1. Arquitectura d’un computador

Estructura bàsica d’un computador

El 1946, A. W. Burks, H. H. Goldstine i J. von Neumann, investigadors de l’*Institute for Advanced Study*, van publicar un article titulat *“Preliminary discussion of the logical design of an electronic computing instrument”*, que va establir les bases per al disseny de computadors que s’han seguit fins al dia d’avui.

El disseny bàsic que van proposar, que es coneix com a Arquitectura von Neumann (Figura 1.1), estructura els computadors en tres mòduls principals:

- la **Memòria** guarda els ***programes***, que són un conjunt de ***dades*** que es modificaran o consultaran en executar el programa, més un conjunt d’***instruccions*** que indiquen les operacions que s’han de fer sobre les dades. La memòria està estructurada com un conjunt de ***paraules***, cadascuna de les quals és capaç de guardar una instrucció o una dada individual. A la ubicació de cada paraula dins de la memòria se li diu ***adreça***.
- la **Unitat Central de Procés**, **processador** o **CPU** (*Central Processing Unit*) executa els programes. Al seu torn, s’estructura en dos blocs: la ***Unitat de Procés*** (UP) és el conjunt de dispositius electrònics necessaris per a executar programes, i la ***Unitat de Control*** (UC) és un sistema seqüencial encarregat de governar la circuiteria de la Unitat de Procés.
- la **Unitat d’Entrada/Sortida** (*Input/Output*) permet relacionar el computador amb l’exterior, mitjançant diversos dispositius perifèrics (teclat, pantalla, disc etc.)

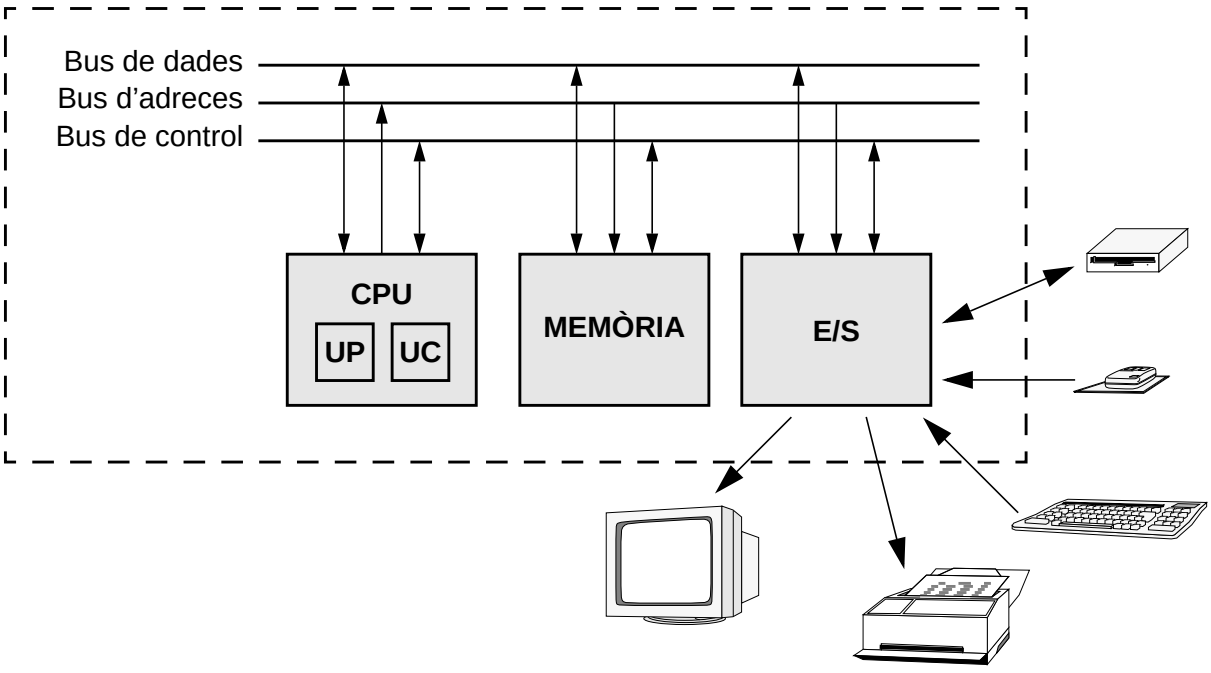


Figura 1.1. Arquitectura von Neumann

Els tres mòduls es comuniquen a través de tres busos:

- el **Bus de Dades** permet la transferència de dades entre tots tres mòduls, i la d'instruccions des de la memòria cap a la CPU.
- el **Bus d'Adreces** permet a la CPU indicar a la memòria (o a la Unitat d'E/S) quina de les paraules vol *llegir* (obtenir-ne el valor) o *escriure* (modificar-ne el valor).
- a través del **Bus de Control** es transmeten els senyals que possibiliten que les tres unitats actuïn de manera coordinada.

Els tres mòduls de l'arquitectura von Neumann constitueixen la circuiteria electrònica o ***nivell físic*** d'un computador, o ***hardware***. El hardware és gestionat pels programes, que constitueixen el ***software***. El software s'estructura en nivells (Figura 1.2):

- **Llenguatge màquina**, o *llenguatge de baix nivell*: els programes d'aquest nivell són els que interactuen directament amb el hardware. Les instruccions són ordres senzilles que es dónen a la circuiteria, escrites en binari.
- **Sistema operatiu**: conjunt de programes que possibiliten un ús còmode del computador per part d'un o més usuaris, permetent que aquests no hagin de conèixer en detall la circuiteria. Alguns sistemes operatius són Windows95, Unix o OSF.
- **Llenguatges d'alt nivell**: llenguatges més propers al llenguatge natural que el binari, que permeten per tant escriure programes de manera còmoda. Són lleguatges d'alt nivell el C, el Fortran i el VisualBasic.

  Els programes escrits en alt nivell s'han de traduir a llenguatge màquina per ser executats. A la traducció se li diu ***compilació***.
- **Aplicacions**: programes escrits en algun llenguatge d'alt nivell i posats a disposició dels usuaris del computador. Aquests només han de conèixer l'aparièncía de les aplicacions per a utilitzar el computador, podent ignorar tots els nivells inferiors.

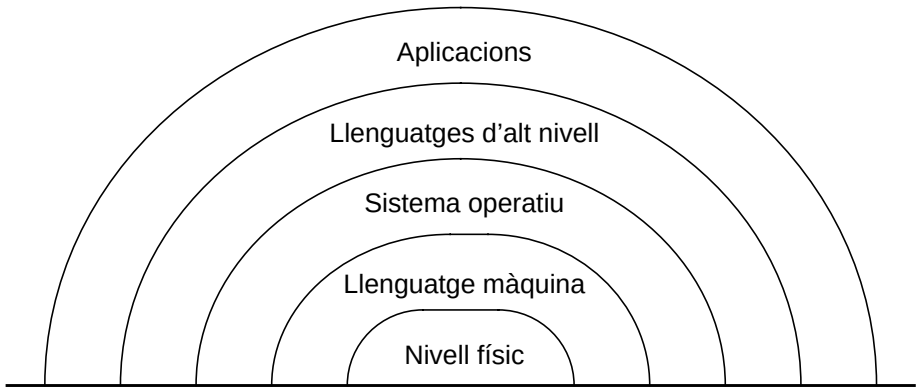


Figura 1.2. Jerarquia de nivells d'un computador.

Funcionament dels computadors

El funcionament bàsic d'un computador és el següent. La Unitat d'E/S permet carregar programes a la memòria, ja sigui des d'una unitat de disc (disc dur, CD-Rom etc.) o a través del teclat. Per executar el programa, les instruccions es transmeten d'una en una des de la memòria cap a la CPU; aquesta les examina i activa els circuits apropiats per dur a terme les operacions indicades per cada instrucció. Per fer una operació sobre unes dades determinades, aquestes també s'han d'haver transferit prèviament des de la memòria cap al processador. Les dades que resulten de l'execució dels programes es transfereixen de la CPU cap a la memòria.

Habitualment, els programes es guarden a la memòria de manera que les instruccions ocupen un bloc de paraules consecutives i les dades ocupen un altre bloc de paraules consecutives. Les instruccions s'executen segons el ***seqüenciament implícit***: es comença per la instrucció que hi ha en una adreça inicial i a continuació s'executen en seqüència les instruccions que hi ha en adreces successives. Aquest seqüenciament es pot trencar mitjançant un tipus d'instruccions que s'anomenen ***instruccions de salt***.

Arquitectura de la Màquina Rudimentària

La Màquina Rudimentaria és un computador senzill i pedagògic basat en l'arquitectura von Neumann. No té Unitat d'E/S, donat que el funcionament d'aquest mòdul s'estudiarà en cursos posteriors. Així, assumirem que els programes són a la memòria sense preguntar-nos com hi han arribat.

A la Figura 1.3 es pot veure l'arquitectura de l'MR. Observem que disposa de dos busos de dades per comunicar la CPU i la memòria, un en cada sentit.

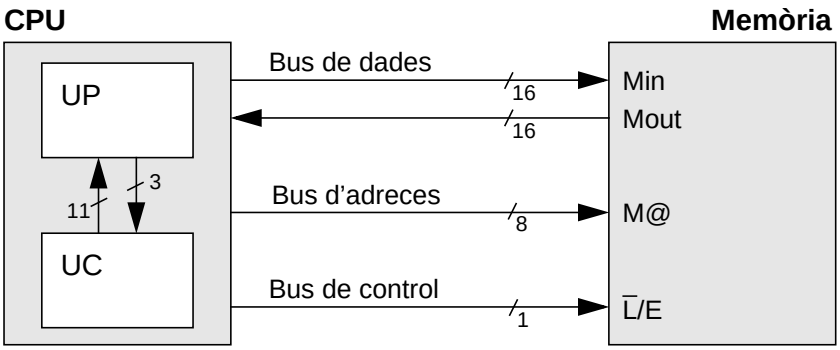


Figura 1.3. Arquitectura de la Màquina Rudimentària

La memòria

Les instruccions i les dades de l'MR es codifiquen en 16 bits; per tant, les paraules de memòria i els busos de dades són de 16 bits d'amplada. La memòria té 256 paraules, i per tant el bus d'adreces és de 8 bits. Pel bus de control hi viatja un sol senyal, $\overline{L}$/E, que indica en cada moment a la memòria si es vol realitzar una operació de lectura (0) o una escriptura (1).

Per referir-nos al contingut de la paraula que hi ha a l'adreça ***a*** usarem la notació ***M[a]***. Per exemple, a la Figura 1.4 es compleix que M[FDh] = D964h.

ADREÇA	CONTINGUT
00h	6D62h
01h	2346h
02h	8675h
03h	0005h
04h	0087h
⋮	⋮
FDh	D964h
FEh	CA10h
FFh	0091h

Figura 1.4. Exemple de memòria

Les dades amb què treballa la Màquina Rudimentària són números enters codificats en 16 bits i en complement a 2. Pot representar per tant el rang [-32.768, 32.767].

La Unitat Central de Procés (CPU)

Com ja hem vist, la CPU es divideix en la Unitat de Procés (UP) i la Unitat de Control (UC). La segona rep 3 bits d'informació de la primera, i li envia ordres a través d'11 senyals. En les seccions 3 i 4 veurem el significat d'aquests 14 senyals.

La UP ha de ser capaç de realitzar totes les accions necessàries per a l'execució d'instruccions. Disposa, entre d'altres, dels dispositius següents:

- **Registre d'Instruccions** (*Instruction Register*, **IR**): és un registre de 16 bits que guarda en cada moment la instrucció que s'està executant.
- **Comptador de Programa** (*Program Counter*, **PC**): és un registre de 8 bits encarregat de guardar en tot moment l'adreça següent a la de la instrucció que s'està executant.
- **Banc de Registres**: conjunt de registres que permeten guardar les dades amb què treballen els programes quan es porten de la memòria a la CPU per a operar amb elles o consultar-les. Hi ha 8 registres de 16 bits, que s'identifiquen pels noms R0, R1, ..., R7. El registre R0 té la particularitat que no s'hi pot escriure i sempre conté un 0.

- **Unitat Aritmètica i Lògica** (*Arithmetic-Logic Unit,* **ALU**): és el circuit capaç de fer les operacions aritmètiques i lògiques requerides pels programes.
- **Bits de Condició** *(flags)*: són dos bits que indiquen en tot moment si el resultat de l'última operació realitzada per l'ALU ha estat negatiu (bit N) o zero (bit Z).

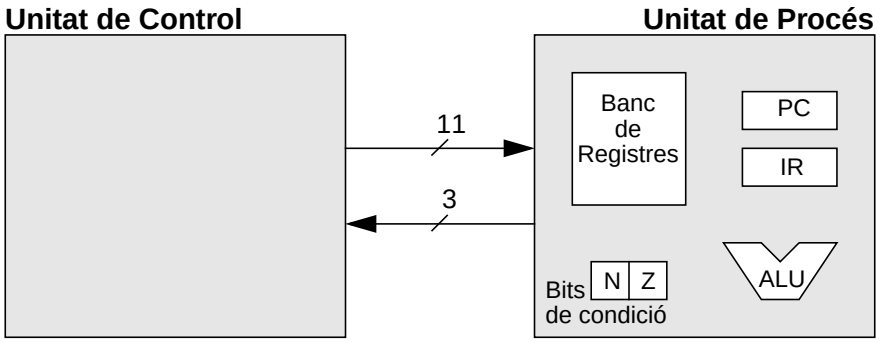


Figura 1.5. Esquema de la CPU de l'MR.

2. Llenguatge Màquina de la Màquina Rudimentària

El Llenguatge Màquina (LM) d'un computador és un conjunt d'instruccions senzilles codificades en binari que actuen directament sobre la circuiteria de la CPU. Donat que una tira de 0's i 1's és incòmoda de manejar pels humans, tots els computadors disposen del que s'anomena ***Llenguatge Assemblador*** (LA), que té les mateixes instruccions que el Llenguatge Màquina però escrites usant termes mnemotècnics. Els programes escrits en LA són traduïts a LM pel programa ***assemblador***. En aquest capítol veurem quines són les instruccions de la Màquina Rudimentària, primer en LA i després en LM. Les instruccions LM de la Màquina Rudimentària es codifiquen en 16 bits.

Tipus d'instruccions

El Llenguatge Màquina de l'MR té tres tipus d'instruccions:

- **Instruccions d'Accés a Memòria:** donen ordre de transferir una dada de la memòria cap a la CPU (concretament cap al Banc de Registres) o al revés.
- **Instruccions Aritmètiques i Lògiques (A-L):** donen ordre de fer operacions aritmètiques o lògiques sobre dades dels programes.
- **Instruccions de Salt:** donen ordre de trencar el seqüenciament implícit si es compleix una condició determinada.

Cada instrucció té un cert nombre d'operands. Els ***operands font*** indiquen on són les dades que ha de fer servir la instrucció, i l'***operand destí*** indica on s'ha de guardar el resultat de l'execució de la instrucció, en cas que n'hi hagi. Les instruccions de salt no utilitzen cap dada ni generen cap resultat, només indiquen a quina adreça s'haurà de saltar en cas que es compleixi la condició.

L'execució d'algunes instruccions (no totes) fa que els bits de condició s'actualitzin d'acord amb el resultat:

$$N := \begin{cases} 1 \text{ si el resultat} < 0 \\ 0 \text{ si no} \end{cases} \qquad Z := \begin{cases} 1 \text{ si el resultat} = 0 \\ 0 \text{ si no} \end{cases}$$

Instruccions d’accés a memòria

En aquestes instruccions l’acció que es fa és un moviment d’una dada entre el processador i la memòria. N’hi ha dues, una per cada sentit de la transferència:

LOAD *font*, *destí*

Càrrega, transferència del contingut d’una paraula de memòria cap a un registre del Banc de Registres. L’operand font indica per tant una adreça de memòria i l’operand destí un registre del Banc. Els bits de condició s’actualitzen d’acord amb el valor de la dada transferida.

