# Introducció als Ordinadors:
# CAPÍTOL 3
# FORMAT DE LES INSTRUCCIONS

# CONJUNT D'INSTRUCCIONS

Instruccions: identificadors de les diferents tasques que sap realitzar la CPU.

Cada instrucció té assignat un codi binari únic.

**Definirem**

**instrucció = Codi binari que és capaç de decodificar la unitat de control de la CPU per tal de du a terme una tasca**.

**Format de la Instrucció** forma en que estan codificades les instruccions. Distribució en camps

Format depèn de cada CPU, però bàsicament consisteix en definir una sèrie de camps de cada instrucció,

Cada camp té una situació i longitud determinada.

## Camps d'instrucció

## Tipus d'instrucció

| **Codi d'operació** *( opcode*) | **Tipus d'instrucció** | **adreça** (*address field*) | **Mode d'adreça** |
|---|---|---|---|
| l'operació que es realitza | Diferents tipus d'instruccions | On es troben els operants | Mode d'adreçament de memòria |

Sovint el opcode i el tipus són un mateix camp

**TIPUS D'INSTRUCCIONS**

- **Instruccions de Registre**
- **Instruccions de Moviment o de Memòria**
- **Instruccions d'operació amb Immediats**
- **Instruccions de Salt Condicional i Incondicional**

# **INSTRUCCIONS DE REGISTRE** (*Register Instructions*)

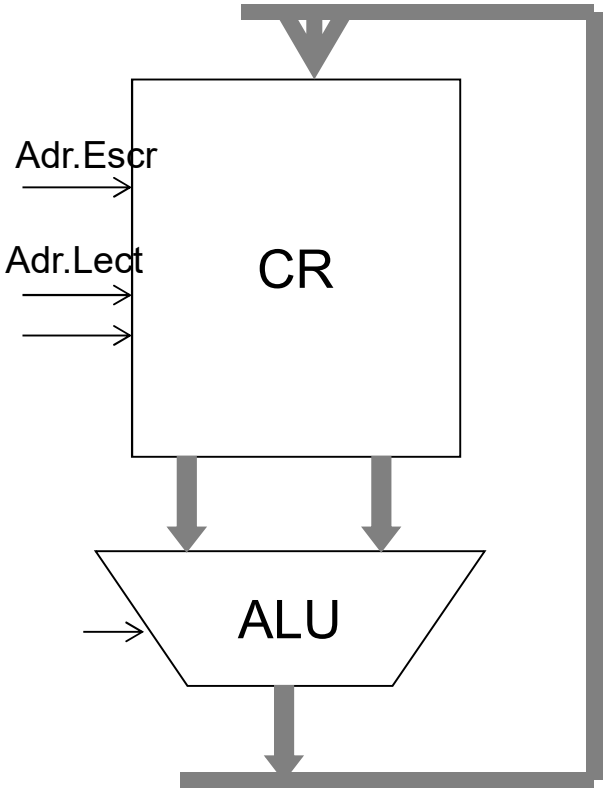
•Operen amb valors guardats a registres
•Realitzen operacions aritmètiques, lògiques sobre operants guardats en els registres de la UP.
•Alguns processadors implementen operacions aritmètiques o lògiques directament amb dades de la memòria

*Add* RA, RB, RC

equival a:

CR[A] <= CR[B] + CR[C]

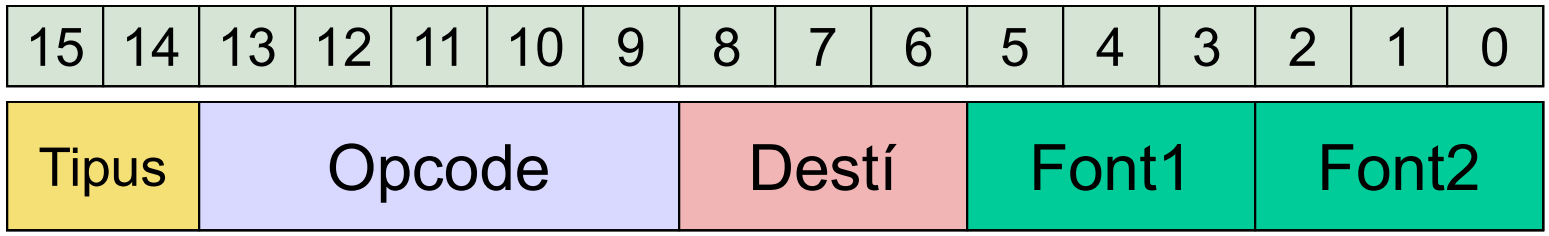
Llegeix continguts dels registres B i C, els suma i el guarda al registre A.




Depenent de l'arquitectura s'haura d'implementar en diversos cicles.

## **INSTRUCCIONS DE REGISTRE** (*Register Instructions*)

exemple




Aritmètica
Lògica
Moure
Desplaçar

| **Instrucció** | **acció** |
|---|---|
| Op Destí Fon1 Font2 | CR[Destí]<=CR[Font1] Op CR[Font2] |

# INSTRUCCIONS AMB IMMEDIATS

•Operen amb valors guardats a registres I valors numèrics que afegim a la pròpia instrucció

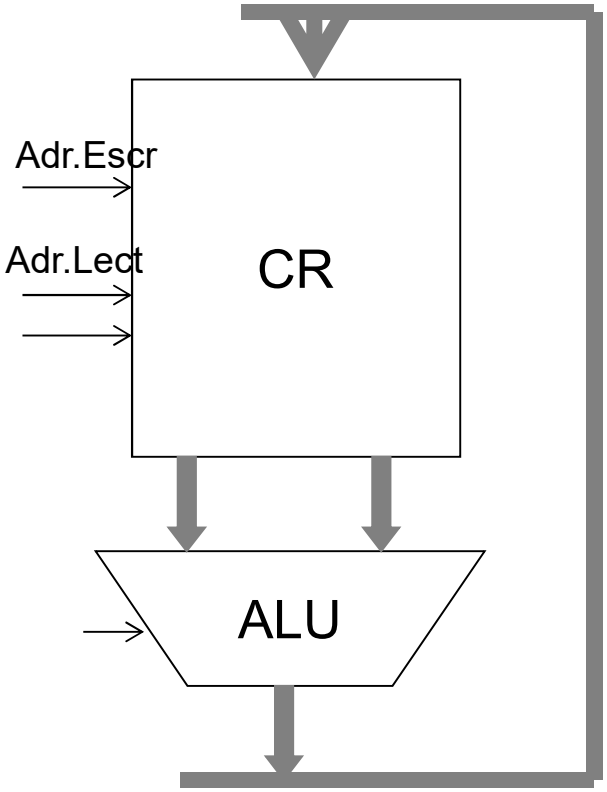
•Realitzen operacions aritmètiques, lògiques sobre operants guardats en els registres de la UP.

