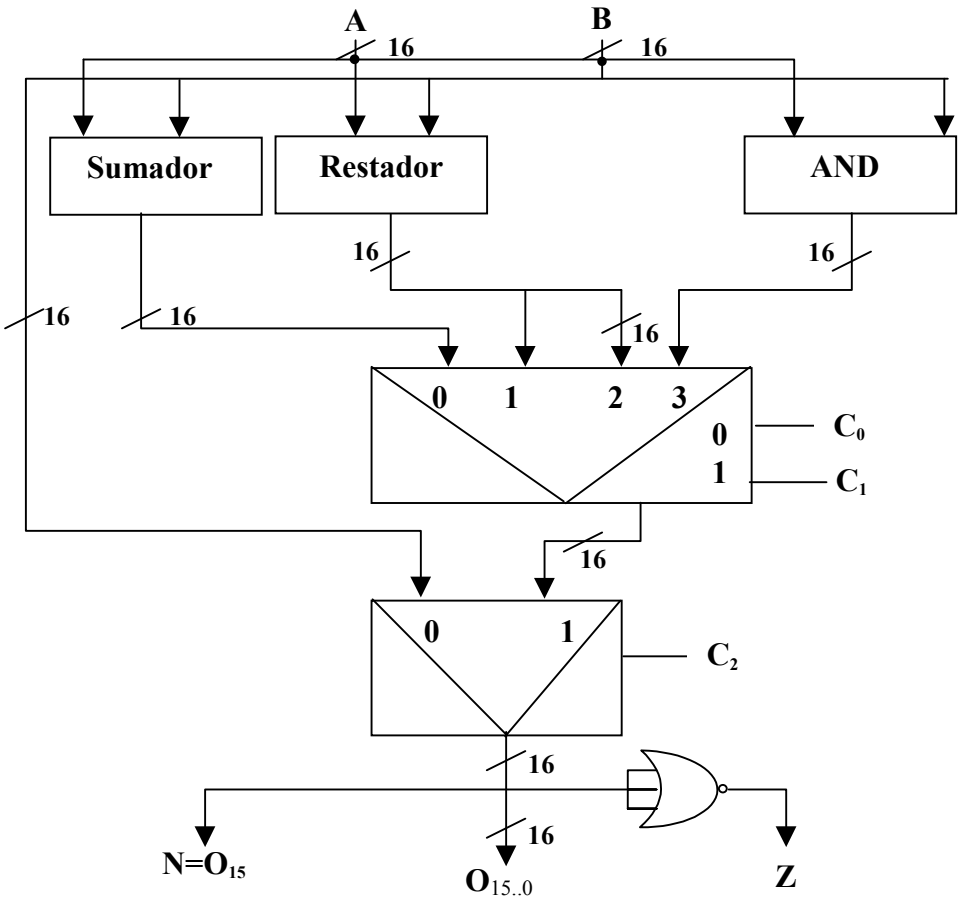
Si la circuiteria del bloc CMP substitueix el que realitza l'ASR, el codi de la instrucció CMP pot ésser el que tenia la instrucció ASR.
Per a l'execució de la instrucció CMP no ens interessa guardar el resultat de realitzar la XOR dels dos operands font ($O_{15\text{-}0}$). Si en el format d'aquesta instrucció indiquem que el registre destinació és R0, no es guardarà el resultat. Aquesta solució permet de tractar la nova instrucció igual que les altres instruccions aritmètiques i lògiques.

| | | | | | |
|---|---|---|---|---|---|
| **11** | **000** | **Rf1** | **Rf2** | **00** | **110** |

**CMP Rf1, Rf2**

Solució 2

Una altra possible solució és usar el restador de la UAL per a implementar l'operació CMP. D'aquesta manera no seria necessari l'ús d'un bloc específic com el dissenyat en l'apartat a). Mantenint la codificació de la solució 1, la nova UAL seria:




La instrucció de comparar CMP Rf1, Rf2 es traduiria igual que a la solució 1:

| 11 | 000 | Rf1 | Rf2 | 00 | 110 |
|---|---|---|---|---|---|

**CMP Rf1, Rf2**

Solució 3

Una altra possible solució seria afegir la nova instrucció CMP sense eliminar-ne cap de les anteriors. No caldria realitzar cap modificació a la UAL, i n'hi hauria prou de codificar-la com una instrucció de restar que deixa el resultat al registre R0. La nova codificació seria, per tant:

| 11 | 000 | Rf1 | Rf2 | 00 | 101 |
|---|---|---|---|---|---|

**CMP Rf1, Rf2 = SUB Rf1, Rf2, R0**

**22.**

a) El temps mínim necessari per a completar un estat del graf depèn del retard provocat pels diferents circuits combinacionals que intervenen en l'execució de les operacions realitzades a l'estat.

Estat FETCH

L'adreça emmagatzemada al PC travessa el multiplexor SELDIR (10 ns.) i arriba al bus d'adreces de la memòria. La memòria subministra la dada (100 ns.) al registre IR. En total, 110 ns.
Juntament amb l'acció anterior, l'adreça present a la sortida del multiplexor SELDIR (10 ns.) s'incrementa en 1 (50 ns.) per tal d'ésser present a l'entrada del PC. En total, 60 ns.
Com que ambdues operacions es realitzen en paral·lel, el temps mínim necessari per a executar aquest estat és de 110 ns.

**Estat DECO**

En aquest estat es realitza, a més a més de la descodificació de la instrucció (el seu temps no el tenim en compte perquè es realitza en un temps despreciable a la Unitat de Control), el càlcul de la Condició de Salt, la càrrega del primer operand de les instruccions aritmètiques i lògiques al registre RA i la càrrega de l'adreça d'accés a memòria de les instruccions Load, Store i de Salt al registre R@.

- El càlcul de la Condició de Salt requereix 40 ns.
- Per a emmagatzemar el primer operand de les instruccions aritmètiques i lògiques a RA cal enviar els 3 bits que codifiquen el registre font travessant el multiplexor SELDIR (10 ns.) i llegir l'operand del Banc de Registres (20 ns.). En total, 30 ns.
- Per a emmagatzemar l'adreça d'accés a memòria al registre R@ s'ha de sumar el contingut del registre índex (30 ns. per a accedir al Banc de Registres, 10 ns. per a seleccionar els bits a SELDIR i 20 ns. per a llegir el registre) amb l'adreça base que hi ha a la instrucció (50 ns. per a realitzar la suma). En total, 80 ns.

Per tant, el temps necessari per a aquesta fase és de 80 ns.

**Estat LOAD**

L'adreça que hi ha a R@ s'envia a la memòria a través del multiplexor SELDIR (10 ns.). La memòria serveix la dada (100 ns.), que passa a través del multiplexor SELDAT (10 ns.) i de la UAL (100 ns.) per a ésser present a l'entrada del Banc de Registres. Total, 220 ns.

**Estat STORE**

L'adreça que hi ha a R@ i s'envia a la memòria travessant el multiplexor SELDIR (10 ns.). En paral·lel, el Banc de Registres envia a la memòria en 30 ns. la dada que ha d'ésser escrita (10 ns. de SELDIR i 20 ns. de lectura del registre). La memòria tarda 100 ns. a escriure la dada, i per tant el temps total és de 130 ns.

**Estat ARIT**

En aquest estat es realitza la fase d'execució de les instruccions aritmètiques i lògiques i, en paral·lel, el fetch de la instrucció següent. Tal com s'ha vist anteriorment, el fetch requereix 110 ns.

- Per a executar una instrucció aritmètica o lògica registre-registre, el segon operand s'envia des del Banc de Registres (30 ns., 10 ns. de SELDIR i 20 ns. de lectura del registre) a través del multiplexor SELDAT (10 ns.) i la UAL realitza l'operació corresponent (100 ns.). En total, 140 ns.
  El temps necessari per a escriure el resultat al Banc de Registres no es comptabilitza, ja que aquesta escriptura es realitza durant la descodificació de la instrucció següent (i tarda menys temps que la descodificació).

- Per a executar una instrucció aritmètica o lògica registre-immediat, el segon operand s'envia des del registre IR (prèvia extensió de signe) a través del multiplexor SELDAT (10 ns.) i la UAL realitza l'operació corresponent (100 ns.). En total, 110 ns.
  Igual com en el cas anterior, el temps necessari per a escriure el resultat al Banc de Registres no es comptabilitza.

Per tant, el temps necessari per a executar aquesta fase és de 140 ns.

**Estat BRANCH**

En aquest estat es realitza un fetch des del registre R@, i tarda el mateix temps que un fetch des del PC. Per tant, aquest estat requereix 110 ns.

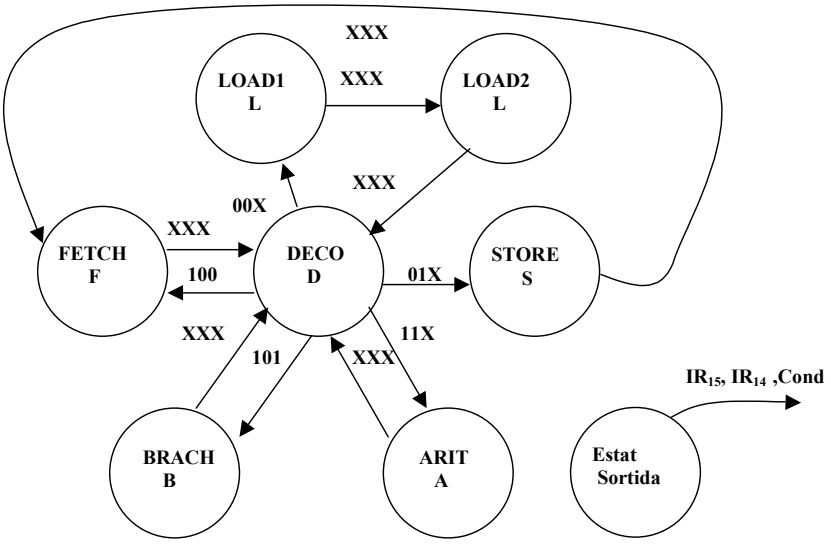
b) La freqüència màxima depèn del temps que tardi en executar-se l'estat que necessita més temps. Tal com hem vist en l'apartat a), l'estat LOAD és el que tarda més temps: 220 ns. Això implica que el període del rellotge ha d'ésser superior o igual a 220 ns., i com que el període (T) és l'invers de la freqüència (F), aleshores tenim que:

$T >= 220 .10^{-9}$ s; $F <= 1/(220 .10^{-9})$ ; $F <= 4{,}5454545 .10^{6}$ = = 4545454,5 Hz. = 4,54 MHz.

