Capítol 5

EL PROCESSADOR EDUP12

EduP12 és un processador educatiu de 12 bits pensat per a introduir al lector en els fonaments del computador. La motivacions que van portar al seu disseny són:

- Necessitat de tenir un processador educatiu per a l'aprenentatge de fonaments de computadors bàsic però potent.
- Que permetés comprendre l'arquitectura bàsica dels computadors.
- Amb un repertori coherent d'instruccions.
- Amb un conjunt adequat de modes d'adreçament.
- Amb l'objectiu de tenir un processador que, tot i ser educatiu, permetés poder ser emprat en aplicacions no educatives.
- Amb el desenvolupament d'eines d'aprenentatge adequades: assemblador, simulador elemental del nucli del processador, simulador de programes en assemblador d'un petit microcontrolador basat en el processador educatiu.
- Amb descripció VHDL del nucli del processador per a ser programat en FPGA.
- I amb codificació d'instruccions clàssica no massa diferenciada del repertori d'instruccions de processadors actuals, que permeti a l'estudiant passar immediatament a treballar amb microcontroladors estàndard.

El mercat ofereix diferents processadors que podrien ser emprats com a processadors pseudo-educatius. Empreses com Motorola, MicroChip (amb els microcontroladors PIC) o Atmel (amb els ATtiny o ATmega com a processadors simples) ofereixen múltiples microcontroladors que permeten el disseny fàcil d'aplicacions. En cap cas, però, tenen eines de treball que mostren el seu nucli, essencial en assignatures d'introducció al computador. Tampoc es té disponible un codi VHDL que permeti desenvolupar sistemes basats en processador educatiu sobre FPGA. Per això caldria anar a cercar processadors RISC com ARM (Acom Computers), NIOS (Altera) o Microblaze (Xilinx), per exemple. Però en aquests casos ens trobem davant de processadors massa complexes si es volen emprar en l'ensenyament bàsic de fonaments de computadors.

Amb EduP12 s'aconsegueix una versió simple però pràctica d'un processador educatiu RISC. Les característiques fonamentals del processador són:

- És un processador de 12 bits. Tot i que la paraula conté un número de bits no múltiple de 8. Els seus avantatges davant alguns dels processadors de 8 bits existents són:
    - Es pot treballar adreçant directament sobre memòries de fins a 2^{12} paraules.
    - Opera amb dades de 12 bits, incrementant significativament la resolució dels processadors de 8 bits.
    - I es simplifica el repertori d'instruccions, fent que la major part de les instruccions necessitin només d'un accés a memòria per a la seva execució.
- L'ample del registre d'instruccions és de 16 bits.
- Es basa en una arquitectura Harvard, amb memòries de programa i de dades separades.
- Les memòries de programa i de dades poden tenir entre 256 i 4096 paraules. La paraula en la memòria de programa és de 16 bits, mentre que en la de dades és de 12 bits.
- Els ports d' entrada/sortida són registrats, fet que permet veure l'entrada/sortida com a una memòria addicional de 64 paraules.
- La unitat de control és seqüenciada, el que permet introduir noves instruccions si és necessari.
- Té una unitat d'interrupcions. El nombre d'interrupcions pot ser programat per un usuari avançat.
- L'E/S també és personalitzada per un usuari avançat. La llibertat d'introduir noves prestacions (amb la limitació de fins a 64 registres per E/S) és gran donat que es tracta d'una *Intellectual Property* (IP)[1].

A continuació s'especifica en detall la composició i funcionament d' EduP12.

5.1. Estructura del processador.

El processador s'estructura en el següents components:

- Unitat central de procés o CPU, que es composa d'unitat de procés i unitat de control basada en màquina d'estats finits.
- Memòria de programes. És una memòria d'escriptura/lectura. L'escriptura es realitza en el moment de carregar el programa. Durant l'execució del programa només és permesa la lectura de dades. La lectura implica tant la cerca de noves instruccions com de dades prèviament emmagatzemades. Durant l'execució de certs programes pot ser convenient tenir dades organitzades en forma de taula en la memòria de dades.
- Memòria de dades. És una memòria de lectura/escriptura per dades que s'empren durant l'execució del programa. La pila (emprada en la crida a subrutines o en interrupcions) s'inicialitza en la posició de memòria més alta i corre cap a posicions inferiors.
- Memòria d'entrada/sortida. Tot port d'E/S es comunica amb la CPU a partir de registres que tenen com a missió principal la sincronització de les operacions amb l'exterior de la CPU. Des de la CPU es veuen als ports d'E/S com a registres de memòria que són accedits a través d'adreces prefixades.

[1] La màquina elemental ve predeterminada amb una configuració bàsica que facilita la comprensió del funcionament del computador.

La figura 5.1 mostra un esquema del cablejat establert entre els mòduls principals del processador.

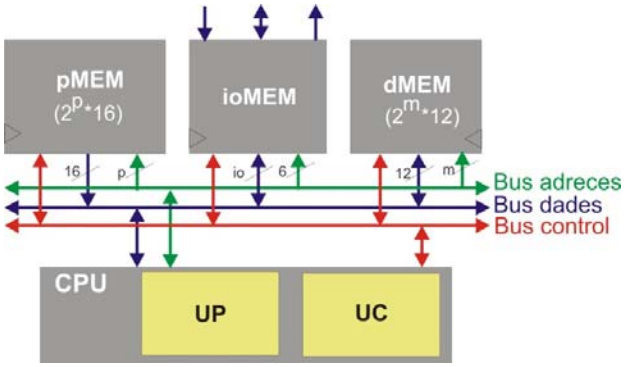

Figura 5.1. Estructura del processador

Els propers apartats analitzen en detall les unitats funcionals del processador.

5.2. Elements interns de la CPU

La CPU és la unitat funcional del processador. Està composta per:

- La unitat de procés o *data-path*. És la unitat encarregada d'efectuar les operacions aritmètiques i lògiques del processadors a través dels seus components interns.
- La unitat de control, que s'encarrega del seqüenciament de les operacions. Es basa en una màquina d'estats finits que seqüencia l'execució de les instruccions.

La figura 5.2 mostra l'estructura fonamental de la CPU. Només es mostren els components fonamentals per poder entendre millor el funcionament de la CPU. Hi ha inclosa també la memòria de programa (anomenada pRAM) per visualitzar millor la interconnexió que s'estableix entre CPU i memòria de programa.