*Add* RA, RB, Imm

equival a:

CR[A] <= CR[B] + Imm

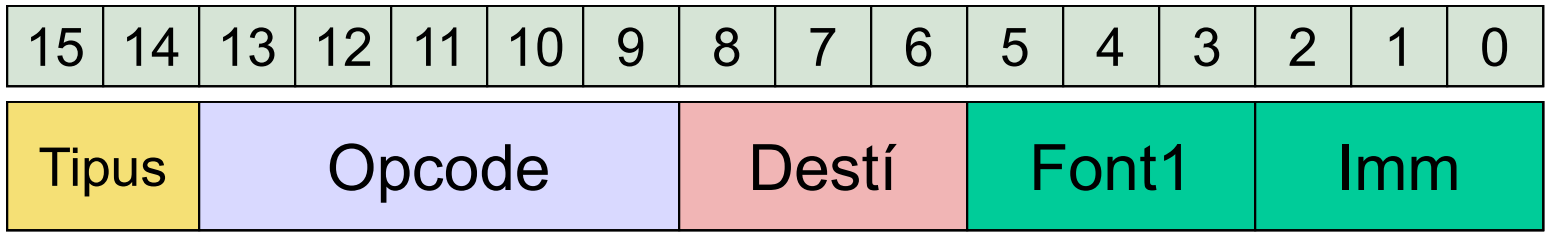
Llegeix continguts dels registres B i el valor immediat, els suma i el guarda al registre A.




Depenent de l'arquitectura s'haura d'implementar en diversos cicles.

## **INSTRUCCIONS DE REGISTRE** (*Register Instructions*)

exemple




Aritmètica
Lògica
Moure
Desplaçar

| **Instrucció** | **acció** |
|---|---|
| Op Destí Fon1 Font2 | CR[Destí]<=CR[Font1] Op Immediat |

# INSTRUCCIONS DE MOVIMENT O DE MEMÒRIA (*Move Instructions*)

Mouen dades entre memòria i registres.
Les instruccions més corrents de moviment són:

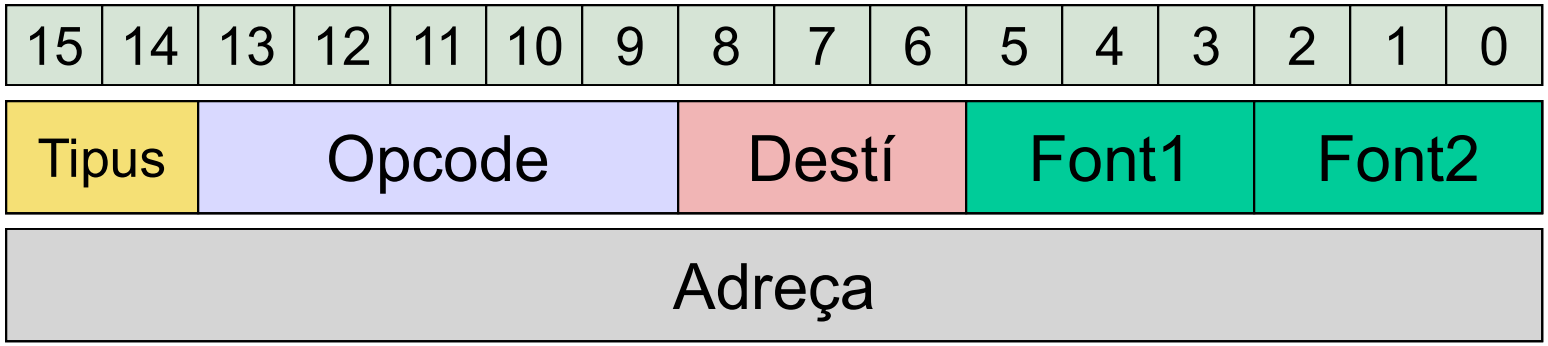
**Instruccions de càrrega** (*Load Instructions*)
Llegeix dada de posició de memòria i guarda en registre. Pex:

*Load* R2, A  equival a :  R2 <= Mem[A]

**Instruccions d'emmagatzematge** (*Store Instructions*)
Guarda contingut d'un registre a una posició de memòria.Pex:

*Store* A, R2  equival a :  Mem[A] <= R2

## INSTRUCCIONS DE MOVIMENT O DE MEMÒRIA (*Move Instructions*)




Load
Store

| instrucció | | acció |
|---|---|---|
| L imm Destí | Adreça | CR[Destí] <= Adreça |
| L dir Destí | Adreça | CR[Destí] <= Mem[Adreça] |
| S dir Font1 | Adreça | Mem[Adreça] <= CR[Font1] |
| L rel Destí, Font2 | Adreça | CR[Destí] <= Mem[CR[Font2] + Adreça] |
| S rel Font1, Font2 | Adreça | Mem[CR[Font2] + Adreça] <= CR[Font1] |
| L in Destí | Adreça | CR[Destí] <= Mem[Mem[Adreça]] |
| S in Font1 | Adreça | Mem[Mem[Adreça]] <= CR[Font1] |

Instruccions de Memòria exemple

# INSTRUCCIONS DE SALT CONDICIONAL I INCONDICIONAL (*Branch, Jump Instructions*)

Seleccionen la instrucció següent a ser executada en la UP

## instruccions de salt Incondicional

El programa salta sempre a una adreça determinada

## instruccions de salt Condicional

Si es compleix una determinada condició, fan saltar el programa a una instrucció, especificada en un operant, sino, s'executa la instrucció següent de la seqüència.

2 formes usuals de **comprovar la condició** d'una instrucció de **salt condicional**.

1. Registre d'Status
2. Comparació de valors guardats a registres

## **INSTRUCCIONS DE SALT CONDICIONAL I INCONDICIONAL** (*Branch, Jump*)

1.Codi de condició que s'actualitza quan s'executen algunes operacions; generalment estan continguts en un **registre d'estatus**.

Pex, resultat d'una operació aritmètica (*Add, Substract. ..*) Pot fixar un codi de condició amb quatre possibles valors,
zero, positiu, negatiu, desbordament.

Així podem tenir 4 instruccions de salt condicional:

*Brp* A Saltar a posició A si resultat positiu.
*Brn* A Saltar a posició A si resultat negatiu.
*Brz* A Saltar a posició A si resultat zero.
*Bro* A Saltar a posició A si produït desbordament

resultat de darrera operació exe. que afecti el codi d'estatus.