Per tant, la freqüència ha d'ésser inferior o igual a 4,54 MHz.

**23.**

a) La nova Unitat de Procés i el nou graf d'estats de la Unitat de Control són els següents:




TAULA DE SORTIDES

| Sortides UC | Load1 | Load2 |
|---|---|---|
| Ld_RA | 0 | 0 |
| Ld_IR | 0 | 1 |
| Ld_PC | 0 | 1 |
| Ld_R@ | 0 | 0 |
| Ld_RMEM | 1 | 0 |
| Ld_RZ | 0 | 1 |
| Ld_RN | 0 | 1 |
| ERd | 0 | 1 |
| $\overline{L/E}$ | 0 | 0 |
| $\overline{PC/@}$ | 1 | 0 |
| CRf | X | X |
| OPERAR | X | 0 |

A l'estat LOAD1 es carrega l'operand font de la instrucció LOAD, un cop ha estat llegit de la memòria, al registre RMEM.
A l'estat LOAD2 s'emmagatzema aquest operand al registre destinació, i en paral·lel es realitza el fetch de la instrucció següent. El nombre de cicles que tarda a executar-se una instrucció LOAD no augmenta malgrat que s'incrementa el nombre d'estats del graf, ja que després de l'estat LOAD2 es passa a la fase de Descodificació, en comptes de passar a la fase de Fetch com es feia al graf original. Les taules de sortida d'ambdós estats es mostren a la Taula.

b) El temps mínim necessari per a cada fase depèn del retard provocat pels diferents circuits combinacionals que intervenen en l'execució de la fase. Com que al graf

només varien els estats LOAD 1 i LOAD2, i la inserció del registre RMEM només afecta l'execució de les instruccions LOAD, la resta dels estats necessiten el temps estudiat en el problema 22.

Estat FETCH: 110 ns.
Estat DECO: 80 ns.
Estat STORE: 130 ns.
Estat ARIT: 140 ns.
Estat BRANCH:110 ns.

**Estat LOAD1**

L'adreça que hi ha a R@ s'envia a la memòria a través del multiplexor SELDIR (10 ns.). La memòria serveix la dada (100 ns.) i el resultat s'escriu, al final del cicle, al registre RMEM. Per tant, el temps necessari per a aquest estat és de 110 ns.

**Estat LOAD2**

La dada que hi ha al registre RMEM passa a través del multiplexor SELDAT (10 ns.) i la UAL (100 ns.) per ésser present a l'entrada del Banc de Registres. En total, 110 ns. En paral·lel es realitza el fetch de la instrucció següent, que tarda 110 ns. Per tant, el temps necessari per a aquest estat és de 110 ns.

La freqüència mínima depèn del temps que tardi la fase més llarga, que en aquest cas és la fase ARIT amb 140 ns. Això implica que el període del rellotge ha d'ésser superior o igual a 140 ns., i com que el període (T) és l'invers de la freqüència (F), tenim que:

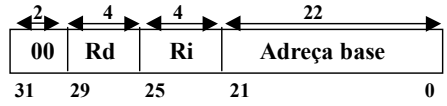
$T >= 140.10^{-9}$ s; $F <= 1/(140.10^{-9})$ ; $F <= 7{,}1428571 \cdot 10^6 = 7142857{,}1$ Hz = 7,14 MHz.

Per tant, F <= 7,14 MHz.

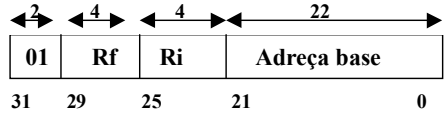
c) Dels apartats anteriors deduïm que, encara que augmenti el nombre d'estats del graf, el nombre d'estats (cicles) necessari per a executar cada instrucció és el mateix. Però la freqüència del rellotge és més gran, i els programes s'executaran més ràpidament a la MR modificada. En concret, la MR anirà 7,14 / 4,54 = 1,57 cops més ràpida en executar cada instrucció. Això comporta un increment de més del 50 % en la seva velocitat.

**24.**
Com que l'amplària de la memòria és de 32 bits, les instruccions poden tenir també una amplària de 32 bits. El camp de la instrucció que codifica un registre del Banc de Registres necessita 4 bits, ja que el Banc de Registres té 16 registres.

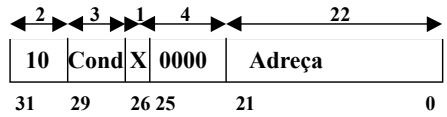
a) Una possible codificació, que manté els mateixos codis d'operació (CO), els mateixos codis de condició (COND) i la mateixa codificació per a cada instrucció aritmètica o lògica, és la que presentem en la figura següent. Com podeu observar, la grandària



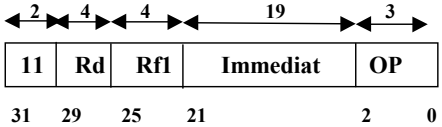

**Format de la instrucció Load**



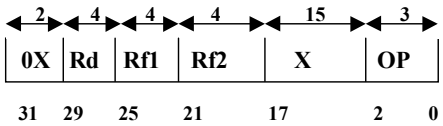

**Format de la instrucció Store**




**Format de les instruccions de Salt**




**Format de les instruccions de suma i resta amb immediat**




**Format de les instruccions de suma, resta, desplaçament a la dreta i AND lògica**

màxima per a l'operand immediat és de 19 bits.

b) Pel que fa a la resta de la Unitat de Procés, caldria modificar els elements següents:
- El Banc de Registres hauria de tenir 16 registres de 32 bits.
- Els busos que interconnecten el Banc de Registres i la memòria amb altres elements de la Unitat de Procés haurien d'ésser de 32 bits.
- El multiplexor SELREG hauria de tenir 4 entrades de dades de 4 bits, una sense usar.
- Els registres IR i RA haurien d'ésser de 32 bits.
- Els registres PC i R@ haurien d'ésser de 22 bits.
- El sumador d'adreces i l'incrementador haurien d'ésser de 22 bits.
- El multiplexor SELDIR hauria d'ésser de 2 entrades de dades de 22 bits.
- El multiplexor SELDAT hauria d'ésser de 4 entrades de dades de 32 bits.
- La UAL hauria d'operar amb 32 bits.

c) Per a augmentar el nombre d'instruccions aritmètiques i lògiques fins a 64 n'hi ha prou de dotar el camp OPS amb 6 bits. Això implica que la grandària de l'operand immediat passa a ésser, com a màxim, de 16 bits. La resta d'instruccions no es veuen alterades.

Com que el bit 26 del format de les instruccions de salt no es fa servir, podria afegir-se als bits 29:27 que codifiquen el camp COND. La MR tindria aleshores 4 bits per a codificar instruccions de salt; això fa un total de 16. Com que 7 ja estan definides, podrien se'n definir fins a 9 més. Per exemple, es podrien definir instruccions que saltessin segons si s'ha produït sobreeiximent, tenint en compte que els operands siguin naturals o enters, o bé, per a cada instrucció de salt ja existent, podria definir-se'n una altra que considerés els operands com a nombres naturals.

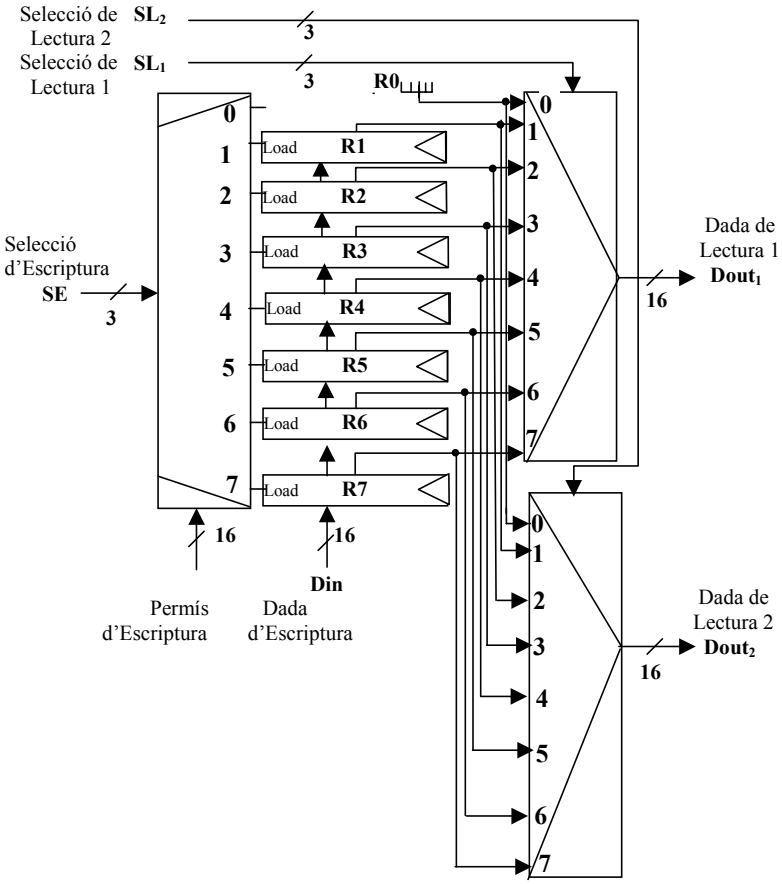
Una possible llista d'instruccions és la següent:

- Saltar si es produeix sobreeiximent amb operands enters (V=1).
- Saltar si es produeix sobreeiximent amb operands naturals (C=1).
- Saltar si més gran o igual, considerant els nombres com a operands naturals (C=0).
- Saltar si menor o igual, considerant els nombres com a operands naturals (C=1 o Z=1).
- Saltar si més gran, considerant els nombres com a operands naturals (C=0 i Z=0).
- Saltar si menor, considerant els nombres com a operands naturals (C=1).
- Saltar si igual, considerant els nombres com a operands naturals (Z=1).
- Saltar si no igual, considerant els nombres com a operands naturals (Z=0).