STORE *font*, *destí*

Emmagatzematge, transferència del contingut d’un registre cap a una adreça de memòria. Els bits de condició no es modifiquen.

Per indicar un registre en Llenguatge Assemblador es fa servir el seu nom; per exemple, R3. Per indicar una adreça de memòria s’escriu

adreça_base(Ri)

on *adreça_base* és un número natural en el rang [0, 255] i *Ri* és el nom d’un registre del Banc, que en aquest context es diu que fa el paper de ***registre índex***. L’***adreça efectiva*** de l’operand, és a dir, la seva adreça a la memòria, es calcula així:

adreça_efectiva = *adreça_base* + *desplaçament*

essent ***desplaçament*** el número enter guardat en els 8 bits més baixos del registre índex. Aquesta suma la fa el processador durant l’execució de la instrucció.

Aquesta manera d’especificar una adreça efectiva (“*mode base+desplaçament”*) permet accedir de manera còmoda tant a dades simples com a dades estructurades (vectors).

A continuació es mostren alguns exemples d’execució d’instruccions d’accés a memòria; els valors modificats per la instrucció estan en negreta.

LOAD 0(R3), R1

	ABANS	**DESPRÉS**
Memòria		
0Ch	1101 0100 0101 0101	1101 0100 0101 0101
Registres		
R1	xxxx xxxx xxxx xxxx	**1101 0100 0101 0101**
R3	xxxx xxxx 0000 1100	xxxx xxxx 0000 1100
Bit Z	x	**0**
Bit N	x	**1**

LOAD 12(R2), R2

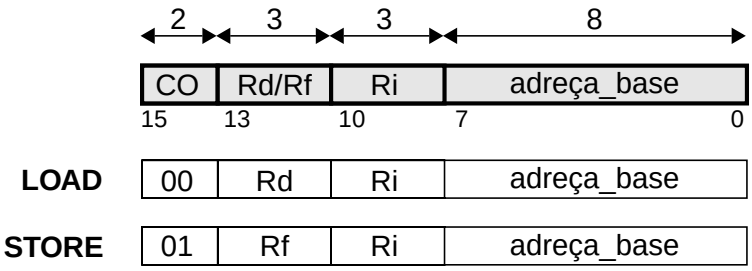
	ABANS	DESPRÉS
Memòria		
0Bh	0110 1111 0101 1011	0110 1111 0101 1011
0Ch	xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx
Registres		
R2	1111 1111 1111 1111	**0110 1111 0101 1011**
Bit Z	x	**0**
Bit N	x	**0**

STORE R5, 11(R0)

	ABANS	DESPRÉS
Memòria		
0Bh	xxxx xxxx xxxx xxxx	**0000 0000 0000 1011**
Registres		
R0	0000 0000 0000 0000	0000 0000 0000 0000
R5	0000 0000 0000 1011	0000 0000 0000 1011
Bit Z	x	x
Bit N	x	x

Codificació en LM

Les instruccions d'accés a memòria es codifiquen en Llenguatge Màquina de la manera següent:



- el camp CO (*Codi d'Operació*, bits 15:14) indica de quin tipus d'instrucció es tracta
- els camps que codifiquen registres (Rd, Rf i Ri) tenen 3 bits, donat que el Banc té 8 registres; per fer referència al registre *Rx* es codifica el número *x* en binari natural

- el camp *adreça_base* té 8 bits, donat que ha de poder guardar números naturals entre 0 i 255.

Vegem alguns exemples de codificació d'instruccions d'accés a memòria en Llenguatge Màquina:

Llenguatge Assemblador	**Llenguatge Màquina**	**en hexadecimal:**
LOAD 10(R3), R1	00 001 011 00001010	0B0Ah
LOAD 0(R0), R1	00 001 000 00000000	0800h
STORE R0, 100(R3)	01 000 011 01100100	4364h
STORE R7, 255(R7)	01 111 111 11111111	7FFFh

Instruccions aritmètiques i lògiques

Les instruccions aritmètiques i lògiques (A-L) de la Màquina Rudimentària són les següents:

ADD *font1, font2, destí*

ADDI *font1, font2, destí*

Addition, suma de dos enters.

SUB *font1, font2, destí*

SUBI *font1, font2, destí*

Substraction, resta de dos enters.

ASR *font*, *destí*

Arithmetic shift right, desplaçament aritmètic cap a la dreta d'un bit; el bit de més pes es replica per tal de mantenir el signe. És equivalent a una divisió entera per 2.

AND *font1, font2, destí*

And lògica de dos operands.

Després d'executar-se qualsevol instrucció A-L, els bits de condició s'actualitzen segons el valor del resultat.

Totes les instruccions aritmètiques i lògiques obtenen els operands font del Banc de Registres i escriuen el resultat al Banc de Registres, excepte les instruccions **ADDI** i **SUBI**. En aquestes dues instruccions, el segon operand font es diu ***operand immediat*** i és un número enter entre -16 i 15. Per indicar un operand immediat s'escriu

#número

on *número* $\in$ [-16, 15].

A continuació es mostren alguns exemples d'execució d'instruccions A-L.

ADD R4, R3, R5

	ABANS	DESPRÉS
R3	1000 1110 1111 1011	1000 1110 1111 1011
R4	0011 1000 1010 1010	0011 1000 1010 1010
R5	xxxx xxxx xxxx xxxx	**1100 0111 1010 0101**
Bit Z	x	**0**
Bit N	x	**1**

SUBI R1, #13, R2

	ABANS	DESPRÉS
R1	0000 0000 0001 1000	0000 0000 0001 1000
R2	xxxx xxxx xxxx xxxx	**0000 0000 0000 1011**
Bit Z	x	**0**
Bit N	x	**0**

ASR R2, R5

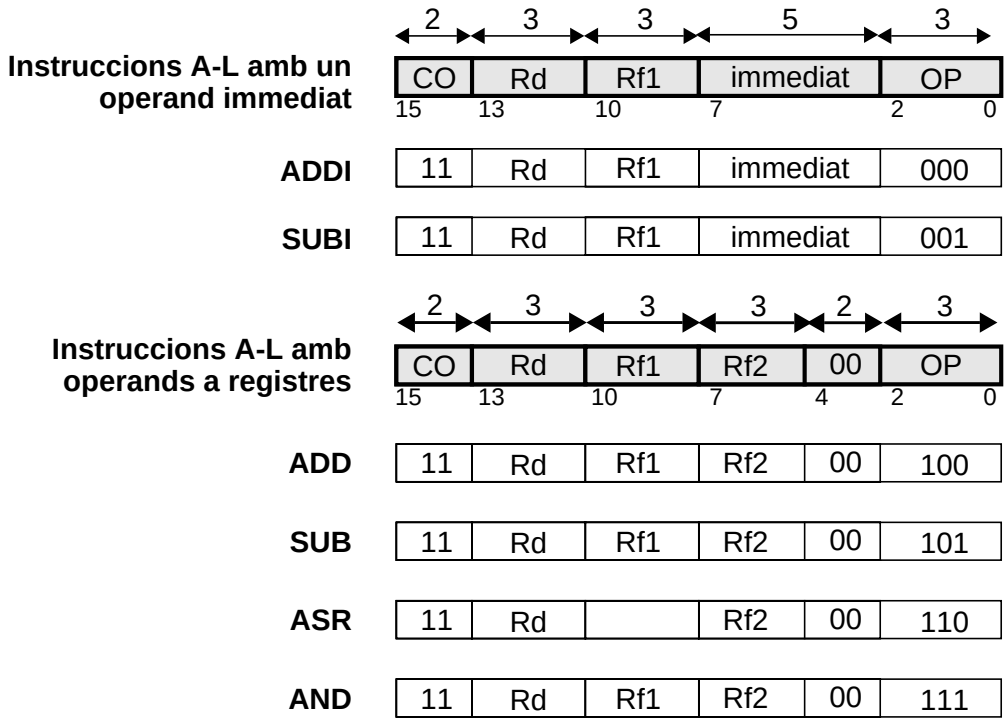
	ABANS	DESPRÉS
R2	1101 0011 0010 0001	1101 0011 0010 0001
R5	xxxx xxxx xxxx xxxx	**1110 1001 1001 0000**
Bit Z	x	**0**
Bit N	x	**1**

AND R1, R0, R1

	ABANS	DESPRÉS
R0	0000 0000 0000 0000	0000 0000 0000 0000
R1	0110 1111 0101 1011	**0000 0000 0000 0000**
Bit Z	x	**1**
Bit N	x	**0**

Codificació en LM

Les instruccions A-L es codifiquen en Llenguatge Màquina de la manera següent:



En totes elles el camp CO té el valor 11. Per distingir entre elles es fa servir el camp OP (*Operació*, bits 2:0); observem que hi ha dues combinacions d'aquests 3 bits que no es fan servir, donat que només hi ha 6 instruccions A-L.

Per codificar els operands immediats calen 5 bits (7:3), donat que el rang de valors possibles és [-16, 15] i es codifiquen en complement a 2. En les instruccions amb tots els operands en registres els bits 4:3 no es fan servir, i contenen el valor 00.

La instrucció ASR té un valor indefinit en els bits 10:8 (camp Rf1), donat que només té un operand font.

A continuació es mostren alguns exemples de codificació d'instruccions A-L en LM.

Llenguatge Assemblador	**Llenguatge Màquina**	**en hexadecimal:**
ADD R4, R0, R5	11 101 100 000 00 100	EC04h
SUBI R3, #1, R3	11 011 011 00001 001	DB09h
ASR R2, R2	11 010 000 010 00 110	D046h
AND R7, R7, R7	11 111 111 111 00 111	FFE7h

Instruccions de salt

Les instruccions de salt donen al processador la possibilitat de trencar el seqüenciament implícit. Per què es produeixi aquest trencament s'ha de complir la ***condició de salt***. Si no es compleix, es segueixen executant les instruccions en seqüència; si es compleix, es diu que el processador "salta" o "pren el salt". Les instruccions de salt són necessàries per implementar sentències d'alt nivell de tipus condicional i iteratiu.

Hi ha diverses instruccions de salt, que es distingeixen perquè la condició de salt que s'avalua és diferent en cadascuna d'elles. La sintaxi és

codi_d'operació adreça_destí_del_salt

on *codi_d'operació* és el nom de la instrucció i *adreça_destí_del_salt* indica l'adreça de la instrucció que s'executarà a continuació en cas que es prengui el salt; per tant, és un número entre 0 i 255.

La condició de salt s'avalua en funció del resultat de l'última instrucció aritmètica o de càrrega que ha executat el processador; la utilitat dels bits de condició és justament facilitar aquesta avaluació. En concret, les instruccions de salt de la Màquina Rudimentària són les següents:

Instrucció:	***Branch if...***	**Condició de salt: si l'últim resultat ha estat...**	**Què s'avalua**
BL	*...lower*	...menor que 0	N = 1
BG	*...greater*	...més gran que 0	N = 0 i Z = 0
BEQ	*...equal*	...igual a 0	Z = 1
BNE	*...not equal*	...diferent de 0	Z = 0
BLE	*...lower or equal*	...menor o igual que 0	N = 1 o Z = 1
BGE	*...greater or equal*	...més gran o igual que 0	N = 0
BR	*Branch*	Salt incondicional	1

En la instrucció de salt incondicional, BR, no s'avalua cap condició, sempre es pren el salt. En executar les instruccions de salt no es modifiquen el bits de condició.

A continuació es mostren alguns exemples d'execució d'instruccions de salt.

BR 25

	ABANS	**DESPRÉS**
PC	xxxx xxxx	**0001 1010**
Bit Z	x	x
Bit N	x	x

BGE 174

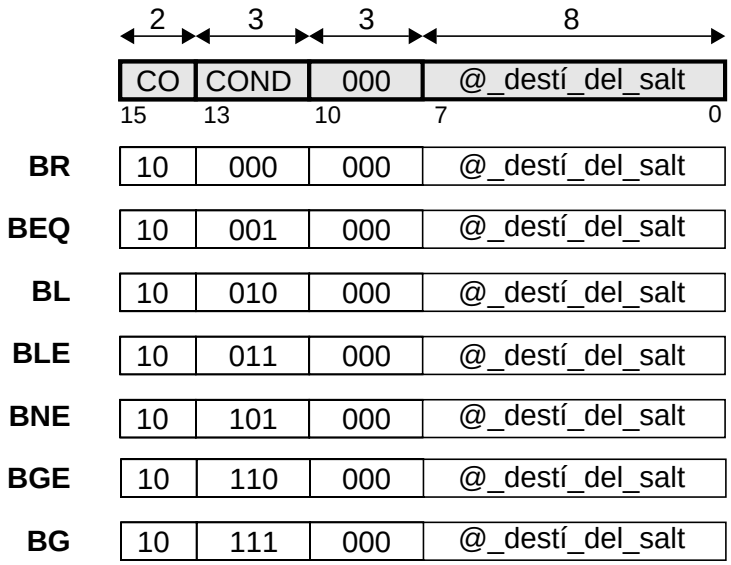
	ABANS	DESPRÉS
PC	0001 1000	**0001 1001**
Bit Z	0	0
Bit N	1	1

BGE 174

	ABANS	DESPRÉS
PC	0001 1000	**1010 1111**
Bit Z	1	1
Bit N	0	0

Codificació en LM

Les instruccions de salt es codifiquen en LM així:



En totes elles el camp CO té el valor 10. Per distingir entre elles es fa servir el camp COND (*Condició*, bits 13:11); observem que hi ha una combinació d'aquests 3 bits que no es fa servir, donat que només hi ha 7 instruccions de salt.

El camp dels bits 10:8 no s'usa, i té el valor 000 per totes les instruccions de salt. Els bits 7:0 codifiquen l'adreça destí del salt.

A continuació es mostren alguns exemples de codificació d'instruccions de salt.

Llenguatge Assemblador	**Llenguatge Màquina**	**en hexadecimal:**
BR 25	10 000 000 00011001	8019h
BL 0	10 010 000 00000000	9000h
BEQ 136	10 001 000 10001000	8888h

Exemple de programació en Llenguatge Màquina de la Màquina Rudimentària

A continuació es presenta la traducció d'un programa en alt nivell a Llenguatge Màquina de l'MR. El que fa el programa és la multiplicació de dues variables *a* i *b* mitjançant una suma acumulada d'*a* sobre la variable *mul b* vegades.

Programa en alt nivell:

```
var a,b,mul: enter;
a:= 10;
b:= 5;
mul:= 0;
mentre b>0 fer
      mul:= mul+a;
      b:= b-1;
fmentre
```

En la traducció a Llenguatge Assemblador s'ha suposat que:

- les variables *a*, *b* i *mul* es guarden a les adreces 00h, 01h i 02h respectivament
- les variables ja estan inicialitzades (és a dir, no cal traduir les 3 primeres sentències d'alt nivell).

Programa en baix nivell:

	Llenguatge Assemblador	@	M[@]: Llenguatge Màquina	en hexadecimal
		00h	0000 0000 0000 1010	000Ah
		01h	0000 0000 0000 0101	0005h
		02h	0000 0000 0000 0000	0000h
carregar dades	LOAD 0(R0), R1	03h	0000 1000 0000 0000	0800h
	LOAD 1(R0), R2	04h	0001 0000 0000 0001	1001h
	ADDI R0, #0, R3	05h	1101 1000 0000 0000	D800h
	SUBI R2, #0, R0	06h	1100 0010 0000 0001	C201h
càlculs	BLE 0Bh	07h	1001 1000 0000 1011	980Bh
	ADD R3, R1, R3	08h	1101 1011 0010 0100	DB24h
	SUBI R2, #1, R2	09h	1101 0010 0000 1001	D209h
	BR 06h	0Ah	1000 0000 0000 0110	8006h
guardar resultat	STORE R3, 2(R0)	0Bh	0101 1000 0000 0010	5802h

Durant l'execució del programa les variables *a*, *b* i *mul* es guarden a R1, R2 i R3 respectivament. Les instruccions del programa comencen a l'adreça 03h.