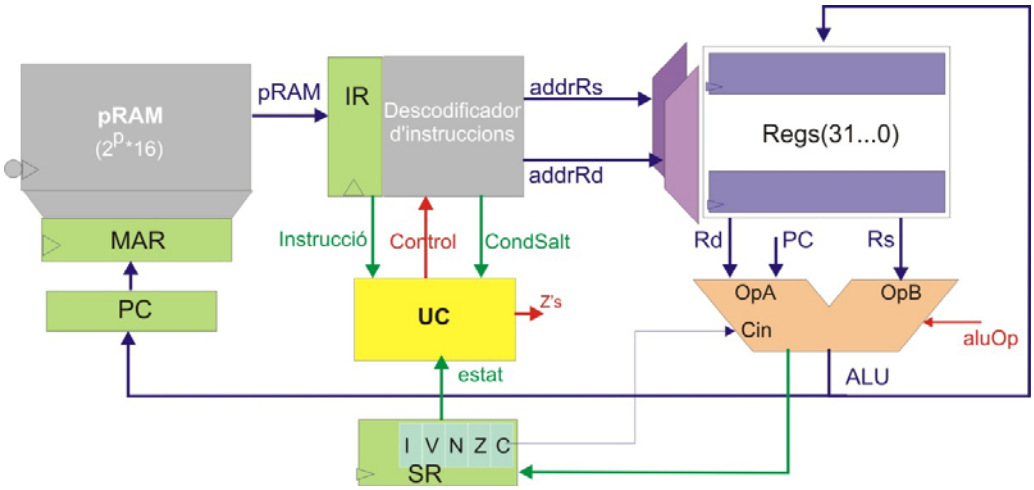

Figura 5.2. La unitat central de procés

5.2.1 Composició de la unitat de procés

Els components essencials que composen la CPU són els següents:

- La unitat aritmètico-lògica (ALU o UAL).
    - Pot realitzar operacions d'un o de dos operands. Els operands amb els que opera procedeixen del banc de registres.
    - Rep com a entrades dos operands i un bit de carreteig (quan ho requereix la instrucció) d'etapes anteriors i dóna com a sortida un resultat.
    - Realitza les operacions aritmètiques i lògiques següents: suma i resta (amb i sense carreteig), increment, decrement, and, or, or-exclusiva, negació (complement a 2), inversió (complement a 1), i operacions de desplaçament i rotació a la dreta i a l'esquerra.
    - Les instruccions aritmètico-logiques (que operen amb la ALU) actualitzen el registre d'estat.
    - A efectes d'agilitzar el pas de dades i realitzar les operacions d'adreçament la ALU també admet com a operands dades procedents de memòria i del comptador de programes, tot i que en aquests casos no s'actua sobre el registre d'estat de la CPU.
- Un banc de 32 registres (Reg31 a Reg0). El banc de registres és una memòria RAM interna ràpida d'accés immediat de la CPU. Constitueix la font de dades amb les que opera la ALU. L'amplada de cada registre és d'una paraula.
- El comptador de programes (PC). És un registre que apunta a l'adreça de memòria que conté la instrucció que s'executarà. Com que el comptador apunta a la memòria de programa, la seva amplada ha de ser suficient com per adreçar tota la memòria de programa.

  En EduP12 la màxima amplada del comptador de programes és de 12 bits, permetent adreçar una memòria de 2^{12} posicions de memòria. Per a realitzar l'increment del comptador de programes s'aprofita la UAL.
- El registre d'instruccions (IR) i el descodificador d'instruccions. El registre d'instruccions conté la instrucció que s'està executant. Com que les instruccions són de 16 bits, l'amplada del registre d'instruccions és de 16 bits.

  El registre d'instruccions té associat un descodificador d'instruccions, llur missió és la de recollir directament del registre d'instruccions la informació que prepara a la unitat de procés per a l'execució de la instrucció. Entre aquesta informació s'hi troba la selecció dels operands (immediats o des de registre), els registres font i destí del banc de registres, senyals de condició sobre la unitat de control i l'adreça d'E/S quan s'usa la memòria d'E/S.
- El registre d'estat (*status register* o SR). Conté informació relacionada amb el resultat que s'ha obtingut durant l'execució d'una instrucció aritmètica o lògica. Consta de 5 bits que proporcionen la següent informació:
    - Bit 0 o C. És el bit de carreteig. Es posa a 1 si l'operació provoca carreteig (*carry*).
    - Bit 1 o Z. Bit de zero. És 1 quan el resultat dóna un número que és zero.
    - Bit 2 o N. Bit negatiu. És 1 quan el resultat dóna un número negatiu. Com que es treballa amb complement a la base (complement a 2), un número serà negatiu quan el bit més significatiu sigui 1.
    - Bit 3 o V. Bit d'*overflow*. Es posa a 1 quan l'operació produeix excés (*overflow*).
    - Bit 7 o I. És un bit d'interrupció. L'actuació sobre aquest bit no és per execució sobre la ALU, sinó que s'hi actua directament per software mitjançant les instruccions d'establiment o anul·lació d'interrupcions.

La informació del registre d'estat la fa servir el processador per realitzar els salts en l'execució d'un programa.

- Registre d'accés a memòria (MAR). Elsa registres MAR són registres intermigs de sincronisme d'operació entre la CPU i les memòries externes, que normalment són més lentes. S'utilitza un registre MAR tant per la memòria de programa com per la memòria de dades.

Addicionalment (i tal com es mostra en el següent apartat on es presenta el processador complert) EduP12 també conté els dos següents elements, indispensables en l'execució d'instruccions de salt a subrutina o en l'execució de rutines de servei d'interrupció:

- La pila o *stack.*

  La pila és una memòria de guarda temporal de dades. Les dades a guardar poden ser dades temporals generades durant l'execució del programa o les adreces de retorn de subrutines. Tot i que pot ser un component més de la CPU en EduP12 la pila es situa en la memòria de dades.

- L'apuntador a pila (SP).

  L'apuntador a pila és el registre que apunta a l'adreça actual de la pila o *stack* en la que es pot guardar una dada.

Apart dels components anteriors algunes màquines també requereixen de registres de guarda de dades en la sortida de dades de memòria. S'anomenen registres *memory buffer* i la seva funció és de sincronisme de les dades de sortida de la memòria amb la CPU.