**25.**

a) Per a permetre la lectura simultània de dos registres del Banc de Registres, aquest ha de tenir dos multiplexors de sortida, cadascun d'ells controlat per un bus de tres bits que codifica el registre a llegir.

b) En la figura es mostra un Banc de Registres amb dos ports de lectura.

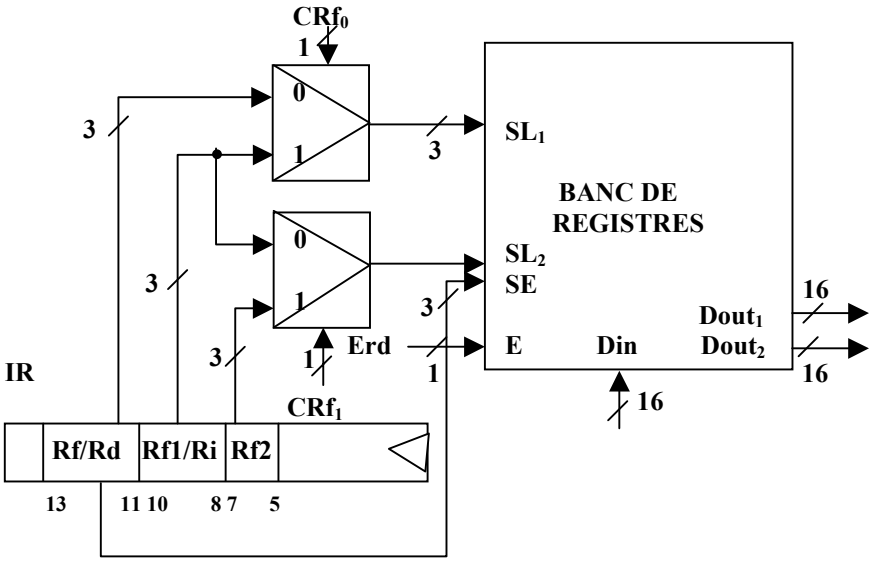



En el context de la MR, un Banc de Registres amb dos ports de lectura pot ser molt interessant en l'execució de:

- Les instruccions aritmètiques i lògiques, ja que es poden llegir simultàniament els dos operands font (en cas que tots dos estiguin emmagatzemats al Banc de Registres). Si aquesta lectura simultània es realitza en la fase d'execució de la instrucció (estat ARIT), no és necessari el registre RA a l'entrada de la UAL. Si la lectura simultània es realitza en la fase de descodificació de la instrucció (estat DECO), es requereix un altre registre a l'entrada B de la UAL, però aconseguim que l'estat ARIT requereixi menys temps (vegeu el problema 22), el de llegir del Banc de Registres el segon operand.

- La instrucció STORE, ja que es poden llegir simultàniament el registre font i el registre índex. Si aquesta lectura simultània es realitza en la fase d'execució de la instrucció (estat STORE), no cal el registre R@. Si la lectura simultània es realitza en la fase de descodificació de la instrucció (estat DECO) es requereix un registre a l'entrada de dades de la memòria (Min), però aconseguim que l'estat STORE necessiti menys temps (vegeu l'activitat 22), el de llegir del Banc de Registres el registre font.

La circuiteria que controla el Banc de Registres s'ha de modificar, i queda tal com es mostra en la figura següent. El control ha de generar els nous senyals $CRf_0$ i $CRf_1$ en l'estat adequat. El format de les instruccions no s'ha de modificar.

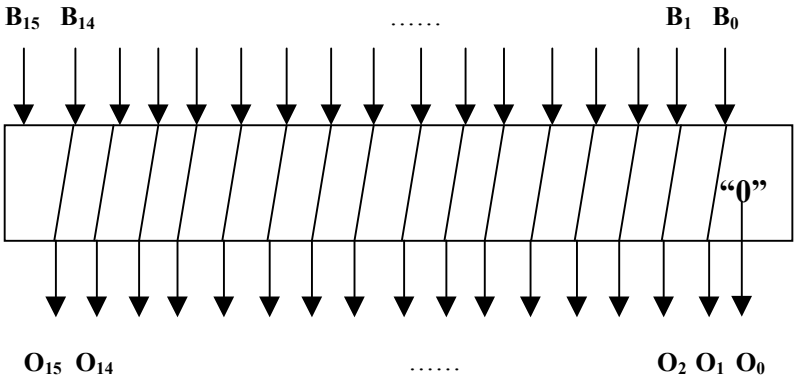



26.

a) El format de les instruccions aritmètiques i lògiques permet de codificar fins a quatre instruccions que treballen amb immediat (codificades als bits IR1 i IR0) i fins a 16 instruccions que treballen amb registres del processador (usant els bits IR1 i IR0 i els bits IR4 i IR3, no usats actualment). Per tant, podem agafar la codificació següent per a les instruccions aritmètiques i lògiques que treballen amb registres:

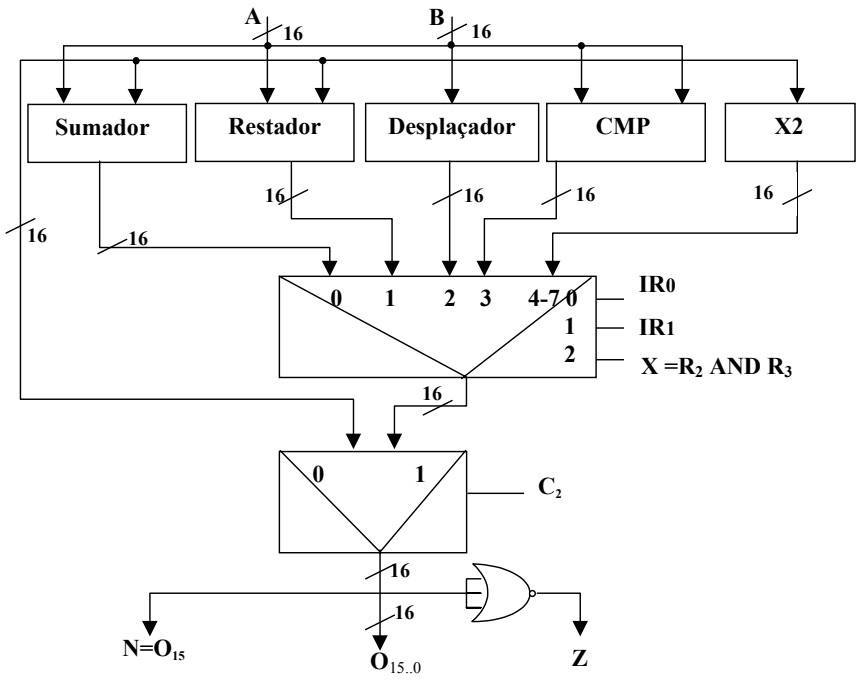
| | | | | | | |
|---|---|---|---|---|---|---|
| **11** | **Rd** | **Rf1** | **Rf2** | **00** | **100** | **Suma** |
| **11** | **Rd** | **Rf1** | **Rf2** | **00** | **101** | **Resta** |
| **11** | **Rd** | | **Rf2** | **00** | **110** | **Desplaçament a la dreta** |
| **11** | **Rd** | **Rf1** | **Rf2** | **00** | **111** | **And-lògica** |
| **11** | **Rd** | | **Rf2** | **01** | **100** | **Multiplicar per dos** |

b) La multiplicació per dos consisteix bàsicament a desplaçar els bits de l'operand font cap a l'esquerra, afegint un zero al bit de menys pes de la sortida. Així el bloc que realitza aquesta operació és:




Si coneixem la codificació de les instruccions [apartat a)] podem veure com s'ha de modificar la UAL. En afegir l'operació de multiplicar per dos, la sortida de la UAL s'ha de seleccionar d'entre les sortides dels cinc blocs que té (sumador, restador, desplaçador, and-lògica i multiplicar per dos). D'aquesta manera són necessaris tres

bits de selecció, dos són els bits $IR_0$ i $IR_1$, i el tercer bit, X, és una funció que identifica la instrucció MULDOS de forma única, $X = IR_2$ *AND* $IR_3$.




**27.**

a) Les instruccions s'executen en l'ordre següent:
1) 00h BR 3
2) 03h ADD R0, R0, R0
3) 04h BGE 1
4) 01h SUB R0, R0, R0
5) 02h BR 5
A partir d'aquí, es continua executant la instrucció que hi ha a la posició 5 de memòria.

b) Calcularem el nombre de cicles de cada instrucció. Com que cada estat dura exactament un cicle, el nombre d'estats que recorre cada instrucció és el nombre de cicles que tarda a executar-se.

| | |
|---|---|
| **BR 3** | 1 cicle FETCH, 1 cicle DECO, 1 cicle BRANCH. Se salta a la fase DECO de la instrucció següent. Total: 3 cicles |
| **ADD R0, R0, R0** | 1 cicle DECO, 1 cicle ARIT. Se salta a la fase DECO de la instrucció següent. Total: 2 cicles |
| **BGE 1** | 1 cicle DECO. Es produeix el salt: 1 cicle BRANCH. Se salta a la fase DECO de la instrucció següent. Total: 2 cicles |
| **SUB R0, R0, R0** | 1 cicle DECO, 1 cicle ARIT. Se salta a la fase DECO de la instrucció següent. Total: 2 cicles |
| **BR 5** | 1 cicle DECO, 1 cicle BRANCH. Total: 2 cicles |

En total, el programa tarda 11 cicles a executar-se.

c) Els valors del PC per a cada fase són els següents (en la taula s'indiquen els valors del PC durant l'execució de la fase indicada):

| | FETCH | DECO | ARIT | BRANCH |
|---|---|---|---|---|
| BR 3 | 00H | 01H | | 01H |

| | | | | |
|---|---|---|---|---|
| ADD | - | 04H | 04H | |
| BGE 1 | - | 05H | | 05H |
| SUB | - | 02H | 02H | |
| BR5 | - | 03H | | |

d) Seqüència d'adreces corresponent a l'execució d'instruccions:
**00h, 03h, 04h, 01h, 02h, 05h**

Seqüència d'adreces emmagatzemada al PC:
**00h, 01h, 04h, 05h, 02h, 03h**

No són iguals perquè, en les instruccions de salt, en el moment en què es produeix el salt l'adreça de la següent instrucció a executar no és al registre PC, sinó al registre R@.