## **INSTRUCCIONS DE SALT CONDICIONAL I INCONDICIONAL** (*Branch, Jump*)

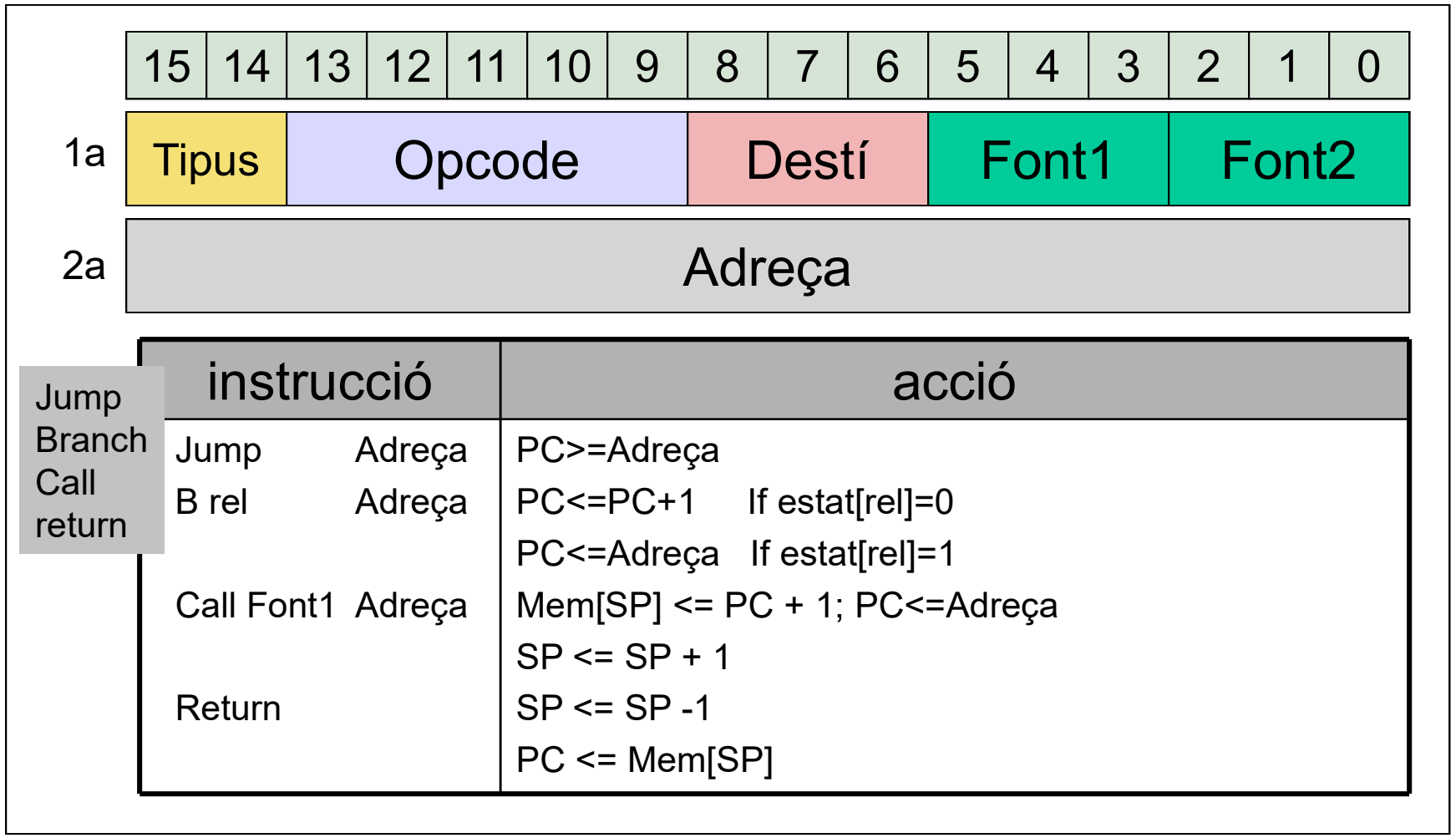
1. Un format d'instrucció de 3 camps.
   Compara els valors de dos registres i decideix la següent instrucció en base a la comparació;
   La comparació es fa a la mateixa instrucció.

Per exemple, la instrucció **Salta-quan-igual** (*Branch-on-equal o Beq*) comprova si els dos dels registres (R2 i R3) són iguals i si ho són executa la instrucció ubicada a la posició de memòria especificada (A):

*Beq* R2, R3, A

compara [R2] i [R3] i si són =
executa instrucció situada a  pos. de memòria *Mem*[A]
Mem[A] sol ser un offset que se suma al valor actual de PC

## INSTRUCCIONS DE SALT CONDICIONAL I INCONDICIONAL (*Branch, Jump*)




Jump
Branch
Call
return

| instrucció | | acció |
|---|---|---|
| Jump | Adreça | PC>=Adreça |
| B rel | Adreça | PC<=PC+1 If estat[rel]=0 |
| | | PC<=Adreça If estat[rel]=1 |
| Call Font1 | Adreça | Mem[SP] <= PC + 1; PC<=Adreça |
| | | SP <= SP + 1 |
| Return | | SP <= SP -1 |
| | | PC <= Mem[SP] |