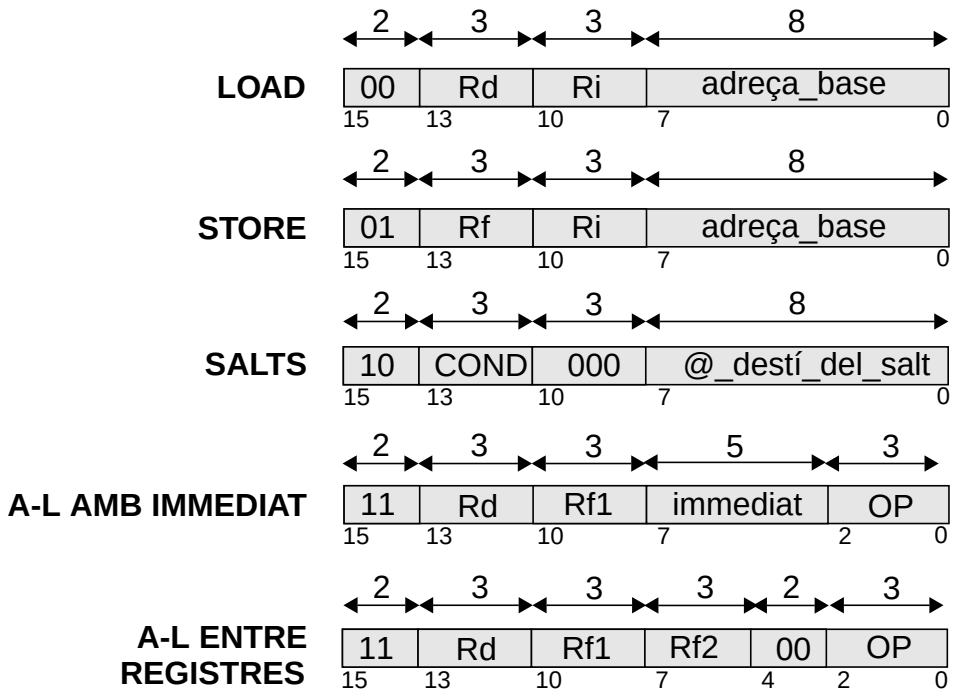
Es pot observar que, tal com és típic, el programa en baix nivell consta de tres parts. En la primera es porten les dades amb les que treballa el programa cap al processador, en la segona es fan els càlculs requerits pel programa, i en l'última els resultats es guarden a la memòria.

- La instrucció **ADDI R0, #0, R3** inicialitza R3 a 0. Això es pot fer de diverses maneres:

  AND R0, R1, R3
  LOAD 2(R0), R3
  etc.

- La instrucció **SUBI R2, #0, R0** és la traducció de l'avaluació de la condició del bucle (*b > 0*). Com que a R0 no s'hi pot escriure, l'únic efecte de la instrucció és que el bits de condició s'actualitzen d'acord amb el valor d'R2 (que guarda la variable *b*).

Resum del Llenguatge Màquina de l'MR



Instrucció de salt	COND
BR	000
BEQ	001
BL	010
BLE	011
BNE	101
BGE	110
BG	111

Instrucció A-L	OP
ADDI	000
SUBI	001
ADD	100
SUB	101
ASR	110
AND	111

Figura 2.1. Llenguatge Màquina de la Màquina Rudimentària.

3. Unitat de Procés de la Màquina Rudimentària

Tal com ja havíem vist, la Unitat de Procés és el conjunt de dispositius electrònics de la CPU que permeten dur a terme totes les tasques necessàries per a l'execució d'instruccions. Vegem quines són aquestes funcions que la UP ha de ser capaç de fer.

En totes les figures d'aquesta secció s'escriuen en cursiva els noms dels senyals generats per la Unitat de Control (que s'estudiaran a la secció següent).

Guardar la instrucció que s'està executant

Ja hem vist que les instruccions es porten d'una en una de la memòria al processador, on són executades. L'execució dura un cert nombre de cicles, durant els quals la instrucció es guarda al **Registre d'Instruccions** (**IR**). És un registre de 16 bits que té l'entrada de dades connectada amb la sortida de dades de la memòria (Mout), tal com es mostra a la Figura 3.1. El senyal de càrrega *Ld_IR* el governa la Unitat de Control, que també dóna ordre de llegir de la memòria posant a 0 el senyal $\overline{L/E}$.

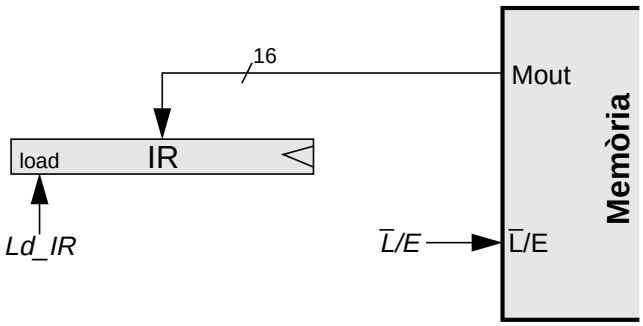


Figura 3.1. Circuiteria per carregar el registre IR.

Seqüenciament de les instruccions

El registre **Comptador de Programa** (**PC**) guarda en tot moment l'adreça següent a la de la instrucció que s'està executant; es diu que *apunta* a la següent instrucció. El PC es fa servir per adreçar la memòria després d'executar cada instrucció per obtenir-ne la instrucció que s'executarà a continuació, excepte quan s'ha de prendre un salt. És doncs un registre de 8 bits, amb la sortida de dades connectada a l'entrada d'adreces de la memòria (M@) a través del bus d'adreces.

Donat que segons el seqüenciament implícit s'executen consecutivament les instruccions d'adreces successives, el contingut del PC s'incrementa en 1 després de llegir de la memòria cada nova instrucció. El senyal de càrrega *Ld_PC* està governat per la UC.

El seqüenciament implícit pot trencar-se en executar instruccions de salt. En aquest cas,

l'adreça de la instrucció a executar l'especifica la pròpia instrucció de salt (que està guardada a l'IR), en els bits 7:0. Per tant, hi ha també una connexió entre els bits 7:0 de l'IR i el bus d'adreces de la memòria. El multiplexor **SELADR (Figura 3.2)**, governat per la UC (senyal $\overline{PC}/@$), permet discriminar amb quin dels dos valors s'ha d'adreçar la memòria en cada moment.

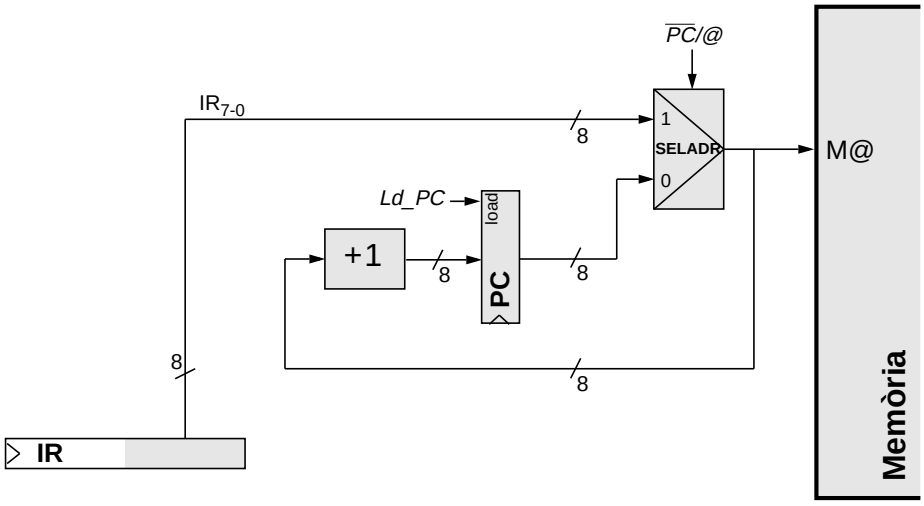


Figura 3.2. Circuiteria per al seqüenciament d'instruccions.

Guardar les dades dels programes

Les dades dels programes es porten de la memòria a la CPU per operar amb elles. Mentre són a la CPU es guarden al Banc de Registres, un conjunt de 8 registres de 16 bits amb els noms R0, R1, ..., R7. El registre R0 sempre conté un 0. Els registres del Banc també es fan servir per guardar resultats de càlculs intermitjos que s'hagin de fer per executar els programes.

Es pot llegir un registre i escriure un registre simultàniament. Per això el Banc de Registres té un bus d'entrada i un de sortida, ambdós de 16 bits, i les entrades de control següents:

SL: selecció de lectura, senyal de 3 bits que permet seleccionar quin dels 8 registres es vol llegir.

SE: selecció d'escriptura, el mateix però per escriure.

E: permís d'escriptura (1 bit).

La Figura 3.3 mostra la implementació interna del Banc de Registres.

A l'entrada SL del Banc de Registres s'hi ha de poder arribar des de:

- els bits $IR_{10:8}$ i $IR_{7:5}$ quan s'està executant una instrucció A-L. En aquest moment, aquests camps corresponen a Rf1 i Rf2 respectivament, és a dir, indiquen quins són els operands font.
- també els bits $IR_{10:8}$ quan s'executen instruccions d'accés a memòria. En aquest cas corresponen al camp Ri, és a dir, codifiquen el registre índex.
- els bits $IR_{13:11}$ en executar un STORE, per obtenir el registre font.

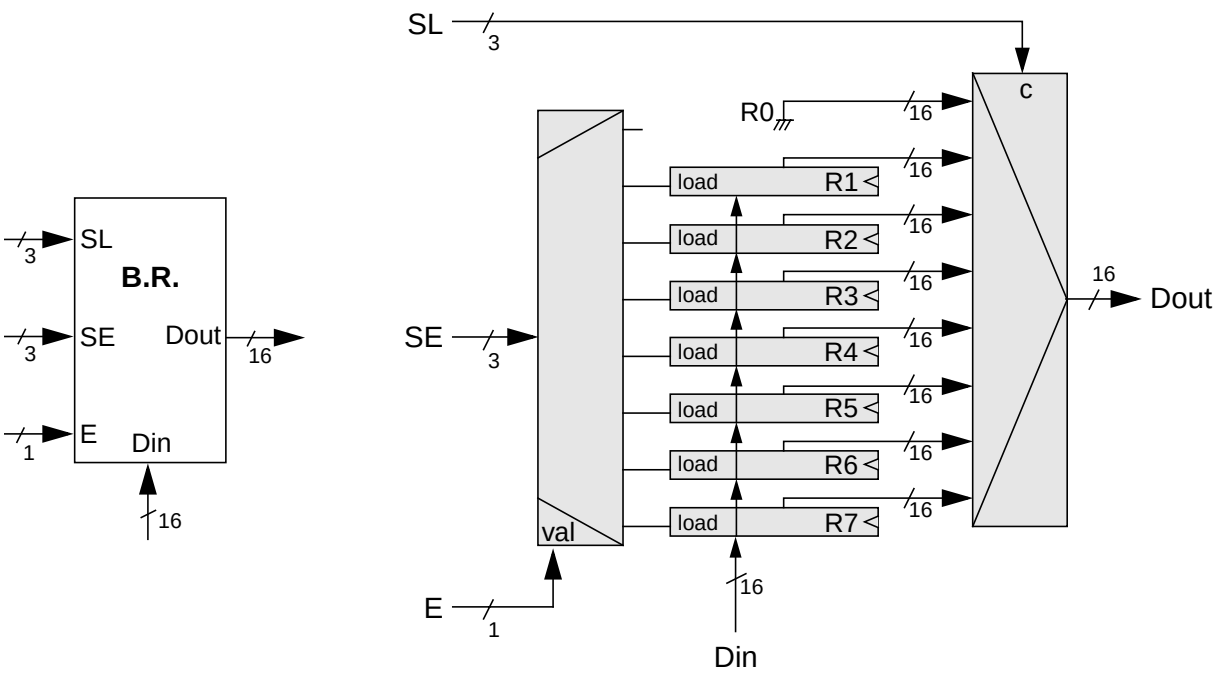


Figura 3.3. Implementació interna del Banc de Registres de l'MR.

El multiplexor **SELREG** possibilita la connexió d'aquests tres camins amb l'entrada SL del Banc (Figura 3.4). SELREG està governat per la UC a través del senyal CRf, de dos bits ja que és un multiplexor 4-1.

Les escriptures al Banc de Registres es fan en acabar l'execució d'una instrucció A-L o un LOAD. En tots dos casos el registre destí està codificat en els bits 13:11, i per això hi ha una connexió entre els bits $IR_{13:11}$ i l'entrada SE del Banc de Registres. La UC donarà ordre d'escriptura quan pertoqui mitjançant el senyal *ERd*.

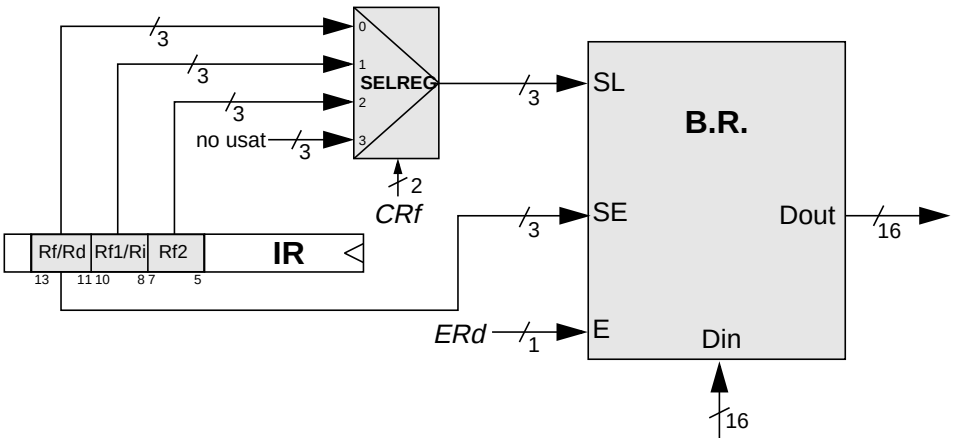


Figura 3.4. Entrades de control del Banc de Registres

Fer operacions amb les dades dels programes

Per a això hi ha la Unitat Aritmètica i Lògica (ALU), que rep dos operands (entrades A i B, de 16 bits) i genera un resultat (sortida O, de 16 bits) i dos senyals d'un bit (N i Z) que indiquen si el resultat ha estat negatiu o zero. L'ALU està governada per tres senyals de control: C2, C1 i C0.

A més de fer les 4 operacions de què disposa l'MR (suma, resta, desplaçament aritmètic a la dreta i *and*), l'ALU també és capaç de treure per la sortida el mateix valor que li arriba per l'entrada B (Figura 3.6.a). Això és degut a què la dada llegida de la memòria per una instrucció LOAD, abans de ser escrita al registre destí entra a l'ALU i en surt sense ser modificada, per tal que els bits N i Z s'actualitzin segons el seu valor.

La Figura 3.5 mostra la implementació interna de l'ALU. Es pot observar que l'operand que arriba al desplaçador és el que entra per l'entrada B.