**28.**

a) El bucle comença a l'adreça 00h (comparació d'entrada) i acaba a l'adreça 07h (salt incondicional a la comparació d'entrada al bucle). El bucle acaba quan R3 és més petit que R2, per la qual cosa s'entra al bucle mentre R3 és més gran o igual que R2 (comparació efectuada a l'adreça 00h).
Dins del bucle, si R4 és igual que R2 s'incrementa R2 en una unitat. En cas contrari, es resten dues unitats a R4 i s'incrementa R2 en una unitat.

b) Com que inicialment R3=$100_{10}$ i R2=$49_{10}$, i R2 s'incrementa en una unitat en cada iteració, el bucle s'executa 52 cops (100 – 49 +1).

**29.**

a) Vegem l'efecte de cadascun dels senyals:

- Amb Ld_PC = 1 es permet que el registre PC es carregui amb l'adreça que ve de la memòria a través del bus M@, incrementada en u.
- Amb Ld_IR = 1 es permet que el registre IR es carregui amb la instrucció que s'envia de la memòria a través del bus Mout.
- Amb PC/@ = 0 es permet que l'adreça que rep la memòria pel bus M@ sigui la que s'emmagatzema al registre PC.

Per tant, l'efecte d'activar els tres senyals simultàniament és realitzar el fetch d'una instrucció usant el contingut del registre PC. Això es fa en l'estat de FETCH o durant l'estat ARIT del graf simplificat.

b) Vegem l'efecte de cadascun dels senyals:
- Amb Ld_PC = 1 es permet que el registre PC es carregui amb l'adreça que ve de la memòria a través del bus M@, incrementada a u.
- Amb Ld_IR = 1 es permet que el registre IR es carregui amb la instrucció que s'envia de la memòria a través del bus Mout.
- Amb PC/@ = 1 es permet que l'adreça que rep la memòria pel bus M@ sigui la que s'emmagatzema al registre R@.

L'efecte d'activar els tres senyals simultàniament és realitzar el fetch d'una instrucció usant l'adreça destinació d'un salt. Això es fa en l'estat de BRANCH.

c) Vegem l'efecte de cadascun dels senyals:

- Amb Ld_RA = 1 es permet que el registre RA es carregui amb la dada que ve del Banc de Registres.
- Amb CRf = 1 es permet seleccionar del Banc de Registres un registre que fa la funció de Rf1 (registre font 1) o de Ri (registre índex).

Tenint en compte el valor dels dos senyals, l'objectiu és emmagatzemar a RA un primer operand font durant la realització de l'estat DECO.

d) L'efecte del senyal ERd = 1 és permetre que un registre del Banc de Registres s'escrigui amb la dada present al bus ALU (sortida de la UAL). Aquesta acció es realitza en dos estats: en ARIT per a emmagatzemar el resultat d'una operació aritmètica o lògica i en LOAD per a emmagatzemar al Banc de Registres la dada que s'ha llegit de la memòria.

e) Vegem l'efecte de cadascun dels senyals:

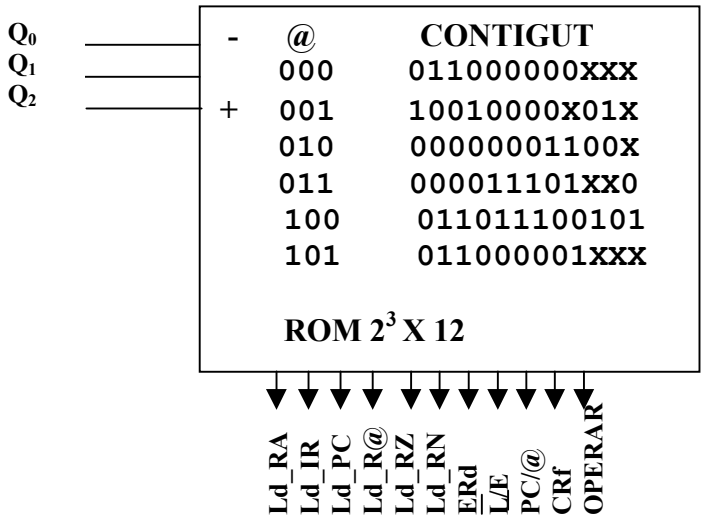
- Amb Ld_RZ = 1 i Ld_RN = 1 es permet que els indicadors de condició s'actualitzin amb els valors calculats per la UAL.
- Amb ERd = 1 es permet que un registre del Banc de Registres s'escrigui amb la dada present al bus ALU (sortida de la UAL).
- Amb OPERAR = 1 s'indica que a la sortida de la UAL (bus ALU) hi ha d'haver el resultat de realitzar una operació aritmètica o lògica. Per a això, el multiplexor *SELDAT* permet que a l'entrada B de la UAL hi hagi una dada procedent del Banc de Registres, o bé l'operand immediat que hi ha emmagatzemat a la instrucció. A més a més, s'indica a la UAL que la seva sortida ha d'ésser el resultat d'una operació aritmètica o lògica.
- Amb CRf = 2 es permet seleccionar del Banc de Registres un registre que fa la funció de Rf2 (registre font 2).

A partir del valor dels senyals es dedueix que s'està realitzant l'estat ARIT, però no es pot concretar si el segon operand font és el que es llegeix del Banc de Registres o és un operand immediat.

**30.**
A continuació es mostra la taula de transicions.

| $Q_2$ | $Q_1$ | $Q_0$ | $X_2$ | $X_1$ | $X_0$ | $Q_2^+$ | $Q_1^+$ | $Q_0^+$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | X | X | X | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | X | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | X | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | X | 1 | 0 | 0 |
| 0 | 1 | 0 | X | X | X | 0 | 0 | 0 |
| 0 | 1 | 1 | X | X | X | 0 | 0 | 0 |
| 1 | 0 | 0 | X | X | X | 0 | 0 | 1 |
| 1 | 0 | 1 | X | X | X | 0 | 0 | 1 |

Funció de sortida amb una ROM.




**31.**

a) Per tal que la Unitat de Control sàpiga quina instrucció s'està executant, i per tant quina seqüència de senyals de control ha de generar.

b) Les transicions entre estats es produeixen a cada flanc ascendent de rellotge. Per tant, l'execució d'una instrucció tardarà tants cicles com estats hagi de recórrer.
La durada del cicle de rellotge del processador és la de l'estat que tarda més temps a realitzar-se. Per tant, en dissenyar el graf d'estats, no s'ha de pensar tant a reduir el nombre d'estats augmentant el contingut semàntic d'algun estat (cosa que podria incrementar el temps de cicle), com a trobar un equilibri entre el nombre d'estats i la durada del cicle del processador.
D'altra banda, com menys estats tingui el graf de la Unitat de Control més senzilla i barata serà la seva implementació.

**32.**

Vegeu la solució del problema 25.

**33.**

a) Si la UAL de la MR, a més a més dels indicadors de condició Z i N, també calculés els bits

- V: sobreeiximent per a enters
- C: sobreeiximent per a naturals

es podria considerar la inclusió de, almenys, quatre instruccions de salt més:

- BV Saltar si hi ha sobreeiximent per a enters: es produeix el salt si V = 1.
- BNV Saltar si no hi ha sobreeiximent per a enters: es produeix el salt si V = 0.
- BC Saltar si hi ha sobreeiximent per a naturals: es produeix el salt si C = 1.
- BNC Saltar si no hi ha sobreeiximent per a naturals: es produeix el salt si C = 0.

b) Conservant el format actual de les instruccions de salt solament es pot incloure una instrucció més, i el seu codi seria COND=100.

Per a incloure les quatre instruccions noves caldria canviar el format de les instruccions de salt. Per tal d'evitar-ho, proposarem un nou conjunt complet de 8 instruccions de salt que permeten de manejar convenientment els quatre indicadors de condició.
Les instruccions BZ, BNZ, BL i BG permeten de realitzar totes les operacions que realitzaven les 7 instruccions de salt que abans tenia la MR. Per exemple, per a saltar si més petit o igual es pot usar la instrucció BG intercanviant els operands de la comparació: la seqüència d'instruccions

- SUB R2, R1, R0
- BLE 64h

és equivalent a la seqüència

- SUB R1, R2, R0
- BG 64h

El conjunt complet d'instruccions de salt serà:

- BZ o BE Saltar si igual: es produeix el salt si Z=1.

- BNZ o BNE Saltar si diferent: es produeix el salt si Z=0.
- BL Saltar si més petit: es produeix el salt si N=1.
- BG Saltar si més gran: es produeix el salt si N=0 i Z=0.
- BV Saltar si hi ha sobreeiximent per a enters: es produeix el salt si V=1.
- BNV Saltar si no hi ha sobreeiximent per a enters: es produeix el salt si V=0.
- BC Saltar si hi ha sobreeiximent per a naturals: es produeix el salt si C=1.
- BNC Saltar si no hi ha sobreeiximent per a naturals: es produeix el salt si C=0.