Instruccions de salt exemple

## Algunes Instruccions de Ripes

AL

ADD R1,R2,R3

SUB R1,R2,R3

ADDI R1, R2,IMM

XOR R1, R2, R3

SLL R3, R1, R2  SRL R1,R2,R3

AND R1, R2, R3

@MEM

LOAD R1, value(R2) ➔ LB, LH, LW

STORE R1, value(R2) ➔ SB, SH, SW

@BRANCH

BEQZ R, MEM

BEQ R1, R2, M

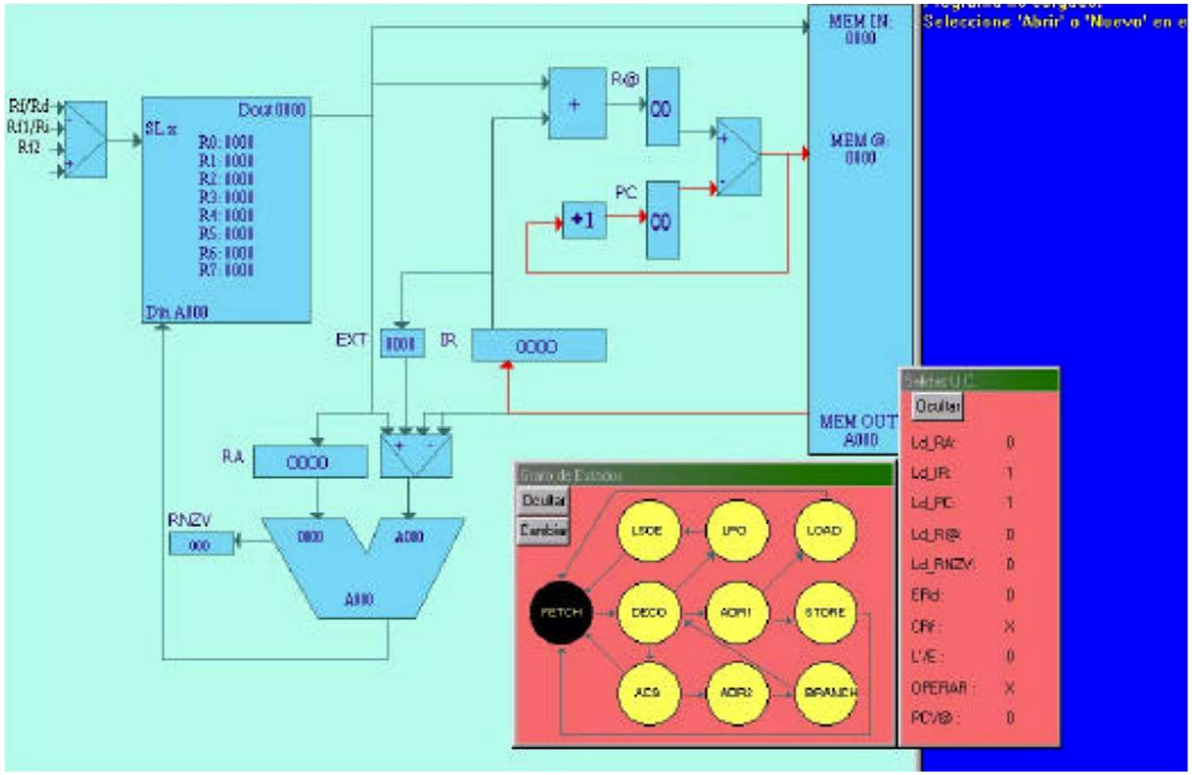
BNE R1, R2, M

BLT R1, R2, M

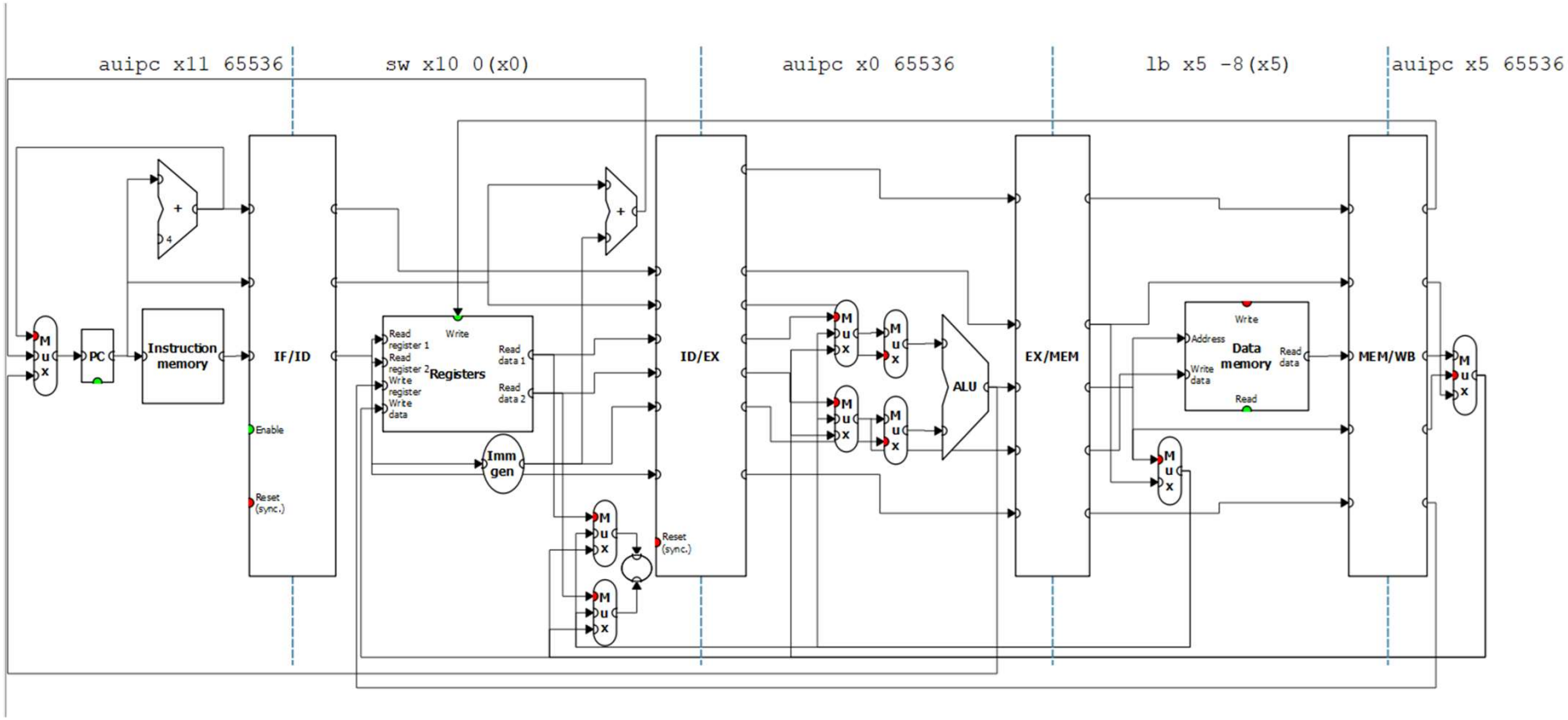
J M

BGE R1, R2, M

# Estructura Von Neumann. Arquitectura ISA - SiMR



# Estructura Harvard. ISA Ripes_RISC-V



# CONJUNT D'INSTRUCCIONS

En definir un conjunt d'instruccions és important considerar quants camps d'adreça podran contenir les instruccions.

El # de camps d'adreça afecta la longitud del programa i les prestacions del processador.

+camps d'adreça → + llargada → - programa (- instruccions)
però
instruccions + llargues → +accessos a memòria per portar instruccions i operands.

**Nombre d'accessos a memòria: paràmetre** per avaluar les prestacions.

Les memòries són + lentes que el processador, per tant és un paràmetre per avaluar la velocitat d'execució d'un programa

## CONJUNT D'INSTRUCCIONS

•Es dissenya el **CONJUNT D'INSTRUCCIONS** de manera que el nombre d'accessos sigui mínim.

•Instruccions amb diferent nombre de camps, des **d'1 a 3**

•Les instruccions de 3 camps són (en principi) les + potents, contenen la posició dels operands i del resultat.

•En gral no més de 3 camps, els recursos de l'UP són de 2 entrades. Moltes instruccions només 1 o 2 operands.

•Per altra banda una instrucció de 3 adreces ocupa molt d'espai, com sempre **cal un compromís**.

## Nombre d'adreces i accessos a memòria

Exemple:

•Analitzarem instruccions de diferent tipus i nombre d'adreces per computar l'expressió:

$$c = (a+b)*(a-b)$$

•Variables a,b,c i variable temporal x en posicions de memòria A, B, C i X, respectivament.