5.2.2 Funcionament intern de la CPU

Tot i que a nivell de blocs la CPU es divideix en unitat de procés i unitat de control, funcionalment la CPU és un bloc compacte que funciona mitjançant el control que la UC exerceix sobre la UP mitjançant les variables de control i les decisions d'actuació que se'n deriven de la UP cap a la UC a través del registre d'estat.

El funcionament està molt pautat i repeteix els següents passos[2]:

- L'execució d'una instrucció la inicia el PC. El PC sempre apunta a la posició de memòria que conté la instrucció que s'ha d'executar. Es va, per tant, a cercar la instrucció en la memòria de programa passant pel registre MAR que sincronitza la operació.
- La instrucció es carrega en el registre d'instruccions i es descodifica. La descodificació proporciona informació a la UP de les connexions que s'han d'establir entre els seus components per a executar la instrucció.
- (Es suposa que s'executa una instrucció d'operació aritmètico-lògica de doble operand) La ALU, agafant els dos operands i executa l'operació. Els operands provenen dels registres font (Rs) i destí (Rd) del banc de registres. Després d'operar amb les dades dels registres la ALU guarda el resultat en el registre destí. L'operació executada per la ALU actualitza el registre d'estat que guarda informació sobre el resultat l'operació.
- Durant el procés d'execució de la instrucció s'incrementat el PC.
- En acabar el cicle anterior s'inicia l'execució d'una nova instrucció anant a cercar la instrucció a la que apunta el PC.
- La unitat responsable del sincronisme global és la unitat de control. Amb un senyal de rellotge sincronitza tots els events que es produeixen en la CPU, emetent els corresponents

[2] Es mostra un esquema genèric del funcionament de la CPU. En propers apartats es detallarà per a cada instrucció.

senyals de control que adeqüen el procés a executar en cada component de la unitat de procés per a cada instrucció. Al seu temps, mitjançant l'estat de l'operació executada per la UP s'actua sobre el flux futur del programa.

L'etapa d'execució de la instrucció que inclou els passos que es dediquen a la captura de la instrucció s'anomena **fase de captació** de la instrucció. L'etapa durant la que es realitza l'execució de la instrucció s'anomena **fase d'execució**. Algunes instruccions necessiten d'una etapa intermitja de cerca d'operands que s'anomena **fase de captura d'operands**.

La figura 5.3 resumeix en un esquema l'execució genèrica d'uns instrucció que segueix el processador. L'execució del cicle d'instrucció és totalment seqüencial i un rellotge sincronitza el flux de dades dintre de la CPU. La duració del cicle d'instrucció sol durar diferents cicles de rellotge. D'acord amb la figura 5.3, en EduP12 l'execució d'una operació aritmètico-logica dura 3 cicles de rellotge.

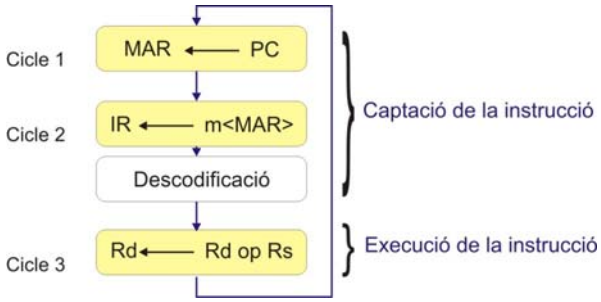

Figura 5.3. Cicle d'instrucció per a una instrucció aritmètico-lògica

Exemple 1. Execució d'una instrucció suma en EduP12

Es mostra un primer exemple d'execució d'una instrucció en el processador. Per a mostrar els passos essencials, es suposa que la unitat de procés ve donada per la figura 5.2.

Es suposa que la instrucció que s'executa és ADD R31, R30, que es troba guardada en la posició 100 de la memòria de programes, i que els registres R31 i R30 tenen, respectivament, els valors 0x300 i 0x500.

Es tracta d'una instrucció aritmètica que involucra als registres R17 i R16 i realitza l'operació

$$R17 \leftarrow R17 + R16$$

La figura 5.4 mostra el punt de sortida de l'execució d'aquesta instrucció. És la mateixa màquina presentada en la figura 5.2, però personalitzada per a l'execució d'aquesta instrucció. En la posició 100 de la pRAM hi ha guardada la instrucció que s'ha d'executar. El comptador de programes apunta a aquesta instrucció que es carrega en el registre d'instruccions. Els registres R31 i R30 contenen les dades amb les que s'operarà.

Per a executar la instrucció primer cal entendre la codificació de la instrucció, que es troba guardada en la posició 100 de memòria.

Si es mira la codificació de les instruccions (Apèndix A) es pot observar que la codificació de la instrucció ADD ve donada pels 16 bits següents

0000 10sd dddd ssss

Que indiquen:

- Els primers 6 bits són la codificació de la instrucció ADD.

- Els 5 bits *d* fan referència al registre Rd amb el que es treballa. Si es tradueix amb el codi màquina de la instrucció es veu que és R17.
- Els 5 bits *s* fan referència al registre Rs amb el que es treballa, que és R16.
- Es té sempre present la nomenclatura posicionals dels bits. Això és, que els bits a l'esquerra són els més significatius.

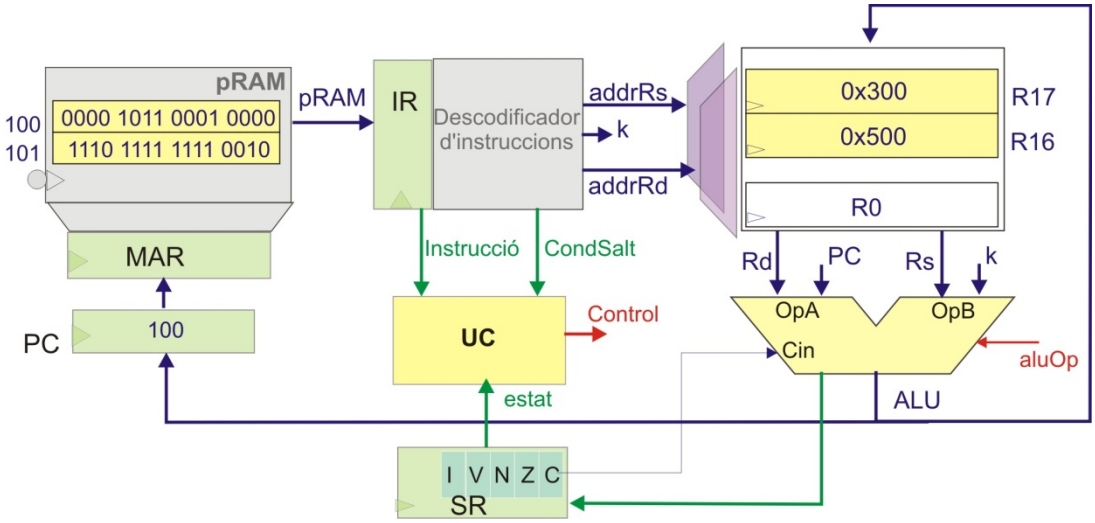

Figura 5.4. Estat de la CPU en el moment d'iniciar l'execució de la instrucció ADD R0, R1

Per tant, si es fa la correspondència dels bits de la codificació de la instrucció ADD amb l'exemple, es veu que la instrucció que hi ha en la posició 100 correspon a ADD R17, R16 (figura 5.5).