**34.** Iniciarem el programa amb la declaració de les variables "a" i "b" mitjançant la utilització de dues directrius de reserva de memòria:

```
a:      .RW 1
b:      .RW 1
```

A continuació s'iniciarà el codi en llenguatge assemblador (utilitzant la directiva d'inici de programa) i es realitzarà la inicialització de les variables. Per a això, suposarem que les variables "a" i "b" s'emmagatzemen temporalment als registres R1 i R2 respectivament.

La constant +16 supera el rang dels valors immediats que es poden representar (–16,...,0,...15), per la qual cosa s'haurà d'inicialitzar amb dues instruccions de suma o bé mitjançant una de resta (pel fet que el rang dels nombres negatius és més gran que el rang dels positius en la representació en complement a dos).

```
        .BEGIN inici
inici:  ADDI R0, #13, R1
        SUBI R0, #-16, R2
```

El bucle *mentre* es construirà amb dues instruccions. La primera calcularà la condició del salt i la segona decidirà si s'ha de saltar o no:

```
mentre:     SUBI R1, #10, R0
        BLE fmentre
```

Les instruccions següents implementaran el cos del bucle *mentre*. S'utilitzarà una instrucció de salt incondicional per a retornar a la instrucció del càlcul de la condició d'entrada del bucle:

```
        SUBI R1, #1, R1
        ADDI R2, #2, R2
        BR mentre
```

A continuació indicarem que s'ha finalitzat l'execució del bucle *mentre* i que s'inicia l'execució de la sentència condicional *si*:

```
fmentre:    SUB R1, R2, R0
        BGE si_no
```

Si la condició de la instrucció es compleix, s'ha d'executar la instrucció SWAP (implementada amb tres instruccions aritmètiques), mentre que en cas contrari s'executarà la instrucció del *si_no*. En ambdós casos, després d'executar les instruccions corresponents s'ha d'arribar al final del programa:

```
si:     SUB R1, R2, R2
        SUB R1, R2, R1
        ADD R1, R2, R2
        BR fsi
si_no:  SUBI R1, #1, R2
fsi:
```

Finalment, el resultat emmagatzemat als registres s'ha de guardar a la memòria, que és on resideixen les variables:

```
STORE R1, a(R0)
STORE R2, b(R0)
.END
```

A continuació es presenta el programa descrit:

```
a:        .RW 1
b:        .RW 1
          .BEGIN inici
inici:    ADDI R0, #13, R1
          SUBI R0, #-16, R2
mentre:   SUBI R1, #10, R0
          BLE fmentre
          SUBI R1, #1, R1
          ADDI R2, #2, R2
          BR mentre
fmentre:  SUB R1, R2, R0
          BGE si_no
si:       SUB R1, R2, R2
          SUB R1, R2, R1
          ADD R1, R2, R2
          BR fsi
si_no:    SUBI R1, #1, R2
fsi:      STORE R1, a(R0)
          STORE R2, b(R0)
          .END
```

Un cop completat el programa en llenguatge assemblador, a continuació presentem la seva traducció a llenguatge màquina i la taula de símbols generada pel programa assemblador:

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| ADDI R0, #13, R1 | 02h: | 11 001 000 01101 000 | C868h |
| SUBI R0, #-16, R2 | 03h: | 11 010 000 10000 001 | D081h |
| SUBI R1, #10, R0 | 04h: | 11 000 001 01010 001 | C151h |
| BLE fmentre | 05h: | 10 011 000 00001001 | 9809h |
| SUBI R1, #1, R1 | 06h: | 11 001 001 00001 001 | C909h |
| ADDI R2, #2, R2 | 07h: | 11 010 010 00010 000 | D210h |
| BR mentre | 08h: | 10 000 000 00000100 | 8004h |
| SUB R1, R2, R0 | 09h: | 11 000 001 010 00 101 | C145h |
| BGE sinó | 0Ah: | 10 110 000 00001111 | B00Fh |
| SUB R1, R2, R2 | 0Dh: | 11 010 001 010 00 101 | D145h |
| SUB R1, R2, R1 | 0Bh: | 11 001 001 010 00 101 | C945h |
| ADD R1, R2, R2 | 0Ch: | 11 010 001 010 00 100 | D144h |
| BR fsi | 0Eh: | 10 000 000 00010000 | 8010h |
| SUBI R1, #1, R2 | 0Fh: | 11 010 001 00001 001 | D109h |
| STORE R1, A(R0) | 10h: | 01 001 000 00000000 | 6800h |
| STORE R2, B(R0) | 11h: | 01 010 000 00000001 | 6001h |

Taula de símbols:

| Símbol | Valor |
|---|---|
| a: | 00h |
| b: | 01h |
| inici: | 02h |
| mentre: | 04h |
| fmentre: | 09h |
| si: | 0Bh |
| sinó: | 0Fh |
| fsi: | 10h |

**35.**

```
a:        .RW 1;                 a -> R1
b:        .RW 1;                 b -> R2
c:        .RW 1;                 c -> R3
          .BEGIN inici;          R4 serà una variable auxiliar
inici:    ADDI R0, #13, R1; a
          ADDI R0, #1, R2;       b
          ADD R0, R0, R3;        c
mentre:   SUBI R1, #10, R0
          BG cos
```

```
            SUBI R2, #10, R4
            SUBI R4, #10, R0
            BGE fmentre
cos:        SUBI R1, #11, R0
            BGE següent
            SUBI R2, #10, R4
            SUBI R4, #8, R0
            BLE següent
            ADDI R3, #3, R3
següent:    SUB R1, R2, R1
            BR mentre
fmentre:    STORE R1, a(R0)
            STORE R2, b(R0)
            STORE R3, c(R0)
            .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| ADDI R0, #13, R1 | 03h: | 11 001 000 01101 000 | C868h |
| ADDI R0, #1, R2 | 04h: | 11 010 000 00001 000 | D008h |
| ADD R0, R0, R3 | 05h: | 11 011 000 000 00 100 | D804h |
| SUBI R1, #10, R0 | 06h: | 11 000 001 01010 001 | C151h |
| BG cos | 07h: | 10 111 000 0000 1011 | B80Bh |
| SUBI R2, #10, R4 | 08h: | 11 100 010 01010 001 | E251h |
| SUBI R4, #10, R0 | 09h: | 11 000 100 01010 001 | C451h |
| BGE fmentre | 0Ah: | 10 110 000 0001 0011 | B813h |
| SUBI R1, #11, R0 | 0Bh: | 11 000 001 01011 001 | C159h |
| BGE següent | 0Ch: | 10 110 000 0001 0001 | B011h |
| SUBI R2, #10, R4 | 0Dh: | 11 100 010 01010 001 | E251h |
| SUBI R4, #8, R0 | 0Eh: | 11 000 100 01000 001 | C441h |
| BLE següent | 0Fh: | 10 011 000 0001 0001 | 9811h |
| ADDI R3, #3, R3 | 10h: | 11 011 011 00011 000 | DB18h |
| SUB R1, R2, R1 | 11h: | 11 001 001 01000 101 | C945h |
| BR mentre | 12h: | 10 000 000 0000 0110 | 8006h |
| STORE R1, A(R0) | 13h: | 01 001 000 0000 0000 | 4800h |
| STORE R2, B(R0) | 14h: | 01 010 000 0000 0001 | 5001h |
| STORE R3, C(R0) | 15h: | 01 011 000 0000 0010 | 5802h |

Taula de símbols

| Símbol | Valor |
|---|---|
| a: | 00h |
| b: | 01h |
| c: | 02h |
| inici: | 03h |
| mentre: | 06h |
| cos: | 0Bh |
| següent: | 11h |
| fmentre: | 13h |

**36.**

a)

```
a:          .DW 10
b:          .DW 15
mcd:        .RW 1
            .BEGIN inici
inici:      LOAD a(R0), R1;         variable a -> R1
            LOAD b(R0), R2;         variable b -> R2
mentre:     SUB R1, R2, R0
            BEQ fmentre
            BLE sinó
            SUB R1, R2, R1
            BR mentre
sinó:       SUB R2, R1, R2
            BR mentre
fmentre:    STORE R1, mcd(R0);      "a" i "b" no s'escriuen en memòria
            .END
```

b)

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| LOAD a(R0), R1 | 12h: | 00 001 000 0000 1111 | 080Fh |
| LOAD b(R0), R2 | 13h: | 00 010 000 0001 0000 | 1010h |
| SUB R1, R2, R0 | 14h: | 11 000 001 01000 101 | C145h |
| BEQ fmentre | 15h: | 10 001 000 0001 1011 | 881Bh |
| BLE sinó | 16h: | 10 011 000 0001 1001 | 9819h |
| SUB R1, R2, R1 | 17h: | 11 001 001 010 00 101 | C945h |
| BR mentre | 18h: | 10 000 000 0001 0100 | 8014h |
| SUB R2, R1, R2 | 19h: | 11 010 010 001 00 101 | D225h |
| BR mentre | 1Ah: | 10 000 000 0001 0100 | 8014h |
| STORE R1, mcd(R0) | 1Bh: | 01 001 000 0001 0001 | 4811h |

Taula de símbols

| Símbol | Valor |
|---|---|
| a: | 0Fh |
| b: | 10h |
| mcd: | 11h |
| inici: | 12h |
| mentre: | 14h |
| sinó: | 19h |
| fmentre: | 1Bh |

**37.**

```
programa quadrat;
var a, a2, i: enter;
; es suposa que la variable "a" està inicialitzada
i := 0;
a2 := 0;
mentre (i < a)  fer
        a2 := a2 + a;
        i := i + 1;
fmentre;
fprograma.
```