•Espai memòria: 16MB : 24 b adreces

•BUS 32 bits

El num. de bits que requereix una instrucció depén de la capacitat total de memòria

## Instruccions de 3 adreces

| 3 adreces | Opcode | Total | Num. Acc/instr |
|---|---|---|---|
| 72b | 8 b | 80 b | 3*32 = 96>80 |

3x24

Mem: 24 b
BUS: 32b

Total: **3** accessos/instrucció

**Accessos a memòria**

| programa | Fetch | Exe |
|---|---|---|
| 1. *Add* X, A, B (Mem[X] <= Mem[A] + Mem[B]) | 3 acces. | 3 acc. |
| 2. *Sub* C, A, B (Mem[C] <= Mem[A] - Mem[B]) | 3 acces | 3 acc. |
| 3. *Mul* C, X, C (Mem[C] <= Mem[X] * Mem[C]) | 3 acces | 3 acc. |

**Total 18 accessos a memòria**

## Instruccions de 2 adreces

el 1r op. i resultat comparteixen mateixa adreça de memòria.

| 2 adreces | Opcode | Total | Num. Acc/instr |
|---|---|---|---|
| 48b | 8 b | 56 b | 2*32 = 64 |

Mem: 24 b
BUS: 32b

2x24 Total: **2** accessos/instrucció

| Programa | | Fetch | Exe |
|---|---|---|---|
| *Move* X,A | (Mem[X] <= Mem[A]) | 2 acc | 2 acc (=>A , X<=) |
| *Add* X,B | (Mem[X] <= Mem[X] + Mem[B]) | 2 acc | 3 acc (B,X ; X) |
| *Move* C,A | (Mem[C] <= Mem[A]) | 2 acc | 2 acc |
| *Sub* C, B | (Mem[C] <= Mem[C] - Mem[B]) | 2 acc | 3 acc (C,B ; C) |
| *Mul* C, X | (Mem[C] <= Mem[C] * Mem[X]) | 2 acc | 3 acc (C,X ; C) |

10 acc. per portar instruccions i
13 per llegir/guardar operands i resultats, **Total 23 accessos a memòria**

Per tant, temps d'execució és major.
Hem necessitat 2 instr. extra de tipus *Move* (la 1 i la 3)

## **Instruccions de 1 adreça i acumulador (ACC)** dedicat

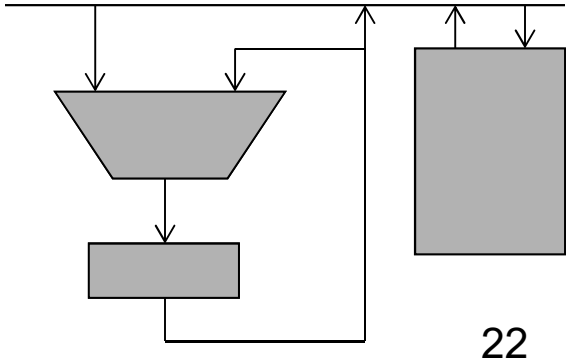
Reducció de nombre d'accessos. ACC guarda resultat d'operació i algun operand

Programa : 7 instruccions

**Accessos a memòria**

| Programa | Fetch | Exe |
|---|---|---|
| *Load* A (ACC <= Mem[A]) | 1 acc | 1 acc |
| *Add* B (ACC <= ACC + Mem[B]) | 1 acc | 1 acc |
| *Store* X (Mem[X] <= ACC) | 1 acc | 1 acc |
| *Load* A (ACC <= Mem[A]) | 1 acc | 1 acc |
| *Sub* B (ACC <= ACC - Mem[B]) | 1 acc | 1 acc |
| *Mul* X (ACC <= ACC * Mem[X]) | 1 acc | 1 acc |
| *Store* C (Mem[C] <= ACC) | 1 acc | 1 acc |

**Total 14 accessos a memòria**



## Instruccions de 2 adreces amb Conjunt de Registres (CR)

•Caldrà incloure camps d'adreces addicionals per adreçar el CR.

•Considerem el programa amb instruccions de dues adreces, on, una de les adreces especifica una posició del (CR),

•El Conjunt de Registres per aquest programa exemple disposa d'almenys cinc registres

## Instruccions de 2 adreces amb Conjunt de Registres

Programa : 6 instruccions

**Accessos a memòria**

| Programa | | Fetch | Exe |
|---|---|---|---|
| *Load* R1,A | (CR[1] <= Mem[A] ) | 2 acc | 1 acc |
| *Load* R2,B | (CR[2] <= Mem[B] ) | 2 acc | 1 acc |
| *Add* R3, R1, R2 | (CR[3] <= CR[1] + CR[2] ) | 1 acc | |
| *Sub* R4, R1, R2 | (CR[4] <= CR[1] - CR[2] ) | 1 acc | |
| *Mul* R5, R3, R4 | (CR[5] <= CR[3] * CR[4] ) | 1 acc | |
| *Store* C, R5 | (Mem[C] <= CR[5] ) | 1 acc | 1 acc |

**Total 11 accessos a memòria**

## Instruccions de 2 adreces amb Conjunt de Registres

•Només les Instructions de *Load* i *Store* accedeixen a memòria,
•Instruccions aritmètiques només accedeixen a regs ~~camp~~ adreça + curt

El CR petit : accés amb pocs bits d'adreça.

Cal usar
• instruccions de 3 camps on els operands estiguin en regs i
• instruccions de 2 adreces en les d'accés a memòria.

Aquesta estratègia usada per la majoria de processadors:
Instruccions més curtes i menor freqüència d'accessos

Estratègia avalada pel fet que cada variable en el programa s'usa més d'un cop, i llegir-les des d'un CR intern i ràpid redueix el nombre d'accessos a la memòria principal + lenta

## Resum d'accessos/instruccions

| instruccions | Longitud ins. | # instr | Total bits | # acc. mem |
|---|---|---|---|---|
| In-3adr | 80 b | 3 | 288 | 18 |
| In-2adr | 56 b | 5 | 320 | 23 |
| In-1adr+ACC | 32 b | 7 | 224 | 14 |
| In-1adr-CR | 32 b | 6 | 216 | 11 |

# MODES D'ADREÇAMENT DE MEMÒRIA

**camp d'adreça**: conté informació necessària per:

- determinar posició d'operands
- determinar posició del resultat de l'operació.

Diversos modes d'interpretar la informació de camp d'adreça.

•Diferents modes d'adreçament,
•Reduir el camp d'adreça, especificant només part de l'adreça, mentre el "mode" defineix com calcular l'adreça sencera.

•En general, es necessiten modes d'adreçament per suportar diferents construccions de llenguatges de programació, estructures de dades i tasques de SOs: bucles, punters de dades, reubicació de programa i commutació de contextos.