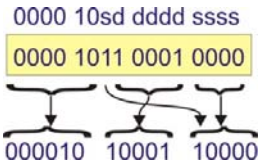

Figura 5.5. Correspondència de bits en la instrucció ADD R17, R16.

La seqüència de passos que el processador executa durant el cicle d'instrucció d'aquesta instrucció és:

- Cicle 1. Es carrega en el registre d'accés a memòria el comptador de programes. Per tant, el registre MAR apuntarà a l'adreça 100 de la memòria. En el següent flanc de rellotge el contingut de l'adreça apuntada per MAR (l'adreça 100) passa a ser accessible a la sortida de la memòria.
- Cicle 2. Seguidament la sortida de memòria (contingut de l'adreça 100 apuntada per MAR) es carrega en el registre d'instruccions.

  La instrucció es descodifica i es prepara la CPU per executar la instrucció. La descodificació, en aquest cas, implica trobar l'operació que ha d'executar la ALU i donar les adreces dels operands amb els que ha d'operar. La instrucció és la suma i registres amb els que s'opera són Rd=17 i Rs=16.

  Donat que durant aquest cicle no es treballa amb la ALU, s'aprofita per incrementar el comptador de programes. La UAL realitzarà l'operació d'incrementar el PC i l'actualitzarà. Per

tant, el PC ja apuntarà a l'adreça següent de la memòria, restant preparat per a l'inici de la propera instrucció.

- Cicle 3. Correspon a la fase d'execució de la instrucció. En aquesta fase la UAL realitza el càlcul. Concretament la UAL agafa els operands continguts en els registres Rd i Rs (0x300 i 0x500, respectivament) i els suma, posant el resultat (0x800) en el registre Rd.

  La suma de 0x300 amb 0x500 dóna com a resultat 0x800. Donat que el bit més significatiu del resultat passa a ser 1, el nombre passa a ser negatiu. Aleshores el bit N del registre d'estat passa a valdre 1. Per altra part, i donat que la ALU realitza l'operació de dos nombres positius i que el resultat obtingut passa a ser negatiu, també es posa a 1 el bit d'excés (V=1).

  Finalment, i com que la instrucció no requereix de cap cicle addicional, la UC inicia l'execució d'una nova instrucció.

Es pot observar que la seqüència d'operacions que es realitza durant l'execució d'una instrucció està totalment predeterminat i controlat a través de la unitat de control. La taula 5.1 mostra l'activitat que provoca en els registres l'execució de la instrucció ADD R17, R16.

Cicle	PC	MAR	IR	R17	R16	Estat	OpA	OpB	CodOp
Inicial	100	-	-	300	500	0b00000	-	-	-
Cicle 1	100	100	-	300	500	0b00000	-	-	-
Cicle 2	101	100	0x0B10	300	500	0b00000	PC	-	INC
Cicle 3	101	100	0x0B10	800	500	0b01100	Rd	Rs	ADD

Taula 5.1. Activitat en els registres de la CPU durant l'execució de la instrucció ADD R17, R16.

Exemple 2. Execució d'una instrucció de salt BRMI en EduP12

Es suposa ara que un determinat programa assemblador té escrit el següent tros de programa escrit a partir de la posició 100 de memòria, amb l'estat inicial en els registres R17 i R16 de l'exemple 1:

```
Bucle:  ADD R17, R16
        BRMI Bucle
```

El que fa el programa és molt simple: mentre el resultat de la suma del contingut dels registres R17 i R16 sigui negatiu el programa es queda executant el bucle.

Es tracta, així, d'executar la instrucció que hi ha en la posició 101 de la memòria pRam en la figura 5.4, que correspon a la instrucció BRMI. L'anàlisi d'aquesta instrucció mostra que es tracta d'una instrucció de salt. La figura 5.6 mostra la codificació d'aquesta instrucció.

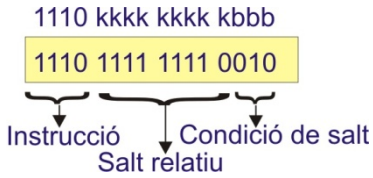

Figura 5.6. Anàlisi de la instrucció BRMI

D'acord amb la figura es pot observar que una instrucció de salt consta de tres camps:

- Els quatre primers bits, els més significatius, formen el camp de codificació de la instrucció. Codifiquen la instrucció i correspon a la instrucció BRMI.

- Els tres bits menys significatius corresponen al camp de condició. Pregunten sobre el bit a consultar per a efectuar el salt.
- La resta de bits formen el camp de salt. Indiquen el salt relatiu a efectuar.

Anàlisi de la instrucció de salt.

El salt en el context de la CPU involucra un canvi en la seqüencialitat d'execució d'un programa. La causa que pot provocar un salt en l'execució d'un programa és un canvi en algun dels bits del registre d'estat. Les instruccions de salt, per tant, pregunten sobre l'estat dels bits del registre d'estat. Concretament la instrucció BRMI pregunta sobre si un nombre és negatiu o no. Per tant, BRMI pregunta sobre l'estat del bit N.

El processador EduP12 pregunta sobre l'estat d'un bit en sentit d'activació o no activació. Per exemple, el bit N pot estar activat o no activat, fet que indica si el número és negatiu o positiu, respectivament. Preguntar si el bit N està activat implica preguntar sobre si el bit N es troba a 1-lògic, fet que implica executar una instrucció BRBS (*Branch if Bit is Set*). Per altre part, preguntar sobre si el bit N està desactivat implica preguntar sobre si el bit N es troba a 0-lògic, el que implica executar una instrucció BRBC (*Branch if Bit is Clear*).