```
a:        .RW 1;              variable a -> R1
a2:       .RW 1;              variable a2 -> R2
          .BEGIN inici;       variable i -> R3
inici:    LOAD a(R0), R1
          ADD R0, R0, R2
          ADD R0, R0, R3
mentre:   SUB R3, R1, R0
          BGE fmentre
          ADD R1, R2, R2
          ADDI R3, #1, R3
          BR mentre
fmentre:  STORE R2, a2(R0)
          .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| LOAD A(R0), R1 | 02h: | 00 001 000 0000 0000 | 0800h |
| ADD R0, R0, R2 | 03h: | 11 010 000 000 00 100 | D004h |
| ADD R0, R0, R3 | 04h: | 11 011 000 000 00 100 | D804h |
| SUB R3, R1, R0 | 05h: | 11 000 011 001 00 101 | C325h |
| BGE fmentre | 06h: | 10 110 000 0000 1010 | B00Ah |
| ADD R1, R2, R2 | 07h: | 11 010 001 010 00 100 | D144h |
| ADDI R3, #1, R3 | 08h: | 11 011 011 00001 000 | DB08h |
| BR mentre | 09h: | 10 000 000 0000 0101 | 8005h |

STORE R2, a2(R0) 0Ah: 01 010 000 0000 0001 5001h

Taula de símbols:

| Símbol | Valor |
|---|---|
| a: | 00h |
| a2: | 01h |
| inici: | 02h |
| mentre: | 05h |
| fmentre: | 0Ah |

**38.**

a) A continuació es presenta el programa escrit en llenguatge assemblador.

```
        .BEGIN ini
n:      .DW 4               ; variable n
fact:   .RW 1               ; reserva d'espai per al resultat
ini:    LOAD n(R0),R1       ; ind1 -> R1
        ADD R0,R1,R3        ; asumar -> R3
        ADD R0,R0,R4        ; acum -> R4
m1:     SUBI R1,#2,R0       ; mentre (ind1 > 2) fer
        BLE fm1
        SUBI R1,#1,R2       ; ind2 := ind1 -1
m2:     SUBI R2,#0,R0       ; mentre (ind2 > 0) fer
        BLE fm2
        ADD R4,R3,R4        ; acum := acum + asumar
        SUBI R2,#1,R2       ; ind2:= ind2 -1
        BR m2               ; fmentre2
fm2:    ADD R0,R4,R3        ; asumar := acum
        ADD R0,R0,R4        ; acum := 0
        SUBI R1,#1,R1       ; ind1 := ind1 -1
        BR m1               ; fmentre1
fm1:    STORE R3,fact(R0)   ; factorial := asumar
        .END
```

b) Taula de símbols i traducció a llenguatge màquina

Taula de símbols

| Símbol | Valor |
|---|---|
| n | 00h |
| fact | 01h |
| ini | 02h |
| m1 | 05h |
| m2 | 08h |
| fm2 | 0Dh |
| fm1 | 11h |

| | L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|---|
| n: | .DW 4 | 00h | 0000 0000 0000 0100 | 0004h |
| fact: | .RW 1 | 01h | xxxx xxxx xxxx xxxx | xxxxh |
| ini: | LOAD n(R0),R1 | 02h | 00 001 000 00000000 | 0800h |
| | ADD R0,R1,R3 | 03h | 11 011 000 001 00 100 | D824h |
| | ADD R0,R0,R4 | 04h | 11 100 000 000 00 100 | E004h |
| m1: | SUBI R1,#2,R0 | 05h | 11 000 001 00010 001 | C111h |
| | BLE fm1 | 06h | 10 011 000 00010001 | 9811h |
| | SUBI R1,#1,R2 | 07h | 11 010 001 00001 001 | D109h |
| | m2:SUBI R2,#0,R0 | 08h | 11 000 010 00000 001 | C201h |
| | BLE fm2 | 09h | 10 011 000 00001101 | 980Dh |
| | ADD R4,R3,R4 | 0Ah | 11 100 100 011 00 100 | E464h |
| | SUBI R2,#1,R2 | 0Bh | 11 010 010 00001 001 | D209h |
| | BR m2 | 0Ch | 10 000 000 00001000 | 8008h |
| | fm2:ADD R0,R4,R3 | 0Dh | 11 011 000 100 00 100 | D884h |
| | ADD R0,R0,R4 | 0Eh | 11 100 000 000 00 100 | E004h |
| | SUBI R1,#1,R1 | 0Fh | 11 001 001 00001 001 | C909h |
| | BR m1 | 10h | 10 000 000 00000101 | 8005h |
| | fm1:STORE R3,fact(R0) | 11h | 01 011 000 00000001 | 5801h |

**39.**

```
programa lletres_a;
var a, punt, nombre, i : enter;
var frase: vector[0.. 124] de enter;
i := 0;
a := 61h;
punt := 2Eh;
nombre := 0;
mentre (frase[i] <> punt) fer
        si (frase[i] = a) llavors nombre := nombre +1 fsi;
        i := i + 1;
fmentre;
fprograma.
```

```
a:        .DW 61h;              a  -> R1
punt:     .DW 2Eh;              punt -> R2
nombre:   .RW 1;                frase[i] -> R3
frase:    .RW 125;              nombre -> R4
          .ORG 80h
          .BEGIN inici;         i  -> R5
inici:    LOAD a(R0), R1
          LOAD punt(R0), R2
          ADD R0, R0, R5
          ADD R0, R0, R4
mentre:   LOAD frase(R5), R3
          SUB R3, R2, R0
          BEQ fmentre
          SUB R3, R1, R0
          BNE fsi
          ADDI R4, #1, R4
fsi:      ADDI R5, #1, R5
          BR mentre
fmentre:  STORE R4, nombre(R0)
          .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| LOAD a(R0), R1 | 80h: | 00 001 000 0000 0000 | 0800h |
| LOAD punt(R0), R2 | 81h: | 00 010 000 0000 0001 | 1001h |
| ADD R0, R0, R5 | 82h: | 11 101 000 000 00 100 | E806h |
| ADD R0, R0, R4 | 83h: | 11 100 000 000 00 100 | E006h |
| LOAD frase(R5), R3 | 84h: | 00 011 101 0000 0011 | 1D03h |
| SUB R3, R2, R0 | 85h: | 11 000 011 010 00 101 | C345h |
| BEQ fmentre | 86h: | 10 001 000 1000 1100 | 888Ch |
| SUB R3, R1, R0 | 87h: | 11 000 011 001 00 101 | C325h |
| BNE fsi | 88h: | 10 101 000 1000 1010 | A88Ah |
| ADDI R4, #1, R4 | 89h: | 11 100 100 00001 000 | E408h |
| ADDI R5, #1, R5 | 8Ah: | 11 101 101 00001 000 | ED08h |
| BR mentre | 8Bh: | 10 000 000 1000 0100 | 8084h |
| STORE R4, num(R0) | 8Ch: | 01 100 000 0000 0010 | 6002h |

Taula de símbols:

| Símbol | Valor |
|---|---|
| a: | 00h |
| punt: | 01h |
| num: | 02h |
| frase: | 03h |
| inici: | 80h |
| mentre: | 84h |
| fsi: | 8Ah |
| fmentre: | 8Ch |

40.

```
programa vector;
const n=3
var suma, max, min, nombre, i : enter; var v :vector[0.. n] de enter;
 i := 1;
 suma := v[0];
 max := v[0];
 min := v[0];
mentre (n > i)  fer
        nombre := v[i];
        i := i + 1;
        suma := suma + nombre;
        si (nombre > max) llavors max := nombre fsi;
        si (nombre < min) llavors min := nombre fsi;
fmentre;
fprograma.
```

```
suma:     .RW 1
max:      .RW 1
min:      .RW 1
          N=3
n:        .DW N
vector:   .RW N
          .ORG 3Ah
          .BEGIN inici
inici:    LOAD n(R0), R2;          n ->R2
          ADDI R0, #1, R1;         i -> R1
          LOAD vector(R0), R6;     nombre -> R6
          ADD R6, R0, R3;          suma -> R3
          ADD R6, R0, R4;          max -> R4
          ADD R6, R0, R5;          min -> R5
mentre:   SUB R2, R1, R0
          BLE fmentre
          LOAD vector(R1), R6
          ADDI R1, #1, R1
          ADD R3, R6, R3
amax:     SUB R6, R4, R0
          BLE amin
          ADD R6, R0, R4
amin:     SUB R6, R5 R0
          BGE mentre
          ADD R6, R0, R5
          BR mentre
fmentre:  STORE R3, suma(R0)
          STORE R4, max(R0)
          STORE R5, min(R0)
          .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| LOAD n(R0), R2 | 3Ah: | 00 010 000 0000 0011 | 1003h |
| ADDI R0, #1, R1 | 3Bh: | 11 001 000 00001 000 | C808h |
| LOAD vector(R0), R6 | 3Ch: | 00 110 000 0000 0100 | 3004h |
| ADD R6, R0, R3 | 3Dh: | 11 011 110 00000 100 | DE04h |
| ADD R6, R0, R4 | 3Eh: | 11 100 110 00000 100 | E604h |
| ADD R6, R0, R5 | 3Fh: | 11 101 110 00000 100 | EE04h |
| SUB R2, R1, R0 | 40h: | 11 000 010 001 00 101 | C225h |
| BLE fmentre | 41h: | 10 011 000 0100 1100 | 984Ch |
| LOAD vector(R1), R6 | 42h: | 00 110 001 00000100 | 3104h |
| ADDI R1, #1, R1 | 43h: | 11 001 001 00001 000 | C908h |
| ADD R3, R6, R3 | 44h: | 11 011 011 11000 100 | DBC4h |
| SUB R6, R4, R0 | 45h: | 11 000 110 100 00 101 | C685h |
| BLE amin | 46h: | 10 011 000 0100 1000 | 9848h |
| ADD R6, R0, R4 | 47h: | 11 100 110 00000 100 | E604h |
| SUB R6, R5 R0 | 48h: | 11 000 110 10100 101 | C6A5h |
| BGE mentre | 49h: | 10 110 000 0100 0000 | B040h |
| ADD R6, R0, R5 | 4Ah: | 11 101 110 00000 100 | EE04h |
| BR mentre | 4Bh: | 10 000 000 0100 0000 | 8040h |
| STORE R3, suma(R0) | 4Ch: | 01 011 000 0000 0000 | 5800h |
| STORE R4, max(R0) | 4Dh: | 01 100 000 0000 0001 | 6001h |