# MODES D'ADREÇAMENT DE MEMÒRIA

La Disponibilitat en modes d'adreçament proporciona al rogramador la possibilitat d'escriure programes eficients en nombre d'instruccions i temps d'execució.

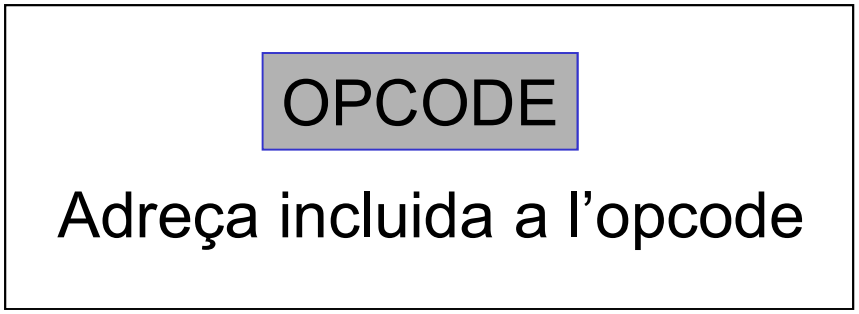
Els modes d'adreçament més comuns són:

1. Mode d'adreçament implícit
2. Mode d'adreçament immediat
3. Mode d'adreçament directe
4. Mode d'adreçament indirecte
5. Mode d'adreçament relatiu
6. Mode d'adreçament Indexat

# 1. Mode d'adreçament implícit

•No necessita camp explícit d'adreça pq la ubicació de l'operand o el resultat ja està especificada a l'opcode.

•Pex. instruccions de clear del registre d'estatus o l'acumulador. registres són únics i implícits en l'opcode (aquests registres solen ser únics i estan per tant implícits en l'opcode)

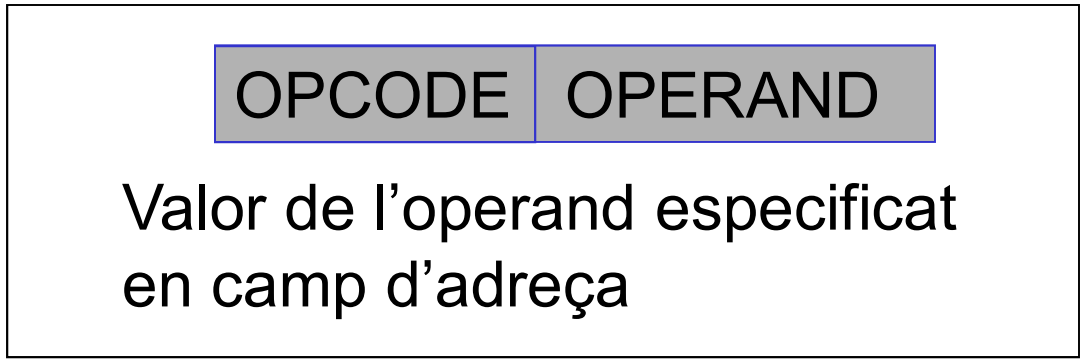



Adreçament Implícit

## 2. Mode d'adreçament immediat

•Operand especificat en camp d'adreça.
•Una instrucció d'adreçament immediat té un camp d'operand en el lloc del camp d'adreça

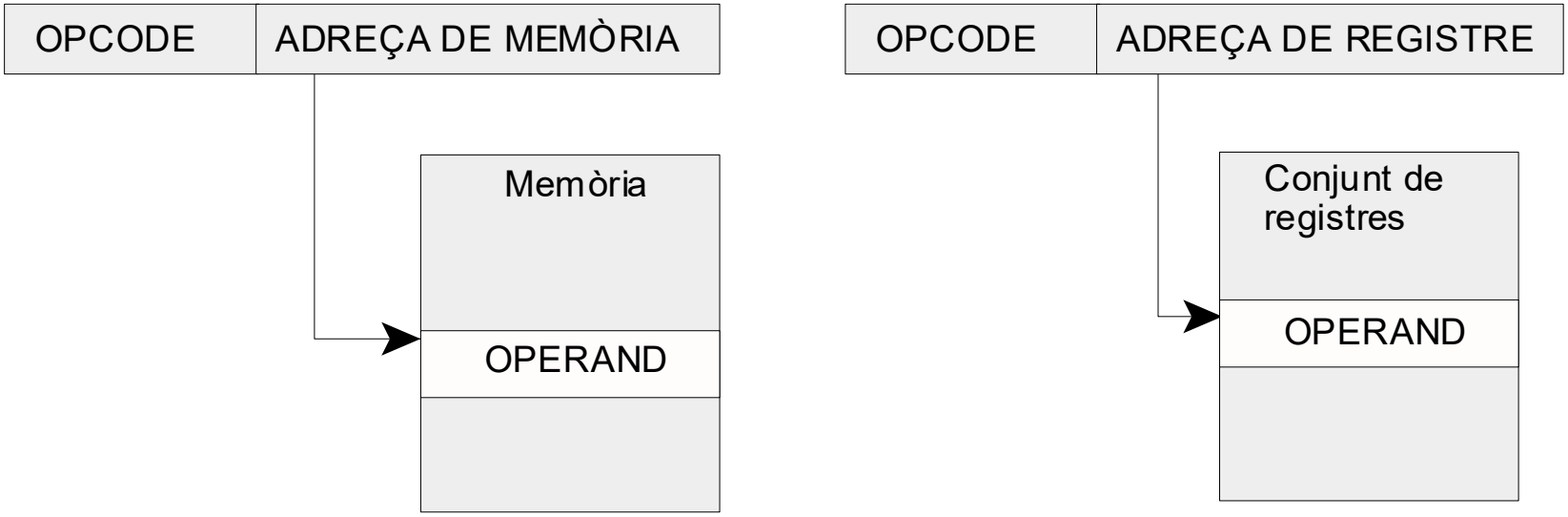
•Per especificar constants usades com operands en l'operació. Quan es treballa amb **constants** que poden ser subministrades en camp d'adreça en lloc d'accedir-les des de la memòria, estalvia, per tant, accessos a memòria.