Les instruccions BRBS i BRBC pregunten sobre l'estat dels bits del registre d'estat en sentit genèric. És a dir, pregunten sobre qualsevol bit del camp de condició. Sovint, però, aquestes instruccions es personalitzen per preguntar només sobre un bit. Així, per exemple, BRMI és la personalització de la instrucció BRBS quan pregunta sobre si el resultat és negatiu (bit N activat); i BRPL és la personalització de la instrucció BRBC quan pregunta per un resultat positiu (bit N no activat).

Una codificació 1111 en el camp d'instrucció correspon a una instrucció BRBC, mentre que la codificació 1110 correspon a una codificació BRBS. La codificació de la instrucció de l'exemple correspon, per tant, a una instrucció BRBS. L'exemple també té com a camp de condició (els tres bits menys significatius) el valor decimal 2. I aquesta és la posició que ocupa precisament el bit N. En conseqüència la instrucció de salt pregunta sobre si el bit N està activat. En conseqüència es tracta de la instrucció BRMI.

Finalment queda per determinar el salt a efectuar per la instrucció quan es compleix la condició de salt. El nombre d'instruccions que s'han de saltar ve determinat pel camp de salt, i és un salt relatiu respecte a la posició actual del comptador de programes. Com que el nombre de bits reservats per la constant k és de 9, el salt pot anar des de $+2^8-1$ a -2^8 posicions (compta que s'ha d'expressar en complement a la base!) a partir de la posició actual del comptador de programes. D'acord amb la figura 5.6 el salt a efectuar és de 111111110 posicions, corresponent a un salt de -2 posicions respecte a la posició a la que es troba el comptador de programes. Com que en el moment en què s'arriba al cicle d'execució de la instrucció BRMI el comptador de programes ja apunta a la posició 102 de memòria, de complir-se la condició de salt el programa tornarà a la instrucció que es troba en la posició 102-2 = 100 de la memòria de programa.

El senyal *CondSalt* que prové de la descodificació de la instrucció és el responsable de comprovar si es compleixen les condicions de salt. Aquest senyal el fa servir la unitat de control per a executar el salt en el cicle de rellotge que toca.

Execució de la instrucció de salt en el context de l'exemple.

Explicat el funcionament de les instruccions de salt és fàcil entendre el funcionament global del programa.

Es parteix de què s'ha executat la primera instrucció i ara en el comptador de programes s'hi troba el valor 101. L'execució de la nova instrucció passa pels següents passos:

- Cicle 1. El valor del comptador de programes passa al registre MAR de la memòria de programa.

- Cicle 2. La nova instrucció, que es troba en la posició 101, es carrega en el registre d'instrucció i és descodifica. Ja s'ha vist que es tracta d'una instrucció de salt BRMI.
- Cicle 3. S'avalua la condició de salt que pregunta sobre el bit N del registre d'estat. Com que en l'execució de la instrucció anterior s'activà el bit N, i com que la instrucció BRMI pregunta si el resultat fou negatiu, l'avaluació de la instrucció és positiva i es calcula la posició de la propera instrucció a executar. Aquest valor es carrega en el comptador de programes i es passa a executar la nova instrucció.

La taula 5.2 mostra l'activitat que se succeeix en el processador durant l'execució de la instrucció BRMI.

Cicle	PC	MAR	IR	R17	R16	Estat	OpA	OpB	CodOp
Inicial	101	100	0x0B10	800	500	0b01100	-	-	-
Cicle 1	101	101	0x0B10	800	500	0b01100	-	-	-
Cicle 2	102	101	0xEFF2	800	500	0b01100	PC	-	INC
Cicle 3(*)	100	101	0xEFF2	800	500	0b01100	PC	k	ADD

Taula 5.2. Activitat de la CPU durant l'execució de la instrucció BRMI bucle. (*) indica que es compleix la condició de salt i es calcula el valor del PC

Nomenclatura.

Durant l'execució d'una instrucció es segueix una seqüència de passos molt concreta en la que, fonamentalment, es mouen dades entre registres. La representació que es fa sol ser a base d'assignacions entre els diferents elements que composen la CPU. Degut a aquest motiu es sol anomenar aquesta representació de format de transferència de registres o RTL.

L'execució de les dues instruccions en aquest format, posant totes les cicles que calen per a executar-les vindria donada en la forma següent:

- Instrucció ADD R17, R16:

  Cicle 1: MAR ← PC;
  Cicle 2: IR ← mem<MAR>, PC ← PC + 1;
  Descodificació
  Cicle 3: R17 ← R17 + R16;

- Instrucció BRMI bucle:

  Cicle 1: MAR ← PC;
  Cicle 2: IR ← mem<MAR>, PC ← PC + 1;
  Descodificació
  Cicle 3: N = 1? (PC ← PC + 1) : (-);

Es pot observar que queden totalment enquadrades les accions a realitzar en cada cicle de rellotge. El format RTL permet expressar de forma coherent la transferència de dades que es produeix a cada cicle. Tot i que no hi ha sintaxi preestablerta, es sol representar sempre amb les símbols emprats en aquest exemple. La seva pròpia forma no dóna peu a confusió sobre les accions a realitzar en cada cicle. Aquesta representació es farà servir en el proper capítol per aprofundir en el llenguatge màquina del processador.

El sincronisme en el cicle d'instrucció.

Un senyal important del processador en l'execució del cicle d'instrucció és el senyal de rellotge. Les dades i els senyals del processador canvien d'acord a un sincronisme introduït pel senyal de rellotge. De forma general, tots els senyals de control s'activen durant el flanc de baixada del senyal de rellotge mentre que les dades que actualitza la unitat de procés ho fan en el flanc de pujada. Amb l'objectiu de guanyar un cicle les memòries actualitzen la sortida de dades en el flanc de baixada.

La figura 5.7 mostra un **diagrama temporal d'evolució del processador** en el que s'observa l'evolució dels registres durant l'execució de les dues instruccions anteriors. S'observa, en la figura, el sincronisme global que s'estableix pel funcionament correcte del processador.