STORE R5, min(R0) 4Eh: 01 101 000 0000 0010 6802h

Taula de símbols

| Símbol | Valor |
|---|---|
| suma: | 00h |
| max: | 01h |
| min: | 02h |
| n: | 03h |
| vector: | 04h |
| inici: | 3Ah |
| mentre: | 40h |
| amax: | 45h |
| amin: | 48h |
| fmentre: | 4Ch |

**41.**

```
programa suma_vector;
var a, b, c: vector[0..9] de enter;
var i : enter;
i := 0;
mentre (i < 10) fer
        c[i] := a[i] + b[i];
        i := i + 1;
fmentre;
fprograma.
```

```
a:      .RW 10 ;        a[i] -> R2
b:      .RW 10 ;        b[i] -> R3
c:      .RW 10
        .ORG 80h
        .BEGIN inici
inici:  ADD R0, R0, R1; i -> R1
mentre: SUB R1, #10, R0
        BGE fmentre
        LOAD a(R1), R2
        LOAD b(R1), R3
        ADD R2, R3, R3
        STORE R3, c(R1)
        ADDI R1, #1, R1
        BR mentre
fmentre: .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| ADD R0, R0, R1 | 80h: | 11 001 000 000 00 100 | C804h |
| SUBI R1, #10, R0 | 81h: | 11 000 001 01010 001 | C151h |
| BGE fmentre | 82h: | 10 110 000 1000 1001 | B089h |
| LOAD a(R1), R2 | 83h: | 00 010 001 0000 0000 | 1100h |
| LOAD b(R1), R3 | 84h: | 00 011 001 0000 1010 | 190Ah |
| ADD R2, R3, R3 | 85h: | 11 011 010 011 00 100 | DA64h |
| STORE R3, c(R1) | 86h: | 01 011 001 0001 0100 | 5914h |
| ADDI R1, #1, R1 | 87h: | 11 001 001 00001 000 | C908h |
| BR mentre | 88h: | 10 000 000 1000 0001 | 8081h |

Taula de símbols

| Símbol | Valor |
|---|---|
| a: | 00h |
| b: | 0Ah |
| c: | 14h |
| inici: | 80h |
| mentre: | 81h |
| fmentre: | 89h |

**42.**

a) clr Rd

```
.DEF clr $1
        ADD R0, R0, $1
.ENDDEF
```

b) clr A(Ri)

```
.DEF clr $d1
        STORE R0, $d1
.ENDDEF
```

c) inc Rd

```
.DEF inc $1
        ADDI $1, #1, $1
.ENDDEF
```

d) dec Rd

```
.DEF dec $1
        SUBI $1, #1, $1
.ENDDEF
```

e) add3 Rf1, Rf2, Rd

```
.DEF add3 $1, $2, $3
        ADD $1, $3, $3
        ADD $2, $3, $3
.ENDDEF
```

f) asl Rd

```
.DEF asl $1
        ADD $1, $1, $1
.ENDDEF
```

g) mov Rf, Rd

```
.DEF mov $1, $2
        ADD $1, R0, $2
.ENDDEF
```

h) MOV Rf, A(Ri)

```
.DEF MOV $1, $d1
        STORE $1, $d1
.ENDDEF
```

i) mov A(Ri), B(Rj)

```
.DEF mov $d1, $d2
        LOAD $d1, R7
        STORE R7, $d2
.ENDDEF
```

j) swap Rf1, Rf2

```
.DEF swap $1, $2
        ADD R0, $1, R7
        ADD $2, R0, $1
        ADD R7, R0, $2
.ENDDEF
```

k) swap A(Ri), B(Rj)

```
.DEF swap $d1, $d2
        LOAD $d1, R6
        LOAD $d2, R7
        STORE R7, $d1
        STORE R6, $d2
.ENDDEF
```

l) asrn #n, Rd

```
.DEF asrn $n1, $1
                ADDI R0, $n1, R7
        basr:   BEQ fasr
                ASR $1, $1
                SUBI R7, #1, R7
                BR basr
        fasr:
.ENDDEF
```

m) asln #n, Rd

```
.DEF asln $n1, $1
                ADD R0, $n1, R7
        basl:   BEQ fasl
                ADD $1, $1, $1
                SUBI R7, #1, R7
                BR basl
        fasl:
.ENDDEF
```

n) inc A(Ri)

```
.DEF inc $d1
        LOAD $d1, R7
        ADD R7, #1, R7
        STORE R7, $d1
.ENDDEF
```

**43.**

```
programa suma_vector;
var suma, max, min, n : enter;
var v: vector[0..N] de enter;
var i : enter;
i := 0;
suma := 0;
mentre (i < n) fer
        si (v[i] < max) i (v[i] > min)
                llavors suma := suma + v[i];
        fsi;
        i := i + 1;
fmentre;
fprograma.
```

```
        N=7
suma:   .RW 1               ; n -> R5
max:    .RW 1               ; i -> R6
min:    .RW 1
n:      .RW 1
vector: .RW N
        .ORG 3Ah
        .BEGIN inici
inici:  LOAD max(R0), R3;   max -> R3
        LOAD min(R0), R4;   min -> R4
        LOAD n(R0), R5;     N -> R5
        ADD R0, R0, R1;     suma -> R1
        ADD R0, R0, R6;     i -> R6
mentre: SUB R6, R5, R0
        BGE fmentre
        LOAD vector(R6), R2;    element del vector -> R2
        SUB R2, R3, R0
        BGE fsi
        SUB R2, R4, R0
        BLE fsi
        ADD R1, R2, R1
fsi:    ADDI R6, #1, R6
        BR mentre
fmentre: STORE R1, suma(R0)
        .END
```

| L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|
| LOAD max(R0), R3 | 3Ah: | 00 011 000 0000 0001 | 1801h |
| LOAD min(R0), R4 | 3Bh: | 00 100 000 00000010 | 2002h |
| LOAD n(R0), R5 | 3Ch: | 00 101 000 0000 0011 | 2803h |
| ADD R0, R0, R1 | 3Dh: | 11 001 000 000 00 100 | C804h |
| ADD R0, R0, R6 | 3Eh: | 11 110 000 000 00 100 | F004h |
| SUB R6, R5, R0 | 3Fh: | 11 000 110 101 00 101 | C6A5h |
| BGE fmentre | 40h: | 10 110 000 01001001 | D049h |
| LOAD vector(R6), R2 | 41h: | 00 010 110 0000 0100 | 1604h |
| SUB R2, R3, R0 | 42h: | 11 000 010 011 00 101 | C265h |
| BGE fsi | 43h: | 10 110 000 0100 0111 | D047h |
| SUB R2, R4, R0 | 44h: | 11 000 010 100 00 101 | C285h |
| BLE fsi | 45h: | 10 011 000 0100 0111 | 9847h |
| ADD R1, R2, R1 | 46h: | 11 001 001 010 00 100 | C944h |
| ADDI R6, #1, R6 | 47h: | 11 110 110 00001 000 | F608h |
| BR mentre | 48h: | 10 000 000 0011 1111 | 803Fh |
| STORE R1, suma(R0) | 49h: | 01 001 000 0000 0000 | 4800h |

Taula de símbols

| Símbol | Valor |
|---|---|
| suma: | 00h |
| max: | 01h |
| min: | 02h |
| n: | 03h |
| vector: | 04h |
| inici: | 3Ah |
| mentre: | 3Fh |
| fsi: | 47h |
| fmentre: | 49h |

**44.**

a) A continuació presentem el disseny de la macro mul $1, $2, $3, que usa tres registres diferents.

```
.DEF mul $1, $2, $3
  ; $3 := $1 * $2
  ; $3, $2 i $1 han d'ésser registres diferents
            BR ini
      aux:  .RW 1                 ;variable auxiliar per a guardar $2
      ini:  STORE $2, aux(R0)     ;salva el valor inicial del registre
                                  ;$2 a aux
            ADDI R0, #0, $3       ;$3 := 0
      loop: SUBI $2, #0, R0       ;mentre $2 > 0 fer
            BLE fi
            ADD $3, $1, $3        ;$3:=$3+$1
            SUBI $2, #1, $2       ;$2:=$2-1
            BR loop               ;fmentre
      fi:   LOAD aux(R0),$2       ;restaura el valor inicial
                                  ;del registre $2
.ENDDEF
```

A continuació presentem la traducció del programa, que usa la macro mul definida anteriorment.

```
            .BEGIN inici
            ; les variables s'emmagatzemen en registres
            ; R1 és "a", R2 és "cont", R3 és "factorial",
            ; R4 és un registre auxiliar
inici:      SUBI R0,#-8,R1          ; R1 := 8
            ADDI R0,#1,R2           ; R2 := 1
            ADDI R0,#1,R3           ; R3 := 1
mentre:     SUB R2,R1,R0            ; mentre R2 < R1 fer
            BGE fmentre
            ADDI R2,#1,R2           ; R2 := R2 + 1
            mul R3, R2, R4          ; els tres registres han d'ésser diferents
            ADD R4,R0,R3            ; R3 := R3 * R2
            BR mentre               ; fmentre
fmentre:    .END
```

b) A continuació presentem el preassemblatge del programa. Per claredat, hem indicat el punt on es comença a expandir la macro i el punt on acaba l'expansió. Aquests límits han estat indicats amb un guió de subratllat davant el nom de la macro per a indicar el principi de l'expansió, i la paraula _end davant el nom de la macro per a indicar el final.