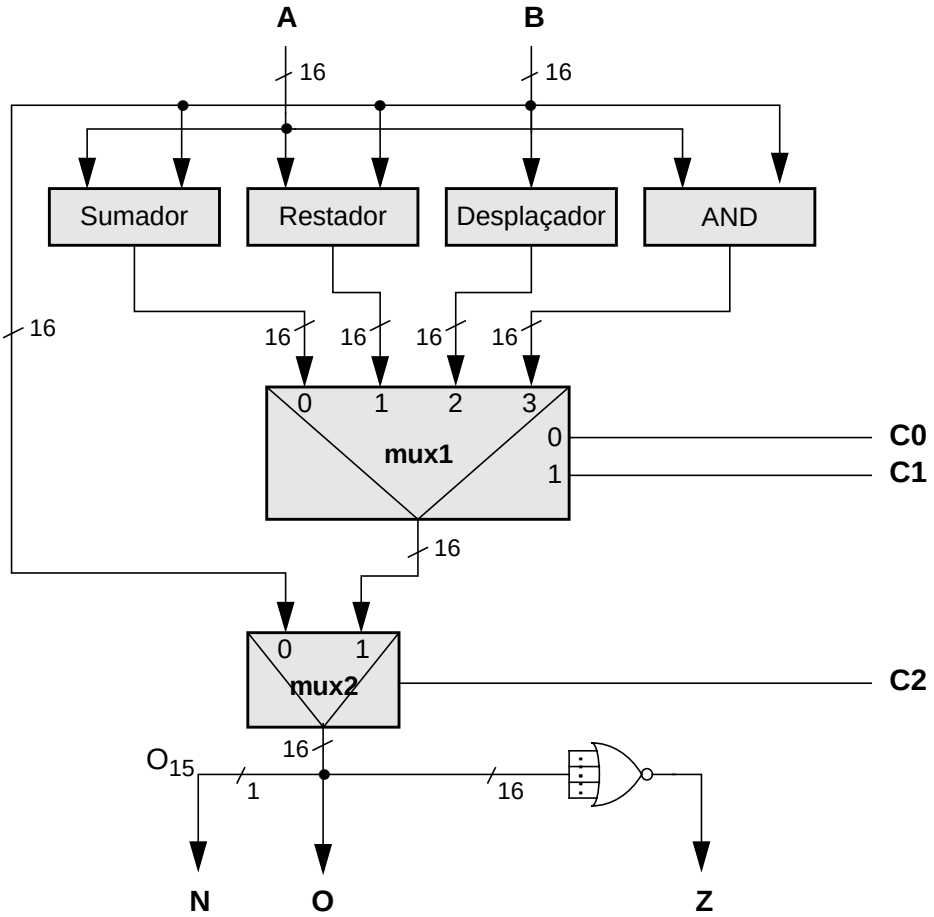


Figura 3.5. Implementació de l'ALU de la Màquina Rudimentària

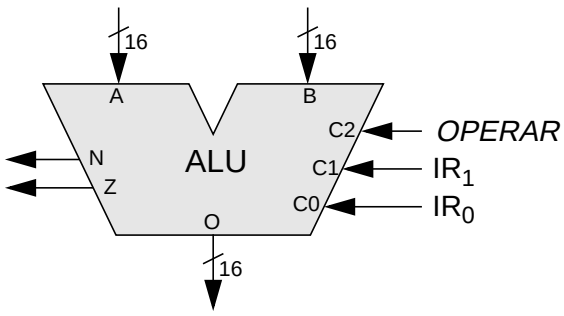
Els senyals C1 i C0 permeten triar entre una de les 4 operacions A-L. Els 4 mòduls operadors

estan connectats a les entrades del multiplexor **mux1** en un ordre tal que els senyals C1 i C0 es poden connectar amb els bits 1 i 0 de l'IR, que mentre s'executen instruccions A-L corresponen als 2 bits més baixos del camp OP (vegeu la Figura 2.1).

El multiplexor **mux2** permet triar entre el resultat d'una operació i el valor de l'entrada B; està governat pel senyal *OPERAR*, generat per la UC. La Figura 3.6 resumeix el paper dels senyals que controlen l'ALU.

C2	C1	C0	Sortida
0	x	x	B
1	0	0	A + B
1	0	1	A - B
1	1	0	B >> 1
1	1	1	A *and* B

(a)



(b)

Figura 3.6. (a) Funcions i (b) senyals de control de l'ALU.

Connexió de l'ALU amb la resta de blocs

Entrada A

Per aquesta entrada hi passen els primers operands de les instruccions aritmètico-lògiques. El Banc de Registres no permet llegir dos registres alhora, i per tant en les instruccions que requereixin dos operands del Banc aquests s'han de llegir en seqüència. Per això cal tenir un registre, el registre RA, que guardi el primer operand mentre es llegeix el segon i així puguin ser presents tots dos alhora a les entrades A i B de l'ALU (Figura 3.7). El senyal *Ld_RA* és generat per la UC.

Entrada B

Per aquesta entrada hi passen:

- els segons operands quan s'executen instruccions amb el segon operand en registre (això inclou l'únic operand de la instrucció ASR)
- els operands immediats quan s'executen instruccions amb el segon operand immediat. En aquest cas, l'operand és als bits $IR_{7:3}$, i se li ha d'extendre el signe fins a 16 bits donat que està codificat en complement a 2; és el que fa el mòdul EXT (Figura 3.7).
- la dada provinent de la memòria quan s'executen instruccions LOAD.

El multiplexor **SELDAT** possibilita la connexió d'aquests tres camins amb l'entrada B de l'ALU. L'entrada de selecció 1, que discrimina entre instrucció A-L o

LOAD, es connecta amb el senyal *OPERAR*, governat per la UC. L'entrada de selecció 0, que discrimina entre operand en registre o immediat, es connecta amb el bit 2 de l'IR (bit de més pes del camp OP, vegeu la Figura 2.1).

Sortida O

El resultat que genera l'ALU sempre s'ha d'escriure al Banc de Registres, i per això la sortida O es connecta amb l'entrada Din del Banc.

Sortides N i Z

Fan falta dos registres d'un bit, RZ i RN, per guardar els bits de condició que genera l'ALU a partir del resultat calculat. Les ordres de càrrega *Ld_RZ* i *Ld_RN* les dóna la UC.

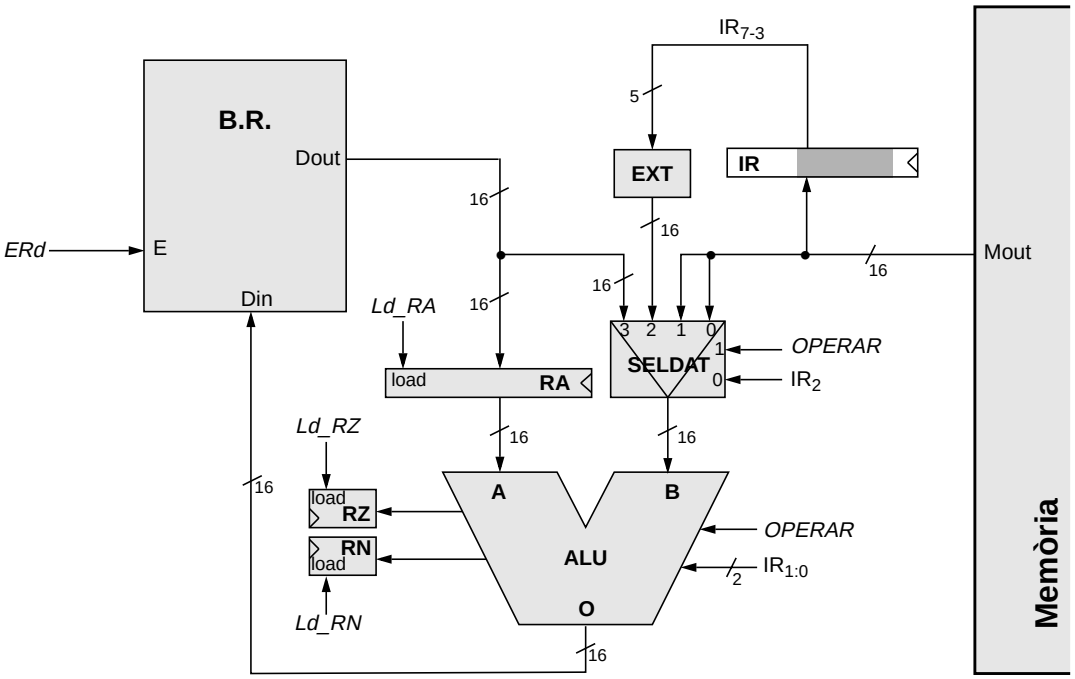


Figura 3.7. Connexió de l'ALU amb la resta de blocs

Avaluar la condició de salt

La decisió de si s'ha de saltar o no en executar una instrucció de salt s'ha de prendre en funció de quina instrucció sigui (bits 13:11 de l'IR, camp COND) i del valor guardat a RZ i RN, tal com es mostra a la taula de la pàgina 12. A la UP de la Màquina Rudimentària hi ha un bloc combinacional que en funció d'aquestes informacions genera un senyal d'un bit, ***Cond***, que val 1 si s'ha de saltar i 0 si no. La UC consulta el valor de Cond per decidir si dóna a la UP ordre de saltar o de seguir executant instruccions en seqüència.

Calcular l’adreça efectiva en les instruccions d’accés a memòria

En executar instruccions d’accés a memòria, l’adreça efectiva de l’operand que és a memòria (font o destí) s’obté sumant l’adreça base, que és als bits $IR_{7:0}$, més el desplaçament, que és als bits 7:0 del registre índex. Una vegada calculada, l’adreca efectiva es guarda al **registre R@**, que es connecta amb el bus d’adreces a través del multiplexor SELADR (Figura 3.8).

El registre R@ és necessari en l’execució d’un STORE, ja que l’adreça efectiva s’ha de mantenir present a l’entrada d’adreces de la memòria mentre del Banc de Registres es llegeix el registre font. Una connexió directa permet que aquest registre font arribi a l’entrada de dades de la memòria, Min. Per executar un LOAD no és estrictament necessari el registre R@, perquè en aquest cas només s’ha de llegir un registre del Banc, el registre índex.

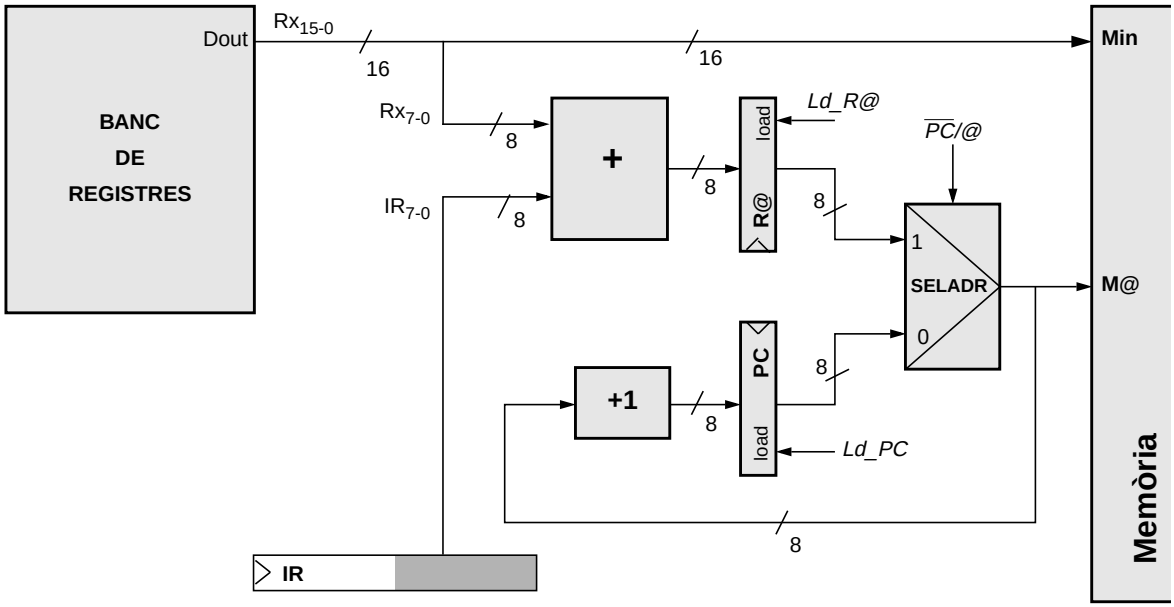
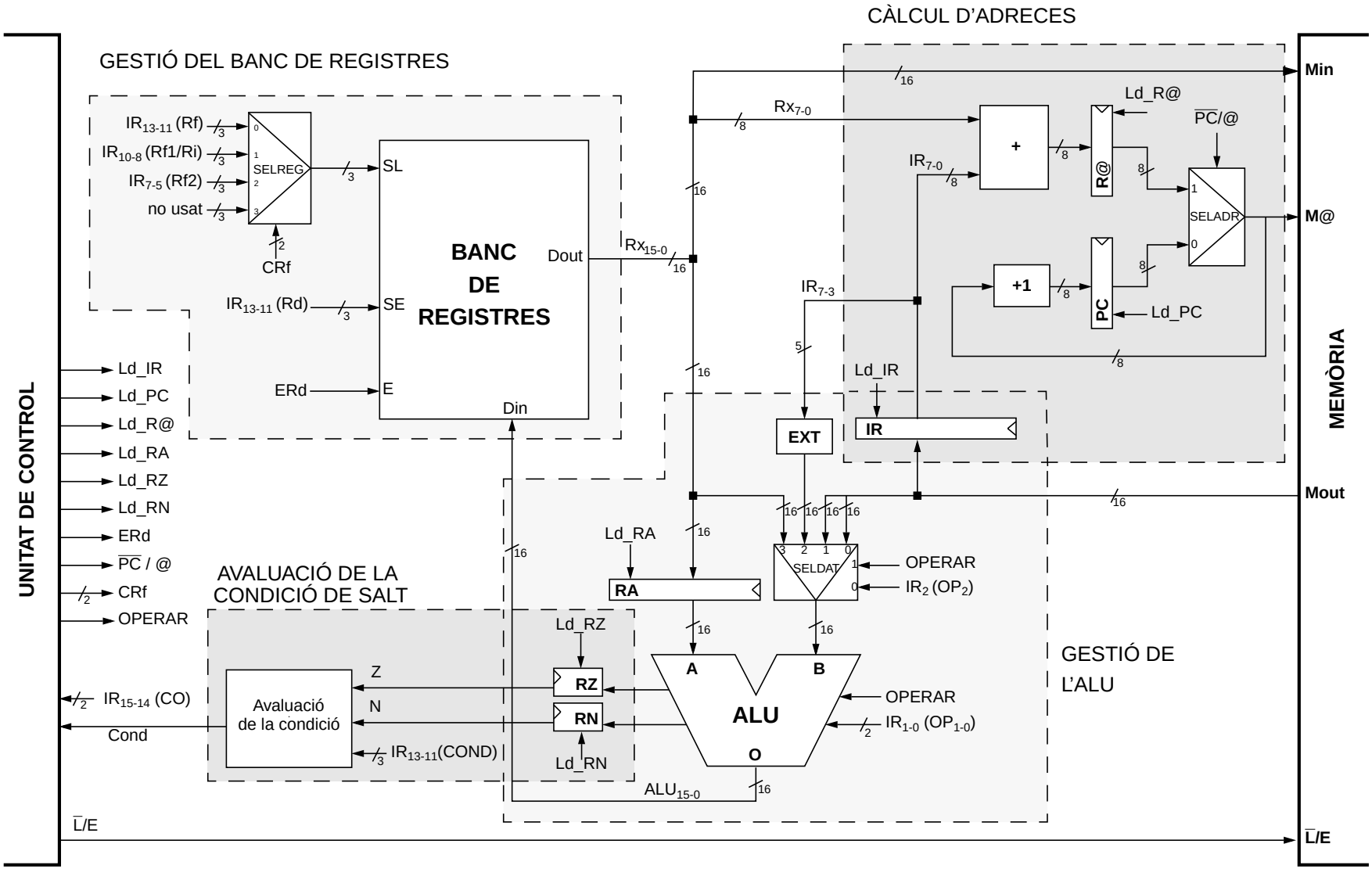


Figura 3.8. Circuiteria per a l’adreçament de la memòria.

Aquesta circuiteria modifica lleugerament l’esquema que s’havia proposat per executar instruccions de salt a la Figura 3.2, ja que ara els bits 7:0 de l’IR (adreça destí del salt) passen pel sumador i pel registre R@ abans d’arribar a SELADR. Per tal que l’adreça destí del salt no es modifiqui, el valor que se li suma ha de ser 0; això es pot aconseguir llegint el registre R0 en aquest moment. És per aquesta raó que les instruccions de salt tenen el valor 000 en els bits 10:8.