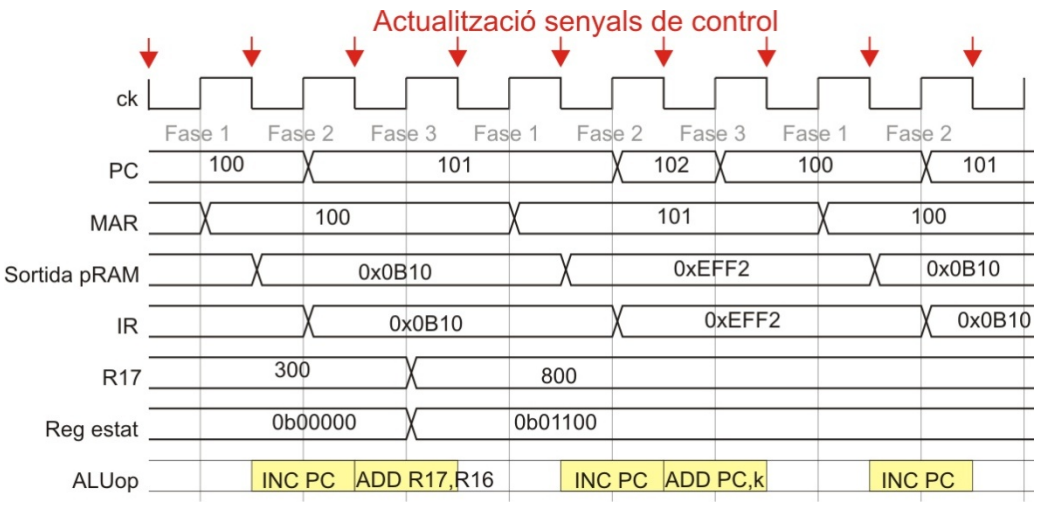

Figura 5.7. Diagrama temporal d'evolució del processador

5.3. El processador complert

La versió simplificada del processador emprada fins ara ha servit per comprendre els aspectes principals del funcionament del processador. En aquest apartat es presenta el processador complert. Els aspectes globals del funcionament del processador es basen en els conceptes introduïts i seguits en els exemples simples que s'han presentat fins ara.

La figura 5.8 mostra el processador amb els seus components i els senyals de control que actuen sobre els diferents mòduls.

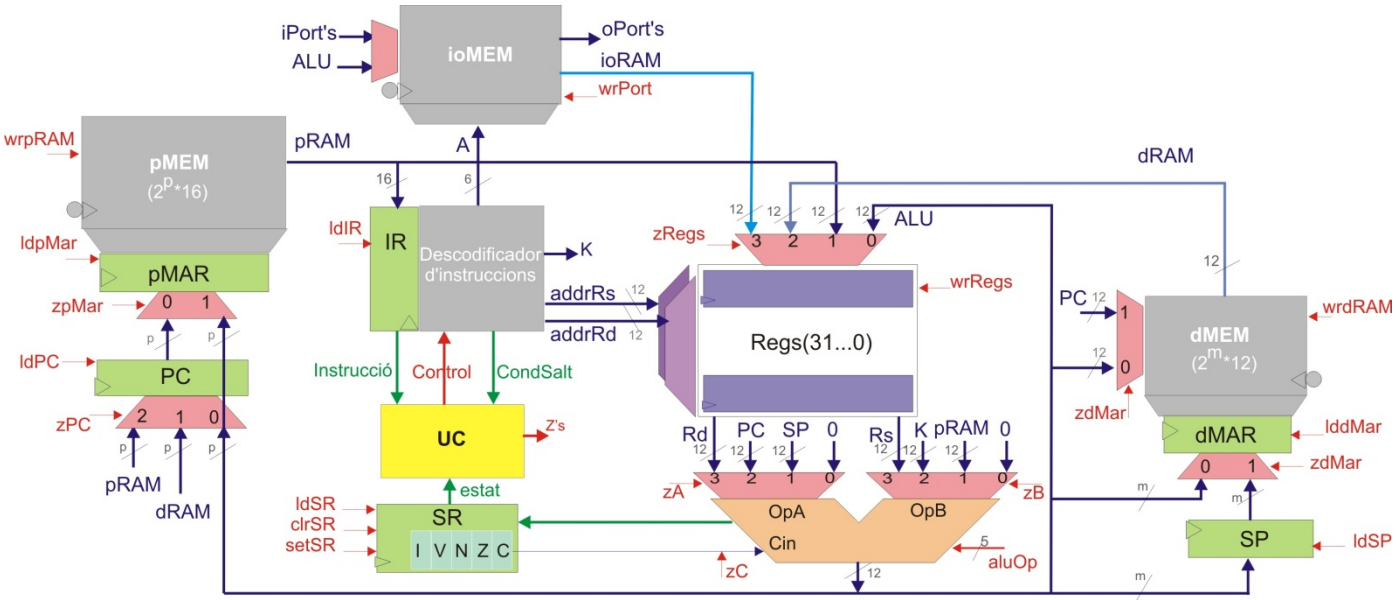

Figura 5.8. Arquitectura de EduP12

Característiques de la CPU addicionals a la CPU simplificada són:

- En la UAL els operands poden provenir directament dels components comptador de programes (PC), apuntador de pila (SP) i memòria de programa (pRAM). L'entrada K correspon a una constant continguda directament en la instrucció en instruccions que impliquen càlcul de salts relatius o amb immediats.
- Els registres configuren la memòria ràpida interna de la CPU de guarda de dades temporals. Per això hi tenen accés totes les memòries, apart de la UAL.
- L'accés a la memòria de dades es realitza a través de la UAL o amb l'apuntador de pila (SP). L'apuntador de pila s'inicialitza a la part final de la memòria de dades (a partir de la darrera adreça). S'espera que l'ús de la memòria de dades es faci a partir de la primera posició de memòria.
- La memòria d'entrada/sortida és una memòria heterogènia que inclou components diversos. Per pròpia construcció tots els components són registrats i incorporen una adreça que els hi és única. El rang d'adreces per a entrada/sortida és de 64. Les operacions que es poden realitzar amb aquests components poden ser d'entrada, de sortida o d'entrada/sortida.
- Per a clarificar la CPU no es mostren en el dibuix alguns detalls addicionals com, per exemple, els busos de guarda i restauració del registre d'estat.

L'execució de qualsevol instrucció en la CPU implica seguir processos similars als mostrats en els exemples 1 i 2.

Exemple 3: Execució d'una instrucció de càrrega immediata en registre (LDI) en EduP12

En els exemples 1 i 2 s'ha partit del cas en què ja s'havia carregat en els registres R17 i R16 un valor concret.

En aquest exemple es suposa que la càrrega del valor 300 en R17 s'efectua en la instrucció anterior, i que prové d'una càrrega amb valor immediat. Això és, s'executa la instrucció *LDI R17, 300* que carrega en el registre R17 el valor 300.