Adreçament IMMEDIAT

# 3. Mode d'adreçament directe

- El Camp Adreça especifica la posició de l'operand
- Adreçament directe a memòria o a registre del CR
- Cal tenir en compte que l'adreça de memòria sempre serà molt més gran que la del CR




Adreçament DIRECTE

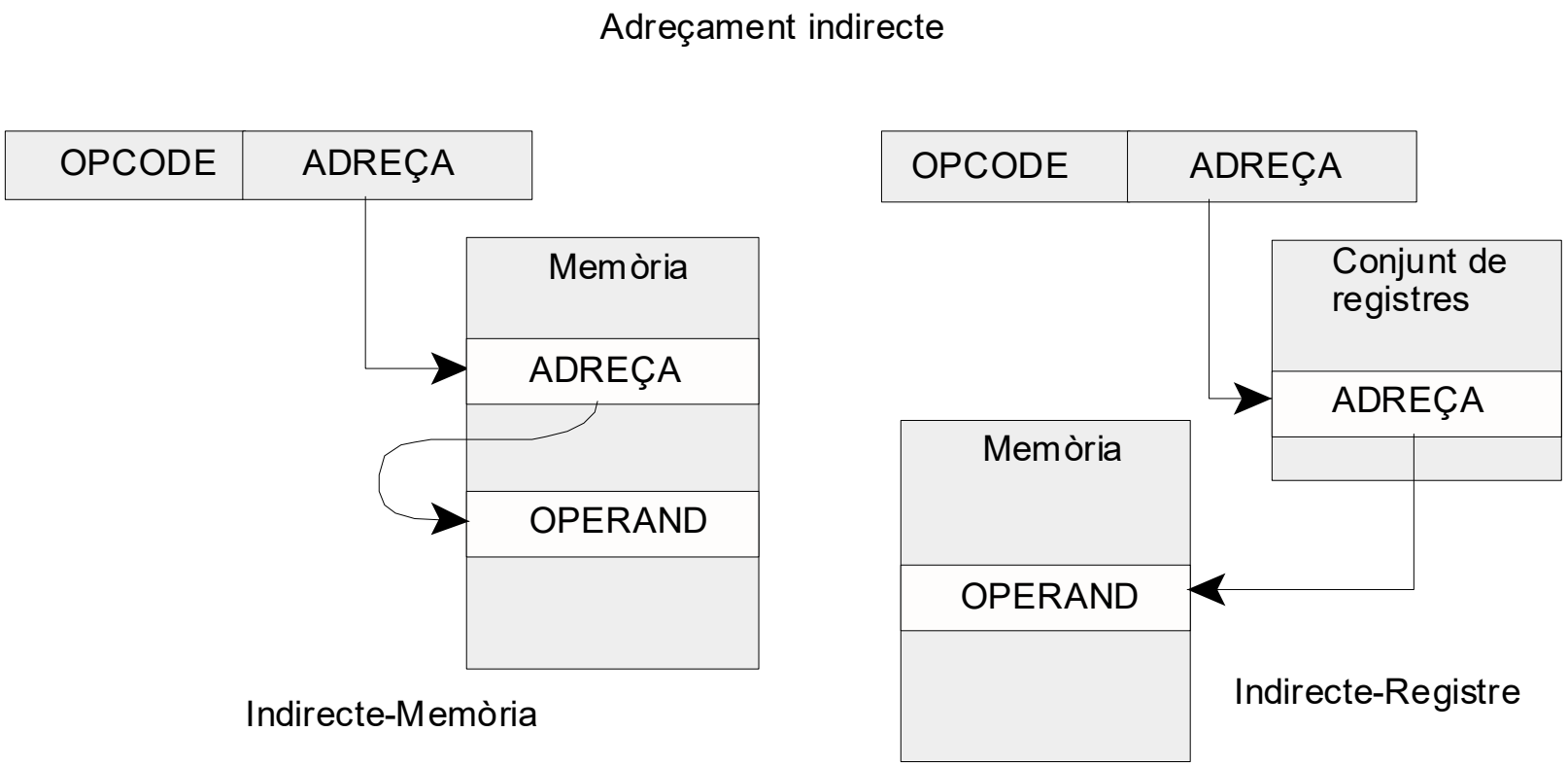
# 4. Mode d'adreçament indirecte

•Camp adreça especifica la ubicació de l'adreça de l'operand

•El processador ha d'accedir a memòria el doble de cops que en l'adreçament directe, 1 per llegir l'adreça i 1 per llegir l'operand o guardar el resultat

•En el mode **indirecte-registre** el camp d'adreça conté la direcció del registre que conté l'adreça de l'operand; el programador s'ha d'assegurar que l'adreça està en el registre adequat, abans d'accedir-hi