La figura següent mostra tota la circuiteria de la UP. Es pot veure que en total la Unitat de Control envia 11 senyals per governar la UP. Les informacions que fa servir la UC per donar valors a aquests 11 senyals són el tipus d’instrucció que s’està executant (bits $IR_{15:14}$, camp CO) i el senyal Cond (que indica si s’ha de prendre o no el salt en executar instruccions de salt).


UNITAT DE CONTROL
Ld_IR
Ld_PC
Ld_R@
Ld_RA
Ld_RZ
Ld_RN
ERd
PC / @
CRf
OPERAR
IR15-14 (CO)
Cond
L/E
GESTIÓ DEL BANC DE REGISTRES
IR13-11 (Rf)
IR10-8 (Rf1/Ri)
IR7-5 (Rf2)
no usat
SELREG
CRf
IR13-11 (Rd)
ERd
SL
SE
E
BANC DE REGISTRES
Dout
Din
Rx15-0
AVALUACIÓ DE LA CONDICIÓ DE SALT
Avaluació de la condició
Z
N
IR13-11(COND)
CÀLCUL D'ADRECES
Rx7-0
IR7-0
+
R@
Ld_R@
PC/@
SELADR
M@
IR7-3
+1
PC
Ld_PC
Ld_IR
EXT
IR
Mout
Ld_RA
RA
SELDAT
OPERAR
IR2 (OP2)
Ld_RZ
RZ
RN
Ld_RN
A
B
ALU
O
OPERAR
IR1-0 (OP1-0)
ALU15-0
GESTIÓ DE L'ALU
Min
MEMÒRIA
L/E

4. Unitat de Control de la Màquina Rudimentària

Fases en l'execució d'una instrucció

Per a executar una instrucció, el processador ha de realitzar seqüencialment una sèrie d'accions; es diu que ha de passar per un cert nombre de ***fases***, cadascuna de les quals pot durar un o més cicles de rellotge. Les fases d'execució varien entre instruccions. En les instruccions A-L són aquestes:

- **Fase de *fetch***: es porta la instrucció que s'ha d'executar des de la memòria fins al registre IR. La instrucció es llegeix de l'adreça apuntada pel PC, que es carrega amb el valor que tenia incrementat en 1.
- **Fase de decodificació**: la Unitat de Control analitza el codi d'operació de la nova instrucció i decideix quines ordres haurà de donar a la UP perquè l'executi.
- **Fase de lectura d'operands**: els operands requerits per la instrucció es fan arribar fins a l'ALU.
- **Fase d'execució:** l'ALU realitza l'operació indicada per la instrucció**.**
- **Fase d'escriptura de resultats**: s'escriu el resultat a la ubicació de destí i s'actualitzen els bits de condició, si la instrucció ho requereix.

Les dues primeres fases són comunes a totes les instruccions, i a partir d'aquí les accions concretes varien. En les instruccions d'accés a memòria, abans de la lectura dels operands cal calcular l'adreça efectiva de l'operand que sigui a memòria; no hi ha fase d'execució, perquè l'operand font s'escriu sense modificar sobre l'operand destí. En les instruccions de salt, les tres últimes fases es substitueixen per:

- **Fase d'avaluació de la condició**: es discerneix si s'ha de saltar o no en funció del valor dels bits de condició i de la instrucció que s'està executant.
- **Fase d'execució:** en cas que s'hagi de saltar es fa el fetch de la instrucció destí del salt i s'actualitza el PC convenientment.

La Unitat de Control és un sistema seqüencial amb 3 entrades i 11 sortides, que canvia d'estat a cada cicle de rellotge. Les ordres generades per la UC són entrades de selecció de multiplexors ($\overline{PC}$/@, CRf i OPERAR) i permisos d'escriptura (senyals de càrrega, senyal ERd i senyal $\overline{L}$/E). A cada cicle es fa servir un cert subconjunt de la circuiteria de la UP, i la part que no intervé està inactiva: els permisos d'escriptura es posen a 0 i les entrades de selecció dels multiplexors tenen un valor indeterminat.

En la representació gràfica de l'autòmat de la UC no escriurem les 11 sortides dintre dels estats, per raons de legibilitat. Dintre dels estats hi escriurem un mnemotècnic identificador de les sortides, i els valors dels 11 senyals els posarem en una taula a part. En aquesta taula escrivim en negreta els valors dels senyals que intervenen en les tasques de l'estat corresponent.

Fases comunes a totes les instruccions

FETCH

En aquesta fase s'han de fer les accions següents, preservant l'ordre:

$$IR := M[PC]$$

$$PC := PC + 1$$

Per a això cal fer arribar el contingut del PC fins al bus d'adreces ($\overline{PC}$/@ = 0), donar ordre de llegir la memòria ($\overline{L}$/E = 0), i activar els senyals de càrrega de l'IR i el PC (el que arriba a l'entrada de dades del PC és en tot moment el que passa pel bus d'adreces més 1). La Figura 4.1 mostra en traç gruixut i en negreta les parts de la UP involucrades en la fase de fetch.

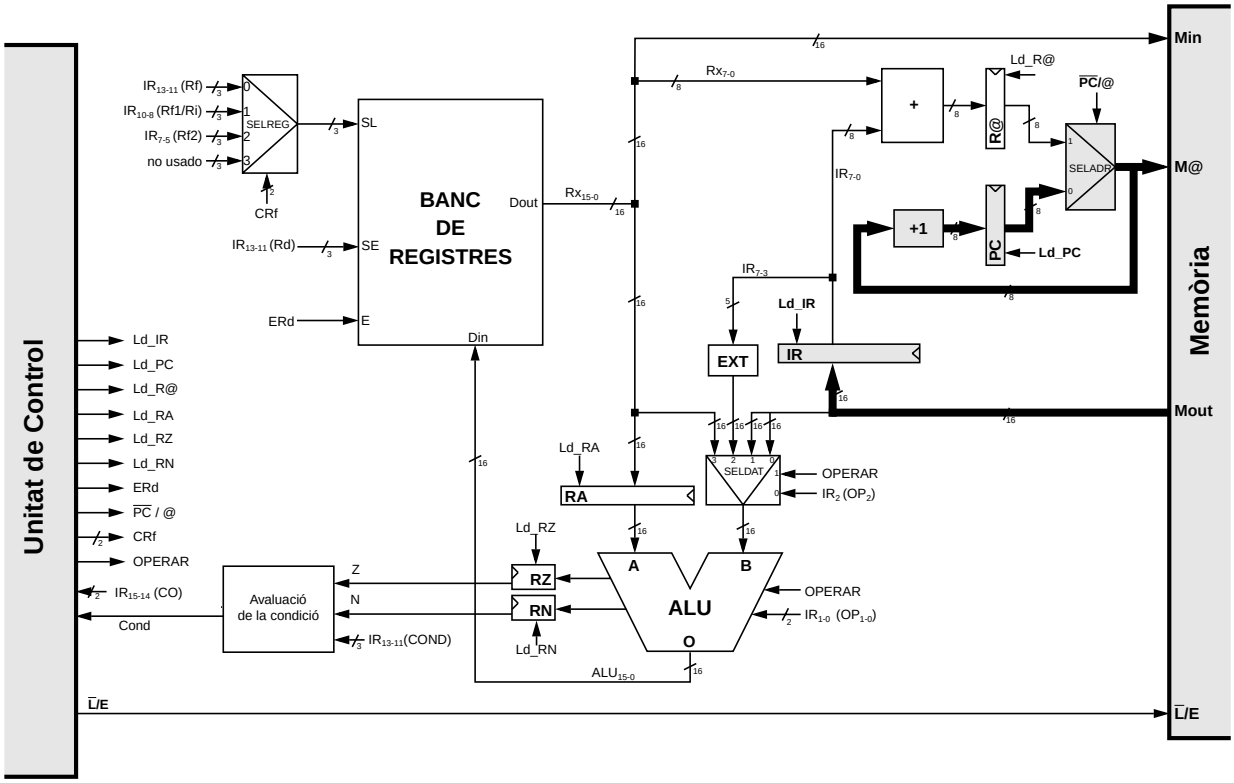


Figura 4.1. Activitat a la Unitat de Procés durant l'estat FETCH.

Totes les accions que s'han de dur a terme durant la fase de fetch es poden fer en un cicle. Els registres PC i IR no es carregaran fins al final del cicle, coincidint amb el flanc ascendent del rellotge, i per tant el valor del PC que arriba al bus d'adreces es manté constant al llarg de tot l'estat de *fetch* (Figura 4.2).

Després de la fase de fetch es passarà sempre a la fase de decodificació, independentment de quina instrucció s'estigui executant (Figura 4.3).

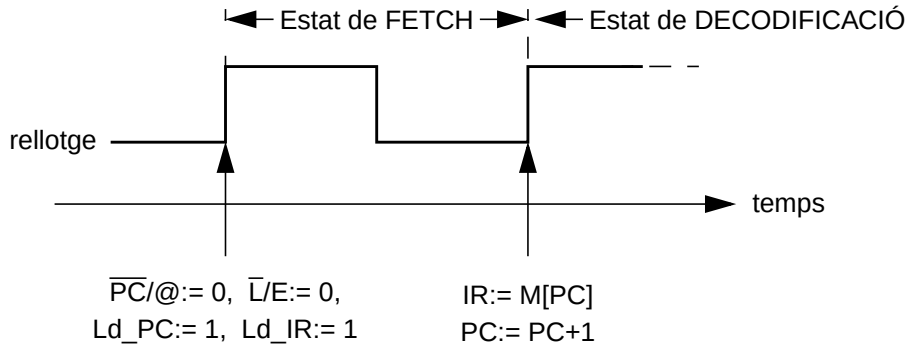
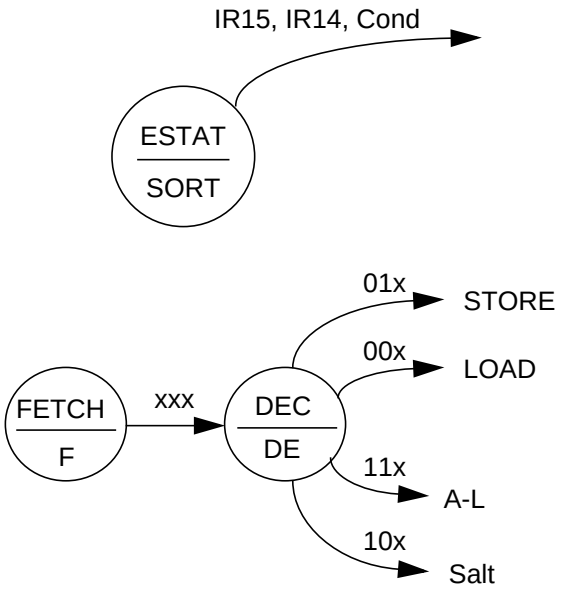


Figura 4.2. Temporització dels events en l'estat de *fetch.*

DECODIFICACIÓ (estat ***DEC***)

Durant aquesta fase, que dura un cicle, la UC avalua els senyals d'entrada $IR_{15:14}$, que han pres el nou valor en l'instant inicial del cicle, en carregar-se el registre IR. En acabar la fase de decodificació, la UC decideix quin serà l'estat futur en funció d'$IR_{15:14}$ (Figura 4.3).

Durant aquesta fase la UP no fa res, i per tant totes les ordres de control estan inactives.



Sortides	F	DE
Ld_IR	**1**	0
Ld_PC	**1**	0
Ld_R@	0	0
Ld_RA	0	0
Ld_RZ	0	0
Ld_RN	0	0
ERd	0	0
L/E	**0**	0
PC/@	**0**	x
CRf	xx	xx
OPERAR	x	x

Figura 4.3. Fases comunes a totes les instruccions

Instruccions d’accés a memòria

Per completar l’execució d’aquestes instruccions calen dos cicles, un per calcular l’adreça efectiva de l’operand que és a memòria, i l’altre per fer la transferència d’informació, en un sentit o en l’altre segons si s’executa un LOAD o un STORE.

CÀLCUL DE L’ADREÇA EFECTIVA (estat ***ADR1***)

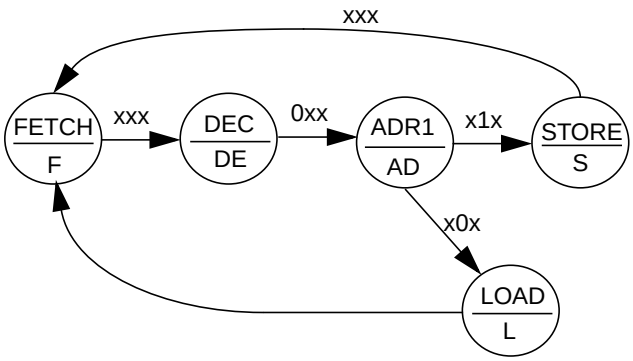
Es llegeix el registre índex del Banc de Registres (CRf = 01), es calcula l’adreça efectiva (sortida del sumador que suma els 8 bits més baixos del registre índex més els 8 bits més baixos de l’IR) i s’escriu al registre R@ al final del cicle (Ld_R@ = 1). Aquesta adreça efectiva es podrà fer servir per accedir a memòria al cicle següent.

LOAD

En aquest estat es fan les accions corresponents a les fases de lectura d’operands i escriptura de resultats. S’adreça la memòria amb el contingut d’R@ ($\overline{PC}$/R@ = 1), es llegeix la memòria ($\overline{L}$/E = 0) i s’escriu el que surt de l’ALU al registre destí (ERd = 1). El que surt de l’ALU és el que ha arribat de la memòria gràcies a què la UC posa OPERAR a 0 (multiplexors SELDAT i mux2). S’actualitzen també els bits de condició (Ld_RN = 1 i Ld_RZ = 1).

STORE

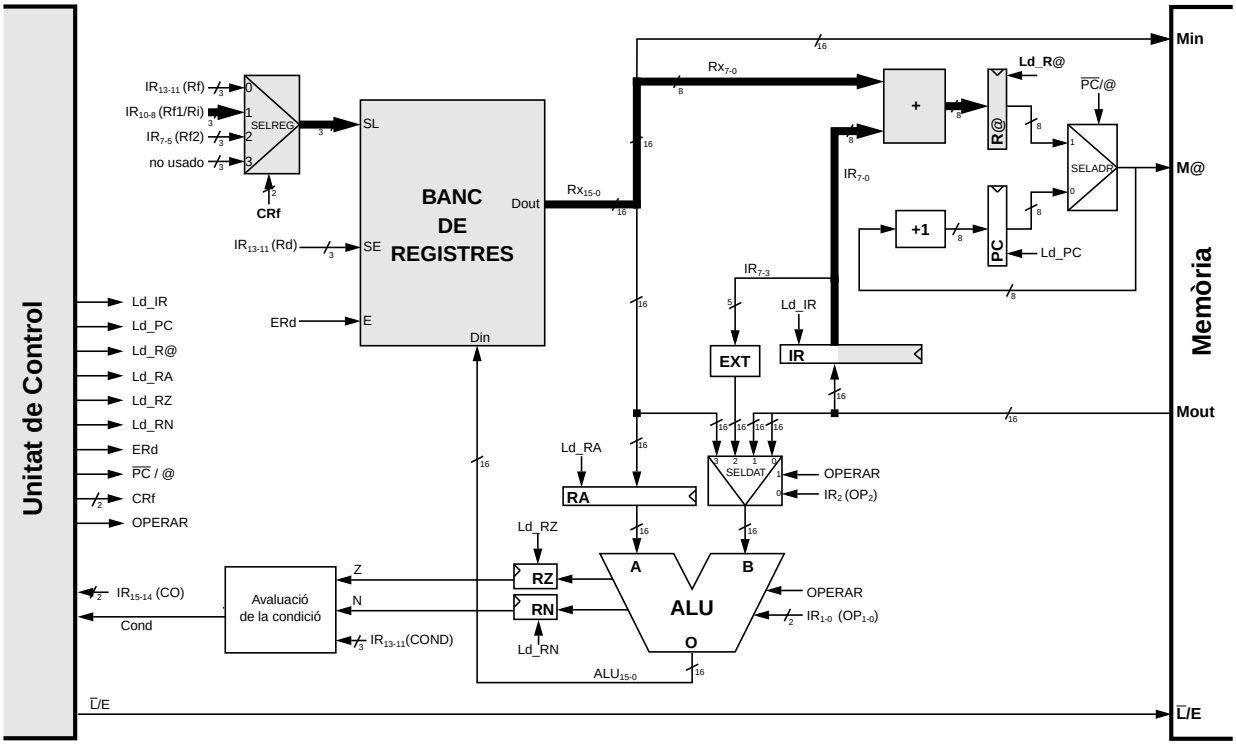
Es llegeix el registre font del Banc de Registres (CRf = 0) i el seu valor, que arriba fins a l’entrada de dades de la memòria, s’escriu ($\overline{L}$/E = 1) a l’adreça indicada pel registre R@ ($\overline{PC}$/R@ = 1). Són les accions corresponents a les fases de lectura d’operands i escriptura de resultats.