```
          .BEGIN inici
inici:     SUBI R0,#-8,R1
           ADDI R0,#1,R2
           ADDI R0,#1,R3
mentre: SUB R2,R1,R0
            BGE fmentre
           ADDI R2,#1,R2

          _mul R3,R2,R4
                      BR $010102
           $010101:   .RW 1
           $010102:   STORE R2,$010101(R0)
                      ADDI R0,#0,R4
           $010103:   SUBI R2,#0,R0
```

```
                    BLE $010104
                    ADD R4,R3,R4
                    SUBI R2,#1,R2
                    BR $010103
        $010104:    LOAD $010101(R0),R2
    _end_mul R3,R2,R4

        ADD R4,R0,R3
         BR mentre
    fmentre: . END
```

A continuació presentem el mateix programa un cop eliminades les indicacions de començament i final de macro.

```
            .BEGIN inici
inici:      SUBI R0,#-8,R1
            ADDI R0,#1,R2
            ADDI R0,#1,R3
mentre:     SUB R2,R1,R0
            BGE fmentre
            ADDI R2,#1,R2
            BR $010102
$010101:    .RW 1
$010102:    STORE R2,$010101(R0)
            ADDI R0,#0,R4
$010103:    SUBI R2,#0,R0
            BLE $010104
            ADD R4,R3,R4
            SUBI R2,#1,R2
            BR $010103
$010104:    LOAD $010101(R0),R2
            ADD R4,R0,R3
            BR mentre
fmentre:    .END
```

A continuació presentem la taula de símbols i la traducció a llenguatge màquina del programa.

Taula de símbols

| Símbol | Valor |
|---|---|
| inici | 00h |
| mentre | 03h |
| $010101 | 07h |
| $010102 | 08h |
| $010103 | 0Ah |
| $010104 | 0Fh |
| fmentre | 12h |

| | L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|---|
| inici: | SUBI R0,#-8,R1 | 00h | 11 001 000 11000 001 | C8C1h |
| | ADDI R0,#1,R2 | 01h | 11 010 000 00001 000 | D008h |
| | ADDI R0,#1,R3 | 02h | 11 011 000 00001 000 | D808h |
| mentre: | SUB R2,R1,R0 | 03h | 11 000 010 001 00 101 | C225h |
| | BGE fmentre | 04h | 10 110 000 00010010 | B012h |
| | ADDI R2,#1,R2 | 05h | 11 010 010 00001 000 | D208h |
| | BR $010102 | 06h | 10 000 000 00001000 | 8008h |
| $010101: | .RW 1 | 07h | xxxxxxxxxxxxxxxx | xxxxh |
| $010102: | STORE R2,$010101(R0) | 08h | 01 010 000 00000111 | 5007h |
| | ADDI R0,#0,R4 | 09h | 11 100 000 00000000 | E000h |
| $010103: | SUBI R2,#0,R0 | 0Ah | 11 000 010 00000 001 | C201h |
| | BLE $010104 | 0Bh | 10 011 000 00001111 | 980Fh |
| | ADD R4,R3,R4 | 0Ch | 11 100 100 011 00 100 | E464h |
| | SUBI R2,#1,R2 | 0Dh | 11 010 010 00001 001 | D209h |
| | BR $010103 | 0Eh | 10 000 000 00001010 | 800Ah |
| $010104: | LOAD $010101(R0),R2 | 0Fh | 00 010 000 00000111 | 1007h |
| | ADD R4,R0,R3 | 10h | 11 011 100 000 00 100 | DC04h |
| | BR mentre | 11h | 10 000 000 00000011 | 8003h |

| | | | |
|---|---|---|---|
| fmentre: | 12h | xxxxxxxxxxxxxxxx | xxxxh |

**45.**

a) A continuació presentem el preassemblatge del programa. Per claredat, hem indicat el punt on es comença a expandir la macro i el punt on acaba l'expansió. Aquests límits els hem indicat amb un guió de subratllat davant el nom de la macro per a indicar el principi de l'expansió, i la paraula _end davant el nom de la macro per a indicar el final.

```
            .BEGIN ini
n:          .DW 4
resul:      .RW 1
ini:        _mov n(R0),R1
                LOAD n(R0),R1
            _end_mov n(R0),R1
            _mov R1,R3
                ADD R0,R1,R3
            _end_mov R1,R3
            _clr R4
                ADD R0,R0,R4
            _end_clr R4
m1:         _cmp R1,#2
                SUBI R1,#2,R0
            _end_cmp R1,#2
                BLE fm1
                SUBI R1,#1,R2
m2:         _cmp R2,#0
                SUBI R2,#0,R0
            _end_cmp R2,#0
                BLE fm2
                ADD R4,R3,R4
            _dec R2
                SUBI R2,#1,R2
            _end_dec R2
                BR m2
fm2:        _mov R4,R3
                ADD R0,R4,R3
            _end_mov R4,R3
            _clr R4
                ADD R0,R0,R4
            _end_clr R4
            _dec R1
                SUBI R1,#1,R1
            _end_dec R1
                BR m1
fm1:        _mov R3,resul(R0)
                STORE R3,resul(R0)
            _end_mov R3,resul(R0)
            .END
```

A continuació presentem el mateix programa un cop eliminades les indicacions de començament i final de macro.

```
            .BEGIN ini
n:          .DW 4
resul:      .RW 1
ini:        LOAD n(R0),R1
            ADD R0,R1,R3
            ADD R0,R0,R4
m1:         SUBI R1,#2,R0
            BLE fm1
            SUBI R1,#1,R2
m2:         SUBI R2,#0,R0
            BLE fm2
            ADD R4,R3,R4
            SUBI R2,#1,R2
            BR m2
fm2:        ADD R0,R4,R3
            ADD R0,R0,R4
            SUBI R1,#1,R1
            BR m1
fm1:        STORE R3,resul(R0)
```

```
        .END
```

b) Taula de símbols i traducció a llenguatge màquina

Taula de símbols

| Símbol | Valor |
|---|---|
| n | 00h |
| resul | 01h |
| ini | 02h |
| m1 | 05h |
| m2 | 08h |
| fm2 | 0Dh |
| fm1 | 11h |

| | L. Assemblador | Adreça | L. Màquina | Hexadecimal |
|---|---|---|---|---|
| n: | .DW 4 | 00h | 0000 0000 0000 0100 | 0004h |
| resul: | .RW 1 | 01h | xxxx xxxx xxxx xxxx | xxxxh |
| ini: | LOAD n(R0),R1 | 02h | 00 001 000 00000000 | 0800h |
| | ADD R0,R1,R3 | 03h | 11 011 000 001 00 100 | D824h |
| | ADD R0,R0,R4 | 04h | 11 100 000 000 00 100 | E004h |
| m1: | SUBI R1,#2,R0 | 05h | 11 000 001 00010 001 | C111h |
| | BLE fm1 | 06h | 10 011 000 00010001 | 9811h |
| | SUBI R1,#1,R2 | 07h | 11 010 001 00001 001 | D109h |
| m2: | SUBI R2,#0,R0 | 08h | 11 000 010 00000 001 | C201h |
| | BLE fm2 | 09h | 10 011 000 00001101 | 980Dh |
| | ADD R4,R3,R4 | 0Ah | 11 100 100 011 00 100 | E464h |
| | SUBI R2,#1,R2 | 0Bh | 11 010 010 00001 001 | D209h |
| | BR m2 | 0Ch | 10 000 000 00001000 | 8008h |
| fm2: | ADD R0,R4,R3 | 0Dh | 11 011 000 100 00 100 | D884h |
| | ADD R0,R0,R4 | 0Eh | 11 100 000 000 00 100 | E004h |
| | SUBI R1,#1,R1 | 0Fh | 11 001 001 00001 001 | C909h |
| | BR m1 | 10h | 10 000 000 00000101 | 8005h |
| fm1: | STORE R3,resul(R0) | 11h | 01 011 000 00000001 | 5801h |

c) Les operacions que realitza cada macro són les següents:

- cmp $1,$2: Compara dos registres (fa la resta) i actualitza el valor de l'indicador de condició Z.
- cmp $1,$i2: Compara un registre i un valor immediat (fa la resta) i actualitza el valor de l'indicador de condició Z
- clr $1: Neteja (posa a zero) el valor d'un registre
- mov $1, $2: Copia un valor del registre font ($1) al registre destinació ($2)
- mov $i1,$2: Copia un valor immediat en un registre.
- mov $d1, $2: Copia el valor d'una posició de memòria en un registre
- mov $1,$d2: Copia el valor d'un registre en una posició de memòria
- inc $1: Suma "1" al valor emmagatzemat en un registre
- dec $1: Resta "1" al valor emmagatzemat en un registre

Per simplicitat, posarem noms a les variables que es guarden en els registres abans de traduir el programa a alt nivell.

```
n:      .DW 4
resul:  .RW 1
        .BEGIN ini
ini:    mov n(R0), R1;      R1 = ind1
        mov R1, R3;         R3 = a_sumar
        clr R4;             R4 = acum
m1:     cmp R1, #2;         ind1 > 2?
        BLE fm1
        SUBI R1, #1, R2
m2:     cmp R2, #0;         ind2 > 0?
        BLE fm2
        ADD R4,R3,R4;       acum := acum + a_sumar
        dec R2;             ind2 := ind2 - 1
        BR m2
fm2:    mov R4,R3;          a_sumar := acum
        clr R4;             acum := 0
        dec R1;             ind1 := ind1 - 1
        BR m1
fm1:    mov R3, resul(R0);  resul := a_sumar
        .END
```

La traducció del programa a alt nivell és la següent:

```
programa càlcul;
const n=4;
var resul : enter;
var ind1, ind2, a_sumar, acum : enter;
ind1 := n;
a_sumar := n;
acum := 0;
mentre (ind1 > 2) fer
        ind2 := ind1 - 1;
        mentre (ind2 > 0) fer
                acum := acum + a_sumar;
                ind2 := ind2 - 1;
        fmentre;
        a_sumar := acum;
        acum := 0;
        ind1 := ind1 - 1;
fmentre;
resul := a_sumar;
fprograma.
```

El programa calcula el factorial d'un nombre. En l'exemple es calcula el factorial de 4.