La figura 5.9 mostra l'estat de la memòria de programa introduint la nova instrucció en la posició 98 de memòria, i suposant que en les posicions 100 i 101 es tenen les instruccions dels exemples 1 i 2.

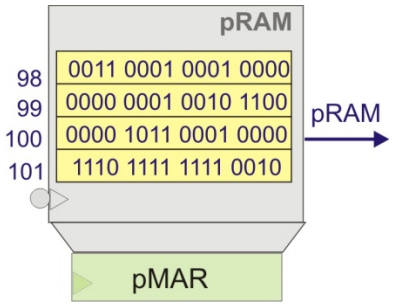

Figura 5.9. Introducció de la instrucció LDI R17, 300 en la posició 98 de la memòria.

El primer que es pot observar és que s'han posat dues paraules (posicions 98 i 99) i no només una. Es deu a que la instrucció LDI forma part d'un conjunt d'instruccions que operen amb valors immediats. Donat que l'ampla de bus del processador és de 12 bits, i es treballa amb instruccions de 16 bits, per permetre un repertori d'instruccions potent les instruccions que operen amb valors immediats necessiten de dues paraules. La primera paraula conté la codificació de la instrucció, mentre que la segona conté el valor. Això implica que l'execució d'una instrucció de càrrega requereix de dos accessos a memòria. La figura 5.10 mostra la descodificació de la instrucció.

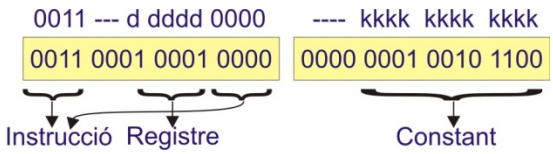

Figura 5.10. Descodificació de la instrucció LDI R17, 300.

Desenvolupar en format RTL el cicle d'instrucció d'aquesta nova instrucció no requereix res més que seguir el flux de dades que estableix l'arquitectura de EduP12 (figura 5.8). La taula 5.2 mostra que aquesta instrucció necessita 4 cicles.

Cicle	Transferència de dades (en RTL)	Activació de senyals de control
Cicle 1	MAR ← PC	zpMAR←0, ldpMAR←1
Cicle 2	IR ← mem<MAR>, PC ← PC + 1 (Descodificació)	ldIR←1, zA←2, aluOp←INC, zPC←0, ldPC←1 (addRd=17)
Cicle 3	MAR ← PC, PC ← PC + 1	zpMAR←0, ldpMAR←1, zA←2, aluOp←INC, zPC←0, ldPC←1
Cicle 4	R17 ← mem<MAR>	zRegs←1, wrRegs←1

Taula 5.2. Cicle d'instrucció de LDI R17, 300

La taula mostra clarament com l'accés a memòria per anar a cercar la constant a carregar en registre requereix d'un cicle de rellotge addicional. En aquest cas, per tant, calen dos cicles per la fase d'execució de la instrucció.

La tercera columna mostra els senyals de control que s'activen a cada cicle de rellotge. La unitat de control és la responsable d'activar adequadament a cada cicle. Es pot veure a la unitat de control com a una màquina d'estats finits que, per a cada instrucció passa per un conjunt d'estats (tants com cicles de rellotge calen per executar la instrucció), i que en cada cicle de rellotge activa els senyals de control que reconfiguren la unitat de procés per executar la corresponent transferència de dades entre registres o components del processador.

Exemple 4 Execució d'una instrucció de crida a subrutina RCALL en EduP12

Concepte

La crida a subrutina és una instrucció potent dels processadors que permet que un petit programa pugui ser executat repetidament sempre que calgui sense necessitat de reescriure'l cada cop que faci falta. En sí,

- La subrutina és un petit programa preparat per a ser cridat
- Es posa en una posició concreta de memòria i es crida a través del nom (etiqueta) amb el que se l'anomena. De fet, l'etiqueta només serveix a l'assemblador, ja que pel llenguatge màquina no és res més que una posició de memòria.
- La instrucció de crida és *RCALL etiqueta*. Les accions que fa són dues:
    - Guarda l'adreça actual del comptador de programes en una pila ja que, quan torni de la subrutina, haurà de continuar en seqüència.
    - Després carrega la posició d'inici de la subrutina (etiqueta) en el comptador de programes per iniciar l'execució de la subrutina.
- La instrucció que fa sortir de la subrutina és *RET*. La seva funció és recuperar de la pila l'adreça guardada i retornar-la al comptador de programes per a continuar l'execució seqüencial del programa.

La figura 5.11 mostra, amb un exemple, els passos que se succeeixen durant l'execució d'una subrutina. En l'exemple es crida la subrutina etiquetada *espera* amb la instrucció *RCALL espera.* En l'execució de la instrucció se succeeixen els passos 1-1 (es guarda el PC en la pila), 1-2 (es calcula la posició de memòria on es troba la nova instrucció) i 1-3 (es salta a l'inici de la subrutina), de manera que un cop guardada l'adreça del comptador de programes en la pila, es comença a executar la subrutina d'espera. Un cop acabada l'execució de la subrutina, RET retorna el control al programa principal. La instrucció de retorn executa les fases 2-1 (es va cercar a la pila l'adreça de retorn), 2-2 (es carrega en el comptador de programa) i 2-3 (es continua en seqüència d'allà on s'havia saltat a executar la subrutina). Un cop carregat el comptador de programes amb l'adreça següent a la instrucció RCALL es continua en seqüència.

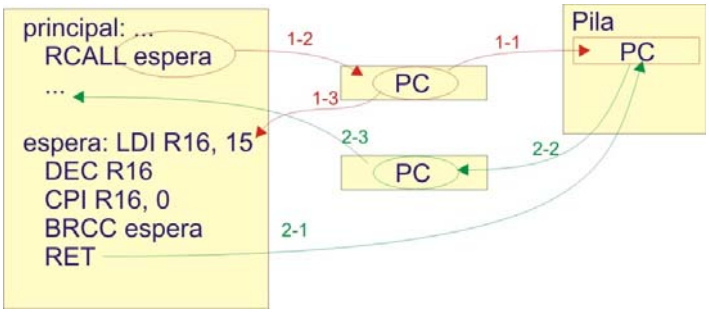

Figura 5.11. Execució instrucció RCALL.