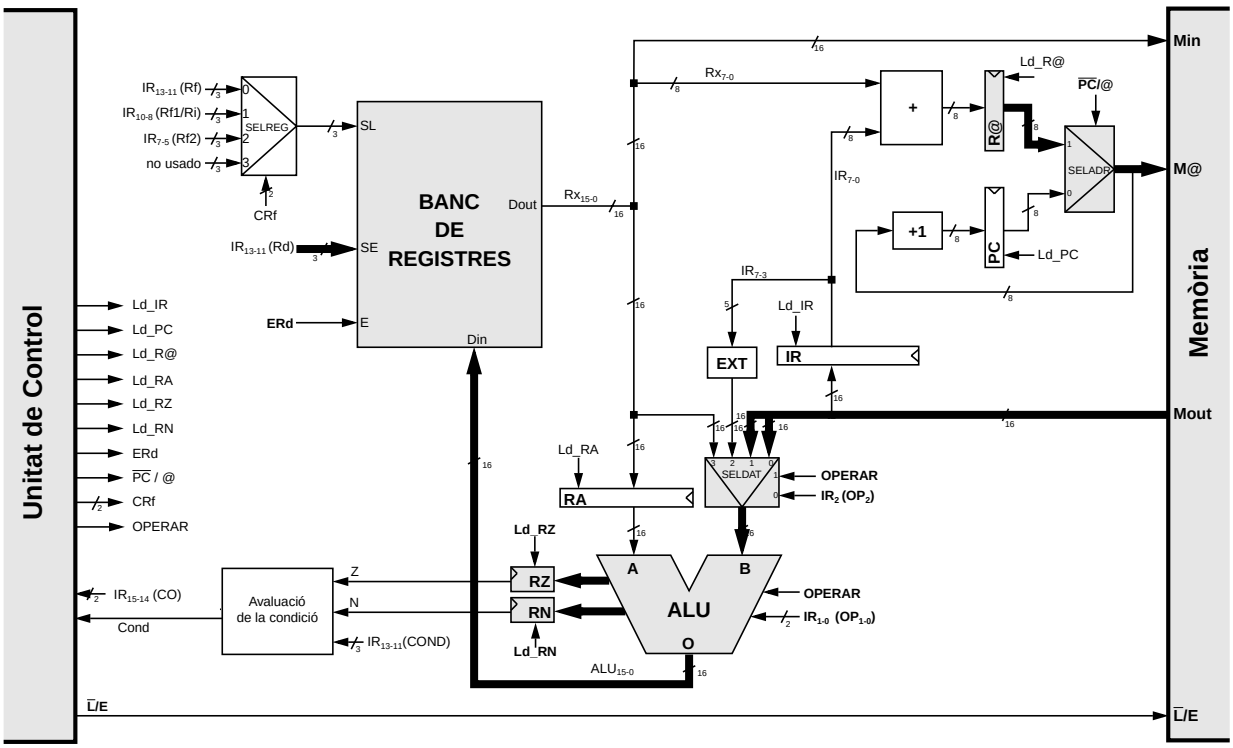
Sortides	AD	L	S
Ld_IR	0	0	0
Ld_PC	0	0	0
Ld_R@	**1**	0	0
Ld_RA	0	0	0
Ld_RZ	0	**1**	0
Ld_RN	0	**1**	0
ERd	0	**1**	0
$\overline{L}$/E	0	**0**	**1**
$\overline{PC}$/@	x	**1**	**1**
CRf	**01**	xx	**00**
OPERAR	x	**0**	x

Figura 4.4. Execució de les instruccions d’accés a memòria

La Figura 4.5 mostra l’activitat a la UP durant els cicles ADR1 i LOAD.



(a)



(b)

Figura 4.5. Activitat a la UP durant els estats ADR1 (a) i LOAD (b).

Instruccions aritmètiques i lògiques

Per a completar l'execució d'aquestes instruccions calen 2 cicles després de la decodificació:

LECTURA DEL PRIMER OPERAND (estat ***LPO***)

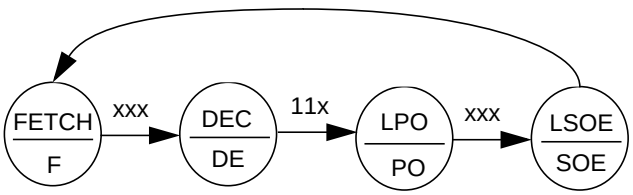
El primer operand es llegeix del Banc de Registres (CRf = 01, arriba el camp Rf1 de l'IR a l'entrada SL del Banc) i s'escriu al registre RA (Ld_RA = 1). Si la instrucció que s'executa és una ASR, el valor escrit al registre RA no es farà servir per res.

LECTURA DEL SEGON OPERAND I EXECUCIÓ (estat ***LSOE***)

Es llegeix del Banc el registre indicat pel camp Rf2 de la instrucció (CRf = 10), que arriba a l'entrada 3 de SELDAT. El bit IR_2 decideix si l'operand que arriba fins a l'ALU és el que ve del registre o l'immediat guardat a l'IR una vegada se li ha extès el signe (el senyal OPERAR val 1 mentre s'executen instruccions A-L).

Durant aquest cicle, els 4 mòduls de càlcul de l'ALU computen un resultat. Els bits $IR_{1:0}$ determinen quin dels 4 valors serà el que surti de l'ALU, ja que el senyal OPERAR estarà a 1. L'ALU computa també els valors dels bits N i Z.

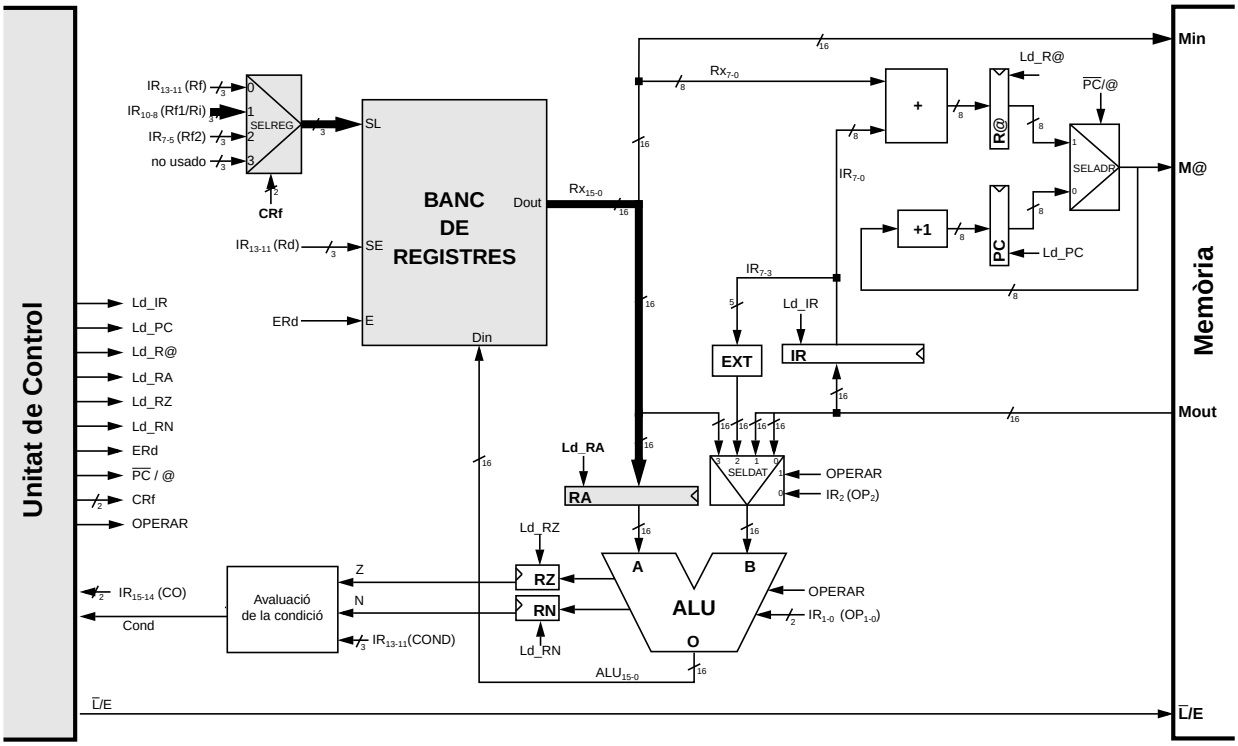
Tant el resultat de l'operació com els bits de condició tindran el valor correcte al final del cicle, i per tant es poden escriure ja a la seva ubicació de destí, ajuntant d'aquesta manera en un mateix cicle les fases d'execució i escriptura de resultats. Durant l'estat LSOE, doncs, la UC dóna ordre d'escriure sobre el Banc de Registres (ERd = 1) i de carregar els bits de condició als biestables (Ld_RZ = 1 i Ld_RN = 1).



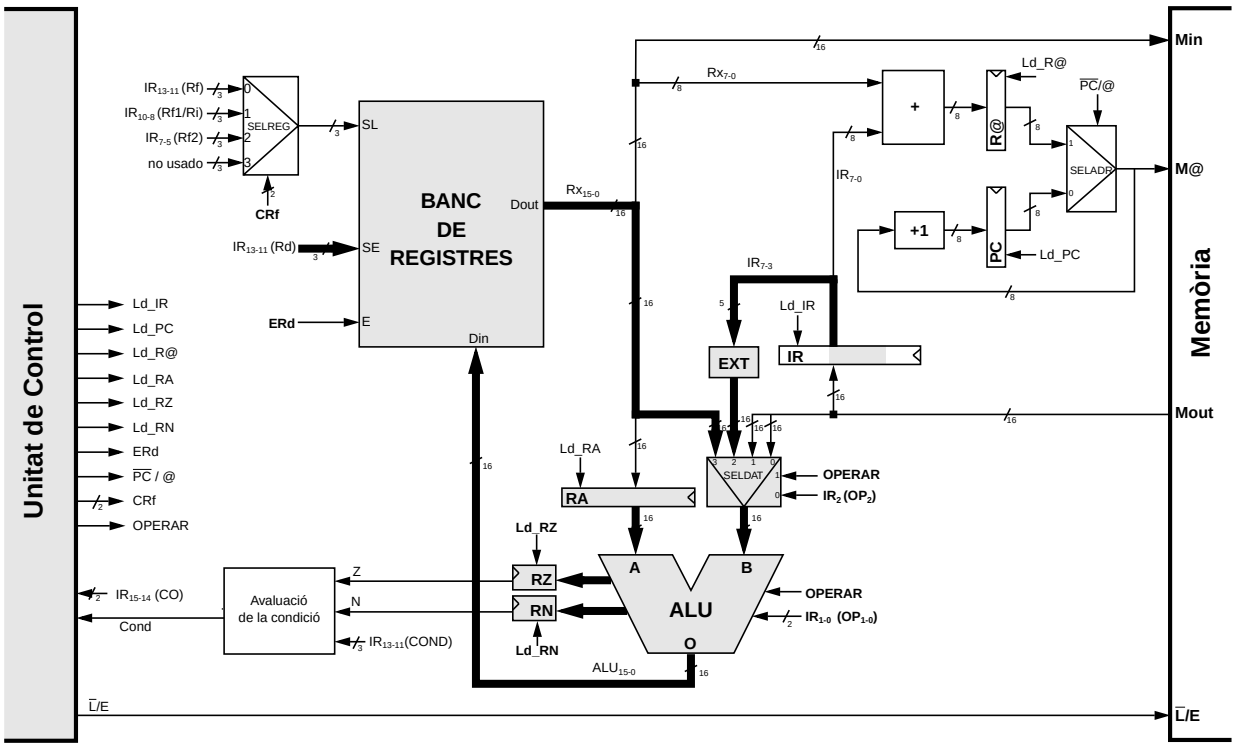
Sortides	PO	SOE
Ld_IR	0	0
Ld_PC	0	0
Ld_R@	0	0
Ld_RA	**1**	0
Ld_RZ	0	**1**
Ld_RN	0	**1**
ERd	0	**1**
$\overline{L}$/E	0	0
$\overline{PC}$/@	x	x
CRf	**01**	**10**
OPERAR	x	**1**

Figura 4.6. Execució de les instruccions aritmètiques i lògiques.

La Figura 4.7 mostra l'activitat a la UP durant els estats LPO i LSOE.



(a)



(b)

Figura 4.7. Activitat a la UP durant els estats LPO (a) i LSOE (b).

Instruccions de salt

Calen tres cicles després de la decodificació per completar l'execució de les instruccions de salt.

AVALUACIÓ DE LA CONDICIÓ DE SALT (estat ***ACS***)

No es fa res, simplement la UC examina l'entrada Cond per decidir si s'ha de saltar o no. En cas que no s'hagi de saltar l'estat futur és el de FETCH, per seguir l'execució del programa en seqüència (Figura 4.8).

Observem que les sortides en aquest estat tenen els mateixos valors que en l'estat de decodificació (la UP està inactiva).

CÀLCUL DE L'ADREÇA DESTÍ DEL SALT (estat ***ADR2***)

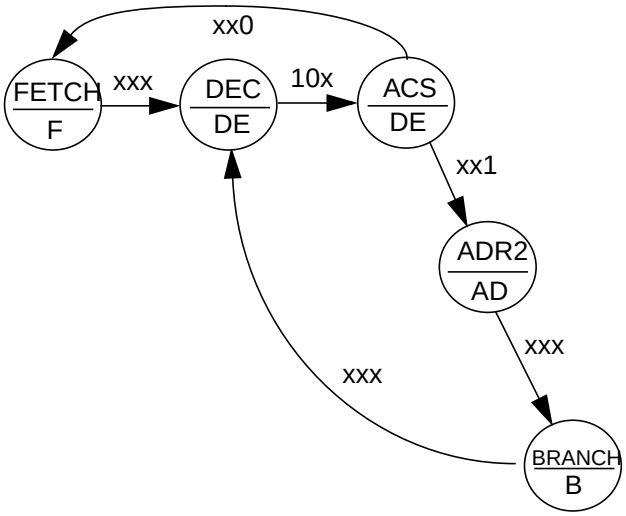
En cas que s'hagi de saltar, s'escriu al registre R@ (Ld_R@ = 1) l'adreça destí del salt, que és als bits 7:0 de l'IR. Perquè aquesta adreça no es modifiqui en passar pel sumador, es llegeix el registre R0 del Banc (CRf = 01, els bits $IR_{10:8}$ tenen el valor 000 en les instruccions de salt).

Les sortides en aquest estat tenen els mateixos valors que en l'estat ADR1.

SALT (estat ***BRANCH***)

Es fa el *fetch* de la instrucció destí del salt: al bus d'adreces hi arriba el contingut d'R@ ($\overline{PC}$/R@ = 1), es llegeix la memòria ($\overline{L}$/E = 0) i s'escriu el que arriba per Mout al registre IR (Ld_IR = 1). A més, el PC es carrega (Ld_PC = 1) amb l'adreça destí del salt més 1.