# 4. Mode d'adreçament indirecte

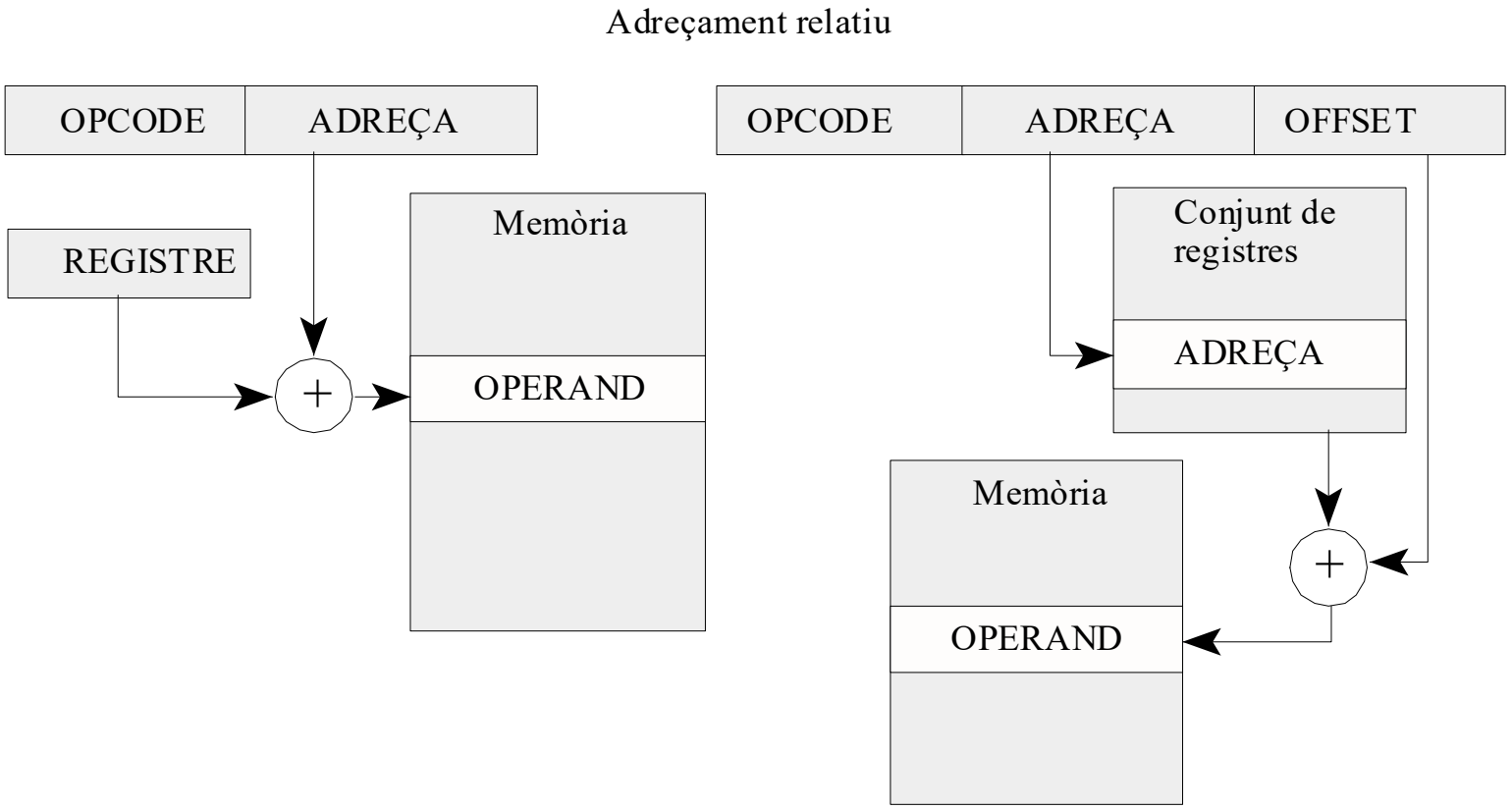



Adreçament indirecte

## 5. Mode d'adreçament relatiu

•Camp adreça=*offset*, és sumat al contingut d'un registre especificat, com el (PC) o un registre del (CR).
•L'*offset* és enter (positiu o negatiu). Quan l'*offset* es suma al PC la suma és l'adreça d'una instrucció de posició propera a la instrucció apuntada pel PC.
•Per tant aquest mode és usat en instruccions de salt condicional, ja que l'adreça de salt condicional està generalment a prop de la pròpia instrucció de salt.
•També l'adreçament relatiu es pot utilitzar respecte a qualsevol registre del CR, en aquest cas implementa taules *Look-up* en què el registre conté l'origen de la taula i l'*offset* apunta a un element específic

# 5. Mode d'adreçament relatiu




Adreçament RELATIU

## *6.* Mode d'adreçament Indexat

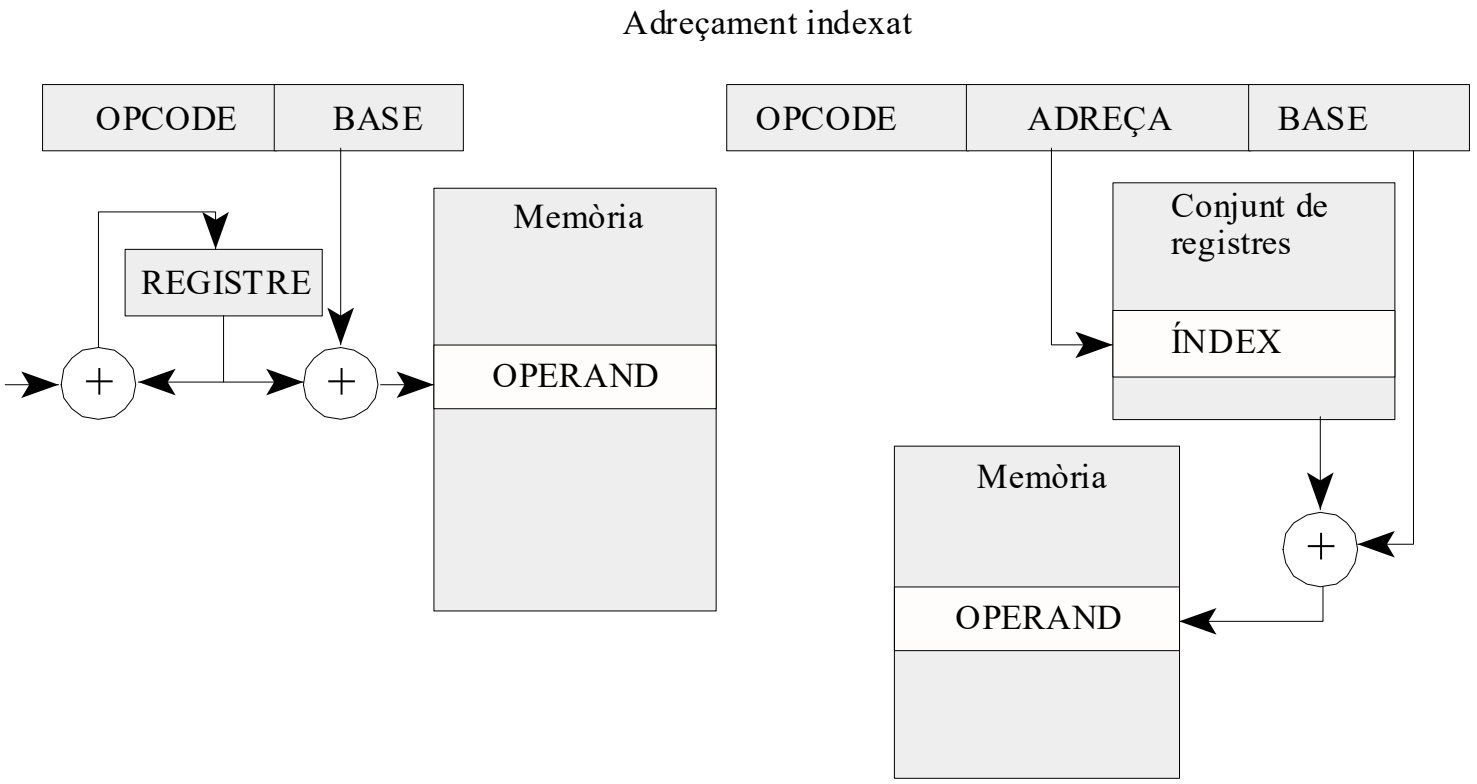
•L'adreçament indexat s'usa quan es necessita accedir a dades guardades en vectors (*arrays*), piles o cues.
•L'adreça especifica una adreça inicial, anomenada base, l'índex d'una dada particular s'especifica en un registre dedicat d'índex o en un del CR.
•Càlcul d'adreça efectiva: R(base) + R(índex)
•En algunes instruccions el valor del R(índex) és incrementat o decrementat automàticament per accedir al proper element del vector.
•Aquest tipus d'instrucció s'anomena instrucció autoincrement o autodecrement. També contribueix a reduir el nombre de bits necessaris en el camp adreça.

## *6.* Mode d'adreçament Indexat

•El mode indexat és similar al relatiu, difereixen només en les posicions de la base i de l'índex o l'*offset*.

•En relatiu la base està en un registre dedicat i l'*offset* està en el camp adreça;
•En  indexat la base està en el camp adreça i l'índex està en un registre.
•El mode relatiu es sol utilitzar en salts i l'indexat en accés a dades.

# 6. Mode d'adreçament Indexat



## MODES D'ADREÇAMENT DE MEMÒRIA

En general la disponibilitat de diversos modes d'adreçament en un conjunt d'instruccions fa l'execució de programes més ràpida, però també incrementa la complexitat del processador.