El fet d'emprar una pila permet l'anidament de subrutines. Això és, cada cop que es crida una subrutina es guarda l'adreça de retorn en la pila, i només es recupera amb l'execució de la instrucció RET que va sortint de les rutines anidades.

La figura 5.12 mostra com la pila va guardant les adreces de retorn conforme s'entra més endins en la crida a subrutines i com RET va recuperant les adreces guardades. En tot moment, l'adreça de retorn serà la de la part de dalt de la pila.

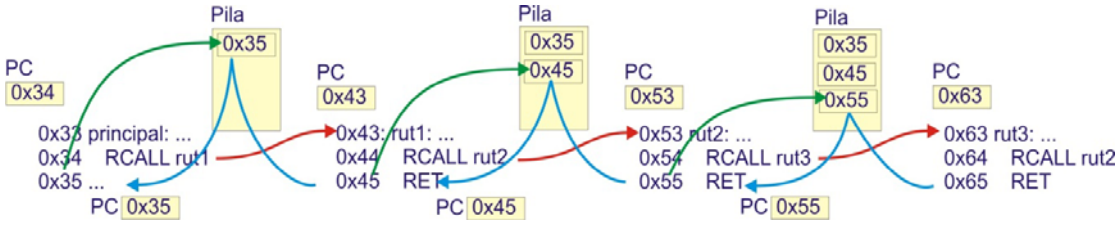

Figura 5.12. Anidament de subrutines: La pila guarda les adreces de retorn que es recuperen en executar *RET*.

La crida a subrutina en EduP12

Quan es treballa amb EduP12 la crida a subrutina la realitza la instrucció RCALL que realitza un salt relatiu des de la posició del comptador de programa. La instrucció RET retorna de la subrutina.

La codificació de la instrucció (veure apèndix A1) mostra que la instrucció RCALL consta de dos camps: el camp d'identificació d'instrucció i el camp de salt. Donat que el camp de salt consta de 12 bits, RCALL pot adreçar tota la memòria (donat que en la implementació actual el PC admet com a màxim 12 bits).

La figura 5.13 mostra la codificació de l'exemple de la figura 5.11, suposant que el programa principal es troba en la posició 0x100 i la subrutina en la posició 0x200.

...	0x100: ...
RCALL espera	0x101: 0101 0010 0000 0000
...	0x102: ...
espera: LDI R16,	0x200: 0011 0001 0000 0000
15	0x201: 0000 0000 0000 1111
bucle: DEC R16	0x202: 0011 0001 0000 1001
CPI R16,	0x203: 0011 0001 0000 0110
0	0x204: 0000 0000 0000 0000
BRCC espera	0x205: 1111 1111 1110 0000
RET	0x206: 0110 0000 0000 0100

Figura 5.13. Codificació de l'exemple presentat en la figura 5.11.

La columna de la dreta mostra el codi màquina d'aquest petit programa. La codificació de cada instrucció, d'acord amb el que s'ha comentat fins ara, es troba a partir de la codificació a baix nivell de cada instrucció (es pot trobar el format de cada instrucció en l'apèndix A1). Es pot observar que les instruccions que operen amb immediats necessiten dues paraules de memòria. També es pot comprovar que la instrucció de salt BRCC, quan es compleix la condició, efectua un salt de -4 posicions. La instrucció que es tracta en aquest apartat és la que s'emmagatzema en la posició 0x101.

La taula 5.3 mostra les operacions que es requereixen en la fase d'execució de la instrucció de salt a subrutina RCALL. En l'execució hi intervenen els registres comptador de programa, apuntador a pila, i la memòria de dades (que guarda la pila).

Cicle	Transferència de dades (en RTL)	Activació de senyals de control
Cicle 1	MAR ← SP, SP ← SP-1	zddMAR←1, lddMAR←1, zA←1, opALU←DEC, ldSP←1
	dRAM ← PC	zdRAM←1, wrdRAM←1
Cicle 2	PC ← PC+k	zA←2, zB←2, opALU←ADD, zPC←0, ldPC←1

Taula 5.3. Fase d'execució de RCALL.

Resumint sobre la subrutina

La subrutina és un programa separat del programa principal que ha de retornar a la instrucció següent a la de la crida. Per tant, la instrucció de crida exigeix guardar el comptador de programes.

La pila s'encarrega de guardar les adreces de retorn. La pila ha de ser una memòria ràpida. En processadors petits es col·loca en memòria ràpida dintre el processador. Normalment, però, la pila forma part de la memòria principal i dintre la CPU hi ha l'apuntador a pila (SP) que indica la capçalera de pila. En aquests casos:

- L'apuntador ens indica l'adreça de la pila on hi hem de posar l'adreça de retorn de subrutina.
- S'ha de portar un control de l'apuntador per evitar sobrepassar la capacitat de la pila.
- Processadors petits i microcontroladors situen la capçalera de pila al final de la memòria, de manera que la capçalera de pila decrementa quan s'introdueix un nou valor.

La pila també serveix per guardar valors temporals durant l'execució d'un programa. Es solen emprar les instruccions PUSH i POP que posen i treuen valors específics de la pila. Aleshores s'ha d'anar en compta de no mesclar dades i adreces de retorn!

5.4. Resum del capítol

Els conceptes fonamentals que s'han introduït en aquest capítol són:

- L'arquitectura de EduP12. S'ha fet especial èmfasis en els elements que composen la CPU del processador.
- S'ha treballat amb el cicle d'instrucció. S'ha vist que el cicle d'instrucció es desglossa en les fases de captació de la instrucció i execució de la instrucció. Que mentre la fase de captació és fixa per a totes les instruccions i necessita de dos cicles de rellotge, la fase d'execució s'ha de personalitzar per a cada instrucció i pot requerir de més d'un cicle de rellotge.
- Lligat amb el cicle d'instrucció s'ha analitzat la transferència de dades entre els registres de la CPU durant el cicle d'instrucció i s'ha introduït el format RTL per visualitzar-ho millor.
- S'ha vist com la unitat de procés i la unitat de control actuen sincronitzadament executant, cicle a cicle, les diferents instruccions emmagatzemades en memòria de programa.
- I s'han introduït diferents instruccions que han permès comprendre el mode de funcionament del processador. És essencial comprendre que cada instrucció té una codificació preestablerta, que la unitat de control descodifica i executa, després, el corresponent cicle d'instrucció.

El proper capítol profunditza en el repertori d'instruccions del processador i en els modes d'adreçament de la CPU.