Per tant, l'estat futur d'aquest cicle és el de decodificació, donat que el *fetch* de la instrucció que s'executarà a continuació ja està fet (Figura 4.8).



Sortides	DE	AD	B
Ld_IR	0	0	**1**
Ld_PC	0	0	**1**
Ld_R@	0	**1**	0
Ld_RA	0	0	0
Ld_RZ	0	0	0
Ld_RN	0	0	0
ERd	0	0	0
$\overline{L}$/E	0	0	**0**
$\overline{PC}$/@	x	x	**1**
CRf	xx	**01**	xx
OPERAR	x	x	x

Figura 4.8. Execució de les instruccions de salt.

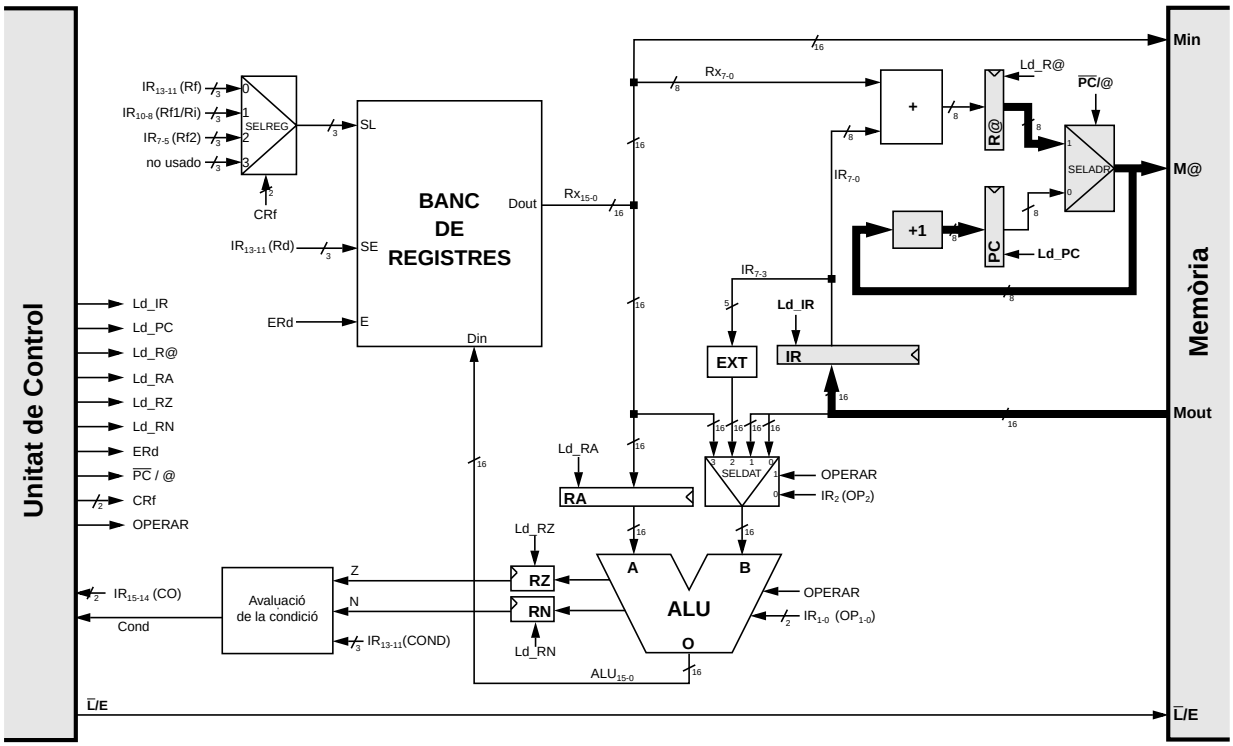
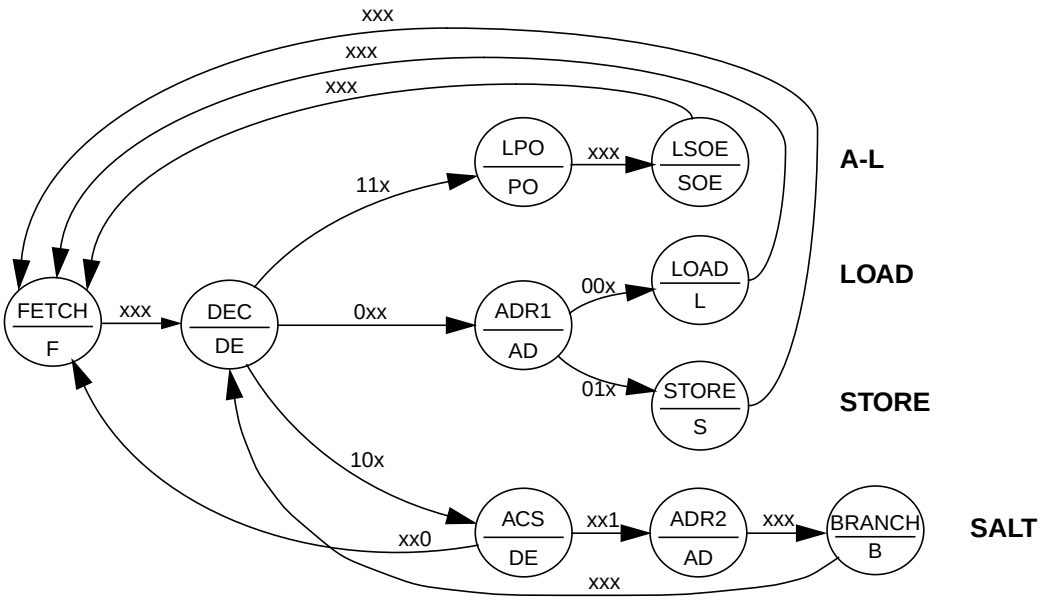


Figura 4.9. Activitat de la UP durant l'estat BRANCH.

Graf d’estats complet de la Unitat de Control



Sortides	F	DE	AD	L	S	PO	SOE	B
Ld_IR	1	0	0	0	0	0	0	1
Ld_PC	1	0	0	0	0	0	0	1
Ld_R@	0	0	1	0	0	0	0	0
Ld_RA	0	0	0	0	0	1	0	0
Ld_RZ	0	0	0	1	0	0	1	0
Ld_RN	0	0	0	1	0	0	1	0
ERd	0	0	0	1	0	0	1	0
L/E	0	0	0	0	1	0	0	0
PC/@	0	x	x	1	1	x	x	1
CRf	x	x	01	xx	00	01	10	xx
OPERAR	x	x	x	0	x	x	1	x

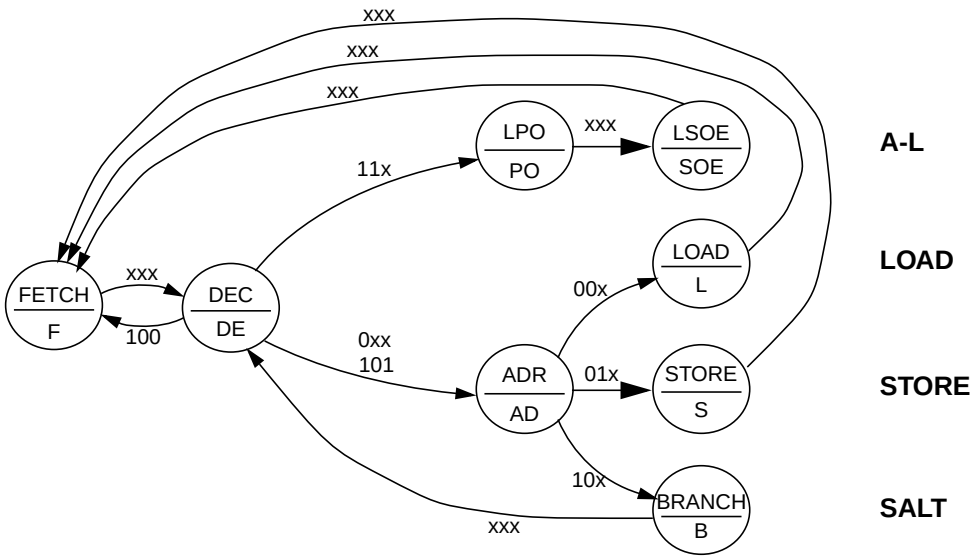
Optimitzacions de la Unitat de Control

Les instruccions de la Màquina Rudimentària es poden executar en menys cicles que els que es dibuixen en el graf anterior.

- El cicle dedicat a avaluar la condició de salt (estat ACS) es pot estalviar, ja que el valor del senyal Cond és conegut per la UC des del principi de l’estat de decodificació, i per tant després d’aquest estat ja es pot triar entre diferents estats

futurs en funció de Cond. Es poden unificar doncs els estats DEC i ACS (que tenen les mateixes sortides).

- Durant els estats ADR1 i ADR2 es donen exactament les mateixes ordres a la UP. Es poden unificar en un sol estat, **ADR**, i discriminar entre instruccions d'accés a memòria i instruccions de salt posteriorment. El graf queda així:



- Les sortides dels estats LPO i ADR es diferencien només en què el primer dóna ordre de carregar el registre RA i el segon dóna ordre de carregar el registre R@. Es poden unificar els dos estats, i donar totes dues ordres de càrrega. Si finalment resulta que la instrucció en execució és A-L, el valor carregat al registre R@ no es farà servir per res, però tampoc no afectarà al correcte funcionament del processador. Igualment, si la instrucció en execució és d'accés a memòria o de salt, el valor escrit al registre RA no tindrà cap efecte sobre l'execució.

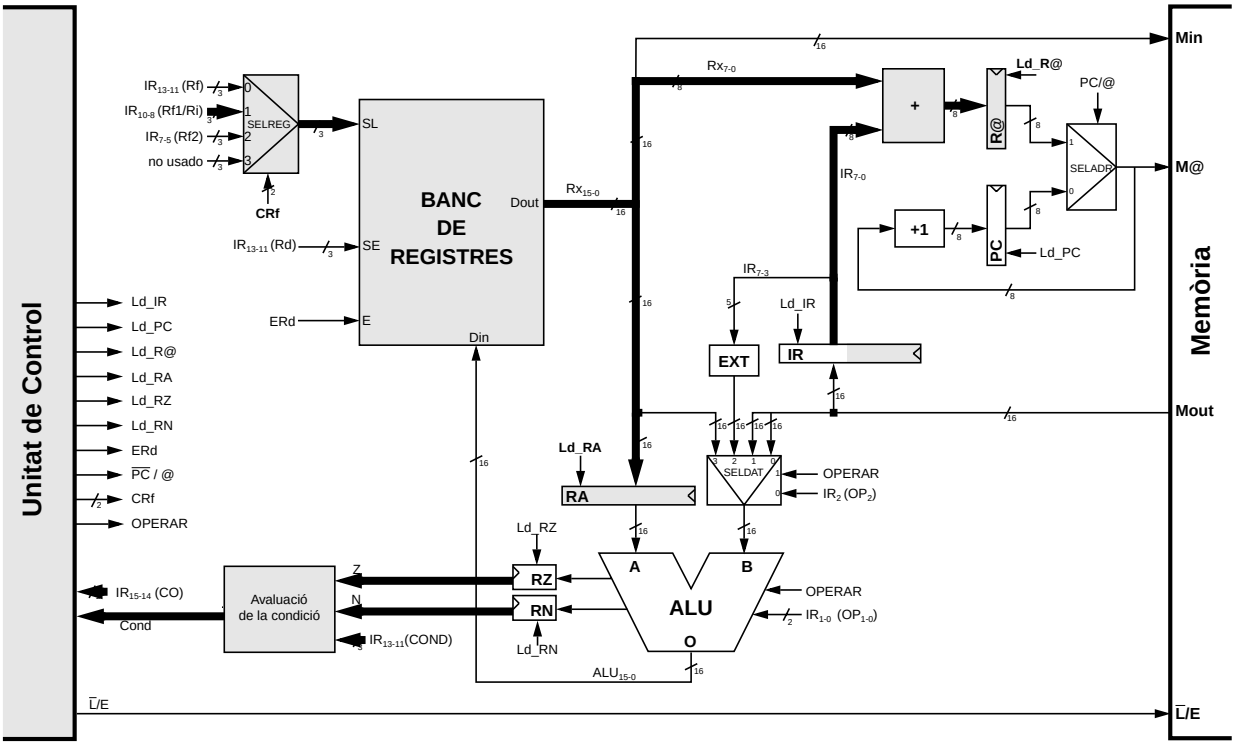
  A més, aquestes tasques es poden fer durant l'estat de decodificació, encara que no se sàpiga encara quina instrucció s'està executant, ja que en aquest estat la UP no estava fent cap feina.

  A l'estat de decodificació modificat li diem *DECO*. La Figura 4.10.a mostra l'activitat a la UP durant aquest estat.

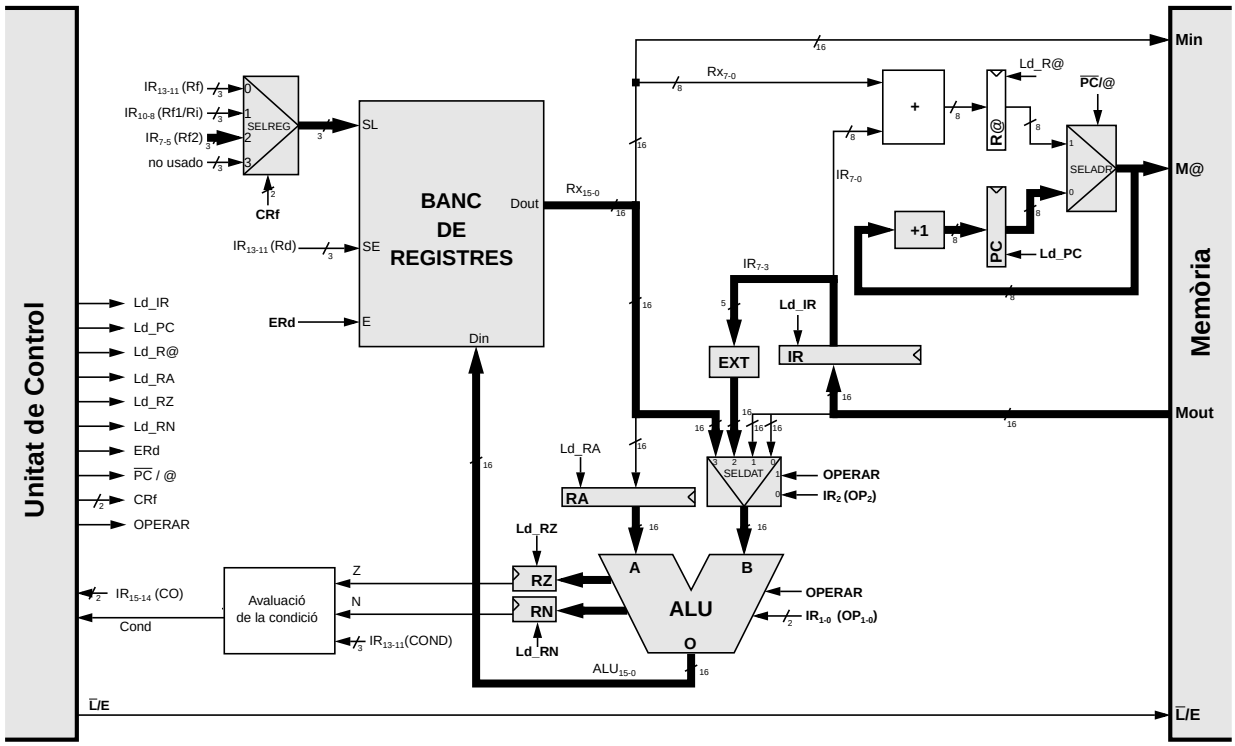
- Durant l'estat LSOE no es fa servir la part de la UP dedicada al *fetch* de la següent instrucció. Per tant, podem fer aquesta feina en el propi estat LSOE, de manera que en acabar l'execució d'una instrucció A-L la instrucció següent s'haurà carregat a l'IR. Per tant, l'estat futur de LSOE passa a ser l'estat de decodificació.

  Després d'aquesta modificació, a l'estat LSOE passem a dir-li ARIT. La Figura 4.10.b mostra l'activitat a la UP durant aquest estat.

A la Figura 4.11 es mostra com queda finalment el graf d'estats i la taula de sortides corresponent.

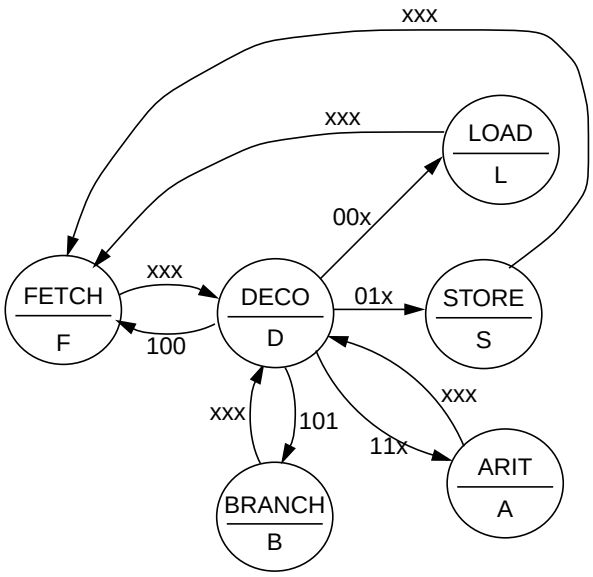


(a)



(b)

Figura 4.10. Activitat a la UP durant els estats DECO (a) i ARIT (b).



Sortides	F	D	A	L	S	B
Ld_IR	**1**	0	**1**	0	0	**1**
Ld_PC	**1**	0	**1**	0	0	**1**
Ld_R@	0	**1**	0	0	0	0
Ld_RA	0	**1**	0	0	0	0
Ld_RZ	0	0	**1**	**1**	0	0
Ld_RN	0	0	**1**	**1**	0	0
ERd	0	0	**1**	**1**	0	0
$\overline{L}$/E	**0**	0	**0**	**0**	**1**	**0**
$\overline{PC}$/@	**0**	x	**0**	**1**	**1**	**1**
CRf	xx	**01**	**10**	xx	**00**	xx
OPERAR	x	x	**1**	**0**	x	x

Figura 4.11. Graf d'estats i taula de sortides de la Unitat de Control de la Màquina Rudimentària.

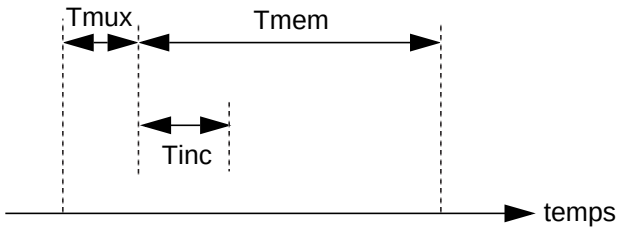
D'acord amb aquest disseny per a la Unitat de Control, les instruccions d'accés a memòria triguen 3 cicles en executar-se. Les instruccions A-L es pot considerar que triguen 2 cicles, ja que en l'estat ARIT es fa el *fetch* de la instrucció següent. I les instruccions de salt triguen també 2 cicles: FETCH i DECO, perquè l'estat BRANCH pot considerar-se també part de la instrucció següent.

La Unitat de Control de l'MR, igual que qualsevol sistema seqüencial, es pot implementar

de diverses maneres, per exemple amb una ROM i biestables.

Temps de cicle del computador

El temps de cicle d'un computador és la durada del període del senyal de rellotge. El temps de cicle d'un computador ha de ser prou gran com per què en un cicle es puguin dur a terme totes les tasques corresponents a qualsevol dels estats. Per exemple, les tasques que es fan en l'estat FETCH de la Màquina Rudimentària tenen la durada següent:



on *Tmux* és el retard del multiplexor SELADR, *Tmem* és el retard de la memòria i *Tinc* és el retard de l'incrementador. Per tant, el temps de cicle de l'MR ha de ser com a mínim *Tmux* + *Tmem*.

El temps de cicle d'un computador està determinat pel retard de l'etapa que té el retard més gran.

Exemple d'execució. Cronograma

A continuació es presenta un exemple de programació en Llenguatge Màquina i un cronograma que mostra l'estat de la UP i la UC durant uns quants cicles d'execució. El programa calcula la suma acumulada dels elements d'un vector de 4 elements.

Programa en alt nivell:

```
var V: vector[0..3] de enter;
suma, i: enter;
i:= 3;              /* el vector es recorre del revés */
suma:= 0;
mentre (i ≥ 0) fer
suma:= suma+vector[i];
i:= i-1;
fmentre
```

En la traducció a baix nivell es fan les suposicions següents:

- les variables *suma* i *i* es guarden a les adreces 00h i 01h respectivament
- les variables *suma* i *i* ja estan inicialitzades, a 0 i 3 respectivament
- el vector *V* es guarda en les posicions 02h...05h; els valors dels seus elements són 2, 8, 5 i -3
- el programa comença a l'adreça 06h

Programa en baix nivell:

	Llenguatge Assemblador	@	Llenguatge Màquina	en hexadecimal
		00h	0000 0000 0000 0000	0000h
		01h	0000 0000 0000 0011	0003h
		02h	0000 0000 0000 0010	0002h
		03h	0000 0000 0000 1000	0008h
		04h	0000 0000 0000 0101	0005h
		05h	1111 1111 1111 1101	FFFDh
carregar dades	LOAD 1(R0), R2	06h	0001 0000 0000 0001	1001h
	LOAD 0(R0), R3	07h	0001 1000 0000 0000	1800h
	LOAD 2(R2), R1	08h	0000 1010 0000 0010	0A02h
càlculs	ADD R1, R3, R3	09h	1101 1001 0110 0100	D964h
	SUBI R2, #1, R2	0Ah	1101 0010 0000 1001	D209h
	BGE 08h	0Bh	1011 0000 0000 1000	B008h
guardar resultat	STORE R3, 0(R0)	0Ch	0101 1000 0000 0000	5800h

Es pot observar que l'avaluació de la condició del bucle es fa després del cos del bucle i no abans. Això es pot fer sempre que es tingui la certesa que s'executarà almenys una volta del bucle (o en sentències *repeat*).

Observem també que el valor d'R2 és diferent a cada iteració, i per tant la instrucció LOAD 02h(R2), R1 accedeix a cada volta a un element diferent del vector.

La Figura 4.12 mostra l'evolució de la UP i la UC cicle a cicle durant l'execució de les 6 primeres instruccions del programa. En el primer cicle de l'execució el contingut de tots els registres és desconegut llevat del PC i l'R0, que s'assumeix que valen 06h i 0 respectivament. També s'assumeix que la UC està en l'estat de FETCH.

	LOAD 01h(R0), R2			LOAD 00h(R0), R3			LOAD 02h(R2), R1			ADD R1, R3, R3			SUBI R2, #1, R2		BGE 08h	
ESTAT	FETCH	DECO	LOAD	FETCH	DECO	LOAD	FETCH	DECO	LOAD	FETCH	DECO	ARIT	DECO	ARIT	DECO	BRANCH	DECO
Ld_IR	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0
Ld_PC	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0
Ld_R@	0	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1
Ld_RA	0	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1
Ld_RZ	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0
Ld_RN	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0
ERd	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0
$\overline{L}$/E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{PC}$/@	0	x	1	0	x	1	0	x	1	0	x	0	x	0	x	1	x
CRf	x	1	x	x	1	x	x	1	x	x	1	2	1	2	1	x	1
OPERAR	x	x	0	x	x	0	x	x	0	x	x	1	x	1	x	x	x
M@	06h	x	01h	07h	x	00h	08h	x	05h	09h	x	0Ah	x	0Bh	x	08h	x
Mout	1001h	x	0003h	1800h	x	0000h	0A02h	x	FFFDh	D964h	x	D209h	x	B008h	x	0A02h	x
Min	x	0000h	x	x	0000h	x	x	0003	x	x	FFFDh	0000h	0003h	0000h	0000h	x	0002h
Din	x	x	0003h	x	x	0000h	x	x	FFFDh	x	x	FFFDh	x	0002h	x	x	x
PC	06h	07h	07h	07h	08h	08h	08h	09h	09h	09h	0Ah	0Ah	0Bh	0Bh	0Ch	0Ch	09h
IR	x	1001h	1001h	1001h	1800h	1800h	1800h	0A02h	0A02h	0A02h	D964h	D964h	D209h	D209h	B008h	B008h	0A02h
R@	x	x	01h	01h	01h	00h	00h	00h	05h	05h	05h	61h	61h	0Ch	0Ch	08h	08h
RA	x	x	0000h	0000h	0000h	0000h	0000h	0000h	0003h	0003h	0003h	FFFDh	FFFDh	0003h	0003h	0000h	0000h
RZ	x	x	x	0	0	0	1	1	1	0	0	0	0	0	0	0	0
RN	x	x	x	0	0	0	0	0	0	1	1	1	1	1	0	0	0
R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R1	x	x	x	x	x	x	x	x	x	FFFDh	FFFDh	FFFDh	FFFDh	FFFDh	FFFDh	FFFDh	FFFDh
R2	x	x	x	0003h	0003h	0003h	0003h	0003h	0003h	0003h	0003h	0003h	0003h	0003h	0002h	0002h	0002h
R3	x	x	x	x	x	x	0000h	0000h	0000h	0000h	0000h	0000h	FFFDh	FFFDh	FFFDh	FFFDh	FFFDh

Figura 4.12. Cronograma de les 6 primeres instruccions del programa exemple.

5. Llenguatge Assemblador de la Màquina Rudimentària

Com ja hem vist, el Llenguatge Assemblador (LA) permet escriure programes en Llenguatge Màquina de manera més còmoda que les tires de 0's i 1's. Els programes escrits en LA són traduïts a LM per l'assemblador. L'assemblador rep com a entrada un fitxer contenint un programa escrit en LA, i genera com a sortida un fitxer que conté el programa traduït a LM i algunes informacions més que l'MR és capaç d'interpretar i que són necessàries per què pugui executar correctament els programes.

En la Màquina Rudimentària, els programes en LM s'ubiquen sempre a partir de l'adreça de memòria 00h. Mentre tradueix, l'assemblador manté el compte de quantes paraules de memòria s'han ocupat fins al moment i per tant a quina adreça s'ubicarà el que tradueixi a continuació.

A més del repertori d'instruccions, el Llenguatge Assemblador de l'MR té aquests altres elements: ***etiquetes***, ***directives*** i ***comentaris***.

Etiquetes

Són cadenes alfanumèriques seguides del caràcter ':' que s'escriuen al començament d'una línia d'un programa en LA. L'etiqueta permet identificar l'adreça de memòria on s'ubicarà el que es tradueixi a continuació.

Per exemple, suposem que el següent fragment de codi estarà ubicat a partir de l'adreça 0Ah de memòria una vegada traduït a LM:

```
        LOAD 0(R3), R1
        BNE 13
        ADD R2, R0, R1
etiq:   ASR R1, R1
```

La segona instrucció es podria escriure

```
        BNE etiq
```

Les etiquetes permeten que mentre s'escriuen els programes en LA no s'hagi de conèixer en quines adreces de memòria s'ubicaran una vegada traduïts a LM.

Directives

Són sentències del LA que no s'executaran. Permeten definir la zona de dades de la memòria i indicar on comença i acaba la part executable del programa. En les línies en LA que continguin directives també s'hi poden definir etiquetes.

El LA de l'MR té 5 directives:

=

Permet donar un nom a valors constants. Si escrivim per exemple

```
Num_elements = 100
```

l'identificador *Num_elements* es podrà fer servir en tot el programa per referir-se al valor 100. La definició de constants no implica ocupar cap espai de memòria.

.dw (*define word*)

Assigna un cert valor inicial a una o més adreces de memòria, a partir de la posició on apareix la directiva. Per exemple, si les primeres línies d'un programa en LA són

```
        .dw 0
vector: .dw 10, 11, 12, -1
```

les adreces de memòria 00h:04h contindran en començar l'execució del programa els valors 0, 10, 11, 12 i -1, codificats en complement a 2. A més, en el programa ens podrem referir a l'adreça 01h usant la paraula *vector*.

.rw (*reserve word*)

Reserva un cert nombre d'adreces consecutives de memòria sense donar-los cap valor inicial; la utilitat és guardar espai per posar-hi els valors calculats durant l'execució del programa. Per exemple, si un programa comença amb les directives

```
        .dw 10, -1
resultats: .rw 2
        .dw 100
```

en començar l'execució a la memòria hi haurà el següent:

ADREÇA	CONTINGUT
00h	000Ah
01h	FFFFh
02h	?
03h	?
04h	0064h
⋮	⋮

.begin

Serveix per indicar a l'MR quina instrucció haurà d'executar en primer lloc. En concret, s'escriu

```
.begin etiq_inici
```

on *etiq_inici* és una etiqueta definida just abans de la primera instrucció a executar del programa. Hi ha d'haver una i només una directiva .begin a tots els programes en LA. Es pot escriure a qualsevol lloc del programa.

La directiva .begin no es tradueix a LM, sinó que és una de les informacions addicionals que hi ha al fitxer generat per l'assemblador que permet a l'MR executar correctament els programes.

.end

S'ha de posar immediatament després de l'última instrucció que s'executarà. N'hi pot haver més d'una, si hi ha diferents punts d'acabada del programa. Tampoc no es tradueix a LM, sinó que genera informació de control per a l'MR.

En general, els programes en LA comencen per un determinat nombre de directives de definició de dades, i a continuació s'escriuen les instruccions.

Comentaris

Són textos que s'escriuen en els programes per tal de fer-los més entenedors a possibles lectors i al propi programador; es solen posar a la dreta de la instrucció o directiva a què fan referència. En una línia qualsevol d'un programa en LA, s'entén que és un comentari tot el que apareix després del caràcter ';'. Si escrivim

```
resul:   .rw 1              ; deixar espai per al resultat
  ini:   ADD R0, R0, R1     ; inicialitzar R1 a 0.
    ;    SUB R1, R2, R3
```

la instrucció SUB no es traduirà a LM, l'assemblador assumirà que forma part d'un comentari.

El procés d'assemblatge

Al programa assemblador se li dóna com a entrada un fitxer contenint un programa en LA. L'assemblador llegeix el programa dues vegades (es diu que els assembladors i compiladors són programes ***de dos passos***).

La primera vegada, cada cop que troba una definició de constant o d'etiqueta apunta el nom i el valor associat a la ***Taula de Símbols***. En el cas de les constants el valor associat és el que apareix a la dreta del "=", en el cas d'etiquetes és l'adreça on es defineix; l'assemblador de l'MR sempre assumeix que el programa s'ubicarà a partir de l'adreça de memòria 00h.

La segona vegada va traduïnt cada línia d'una manera o altra segons la directiva o instrucció que contingui. Quan troba una referència a una etiqueta o constant, la tradueix pel valor que ha apuntat anteriorment a la Taula de Símbols. Si a la Taula de Símbols no hi ha l'etiqueta o la constant, dóna un missatge al programador de què ha comès l'error d'usar un identificador sense haver-lo definit.

El simulador de la Màquina Rudimentària

Per a usar el simulador de la Màquina Rudimentària s'han de seguir els passos següents:

- El codi font s'ha d'escriure en un fitxer amb extensió *.mr*.
- Per a assemblar-lo, s'ha d'invocar la comanda *enmr* amb el nom de fitxer com a argument. El programa *enmr* genera dos fitxers:
  - un amb extensió *.asm* que conté el mateix codi que l'original però sense comentaris
  - un amb extensió *.cod* que conté el codi màquina executable.
- Per a simular l'execució del programa s'ha d'invocar la comanda *mr* amb el nom del programa com a argument.